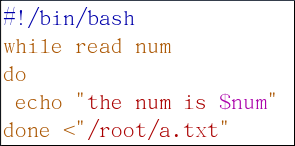
**基础篇**

1. **< >**

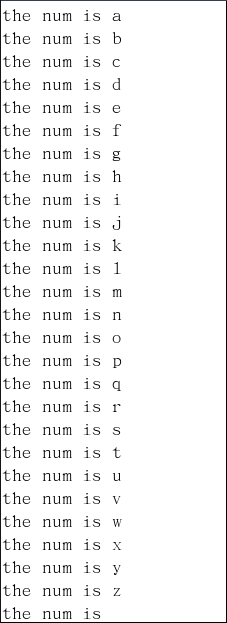
将命令中接收输入的路径由默认的键盘改为指定的文件。给用户aaa设置123456的密码时，我们将123456写在apswd.txt文件，通过”—stdin“的选项来自动为aaa设置密码：passwd –stdin aaa < apswd.txt。”—stdin”选项识别标准输入，”<”是重定向操作符。

重定向输入省略了交互式标准输入过程，脚本运行的程序保护被打断。

重定向输出在使用“while read file”命令读取文件时非常有用，read.sh脚本读取/root/a.txt文件，脚本内容如下：



脚本执行的结果：



**错误重定向 >**

我们习惯使用导入符号（重定向输入）“>”（覆盖）和”>>“（追加）将手动输入的内容或命令执行的结果导入到文件中，错误重定向符号”2>”、“2>>”和“&>”是将命令执行过程中产生的信息（错误或不重要）导入到文件中或丢弃。

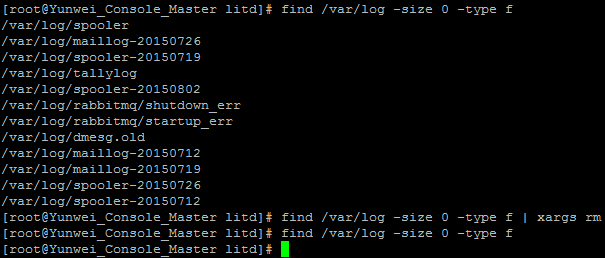
1. **| xargs**

筛选上个命令执行后产生的多个字符串中的字符，我们使用“|”，如grep “^root”、grep “/$”。如果这个命令需要调用上个命令执行的结果，一般使用” | xargs”，如pgrep makewhatis | xargs kill -9”。

实例：

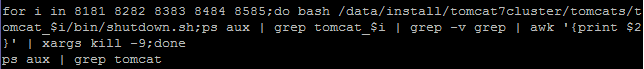
删除/var/log目录下大小为0的文件

find /var/log -size 0 -type f | xargs rm



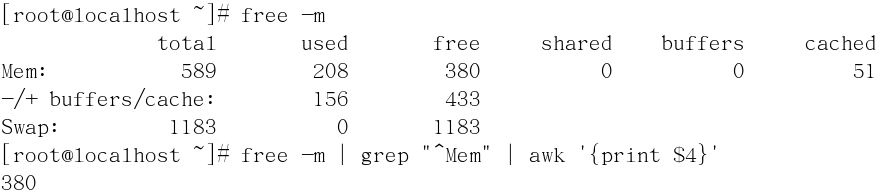
停止tomcat

ps aux | grep tomcat\_$i | grep -v grep | awk '{print $2}' | xargs kill -9



1. **awk**

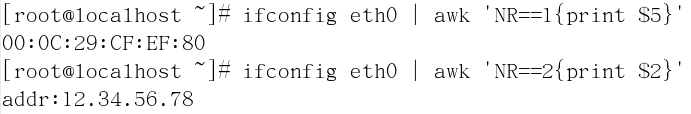
”grep”常用于筛选一行字符串，awk可以筛选出一个字符（串）。“awk ’{print $数字}‘”默认以空格分隔字符串，提取指定字符。用awk的命令查看剩余物理内存：



“awk [-F:/% ] ‘{[if(….)] print $(0-+)}’”花括号外面是单引号，”-F”指定原字符串的分隔符，如/etc/passwd文件的一行中的字符串以“：”为分隔符，大多数字符串以空格隔开，所以默认不用“-F”参数。如果需要对字符串做出判断，使用”if(…)”条件判断语句。”print”输出内容，一行隔开的字符串，从左自右编号依次为0、1、2…，”$（编号）”就是我们要得到的结果。如果要得到多个结果并将它们以自定义的符合隔开，使用‘{print $6”:”$3}’。从/etc/passwd文件筛选出root的PID、用户名（root）、初始化shell，以”-“隔开（原分隔符为”：”）:

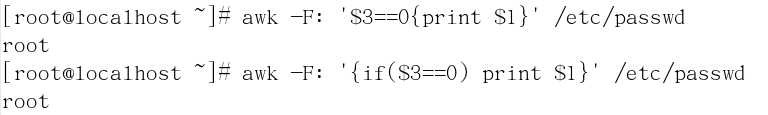


“awk ‘NR==….”选项可以在指定的行中进行筛选，如要提前网卡eth0的MAC地址或IP地址，可以使用awk ‘NR==1{print …}’命令：



其中“NR==1”是根据条件进行筛选的参数，根据条件进行筛选的方式还有：

筛选出UID=0的用户



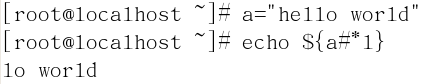
1. **echo : # %**

“:” echo ${a:4}从字符串a的第5个字符向后显示；

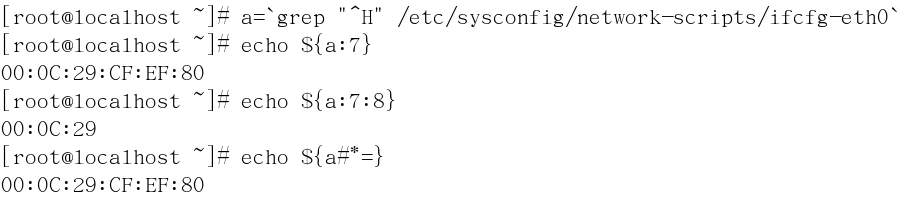
echo ${a:4:3}从字符串a的第5个字符向后显示3个字符；

“#” echo ${#a}统计字符串a的长度

echo ${a#\*A}从左自右，最短匹配，删除“A”在内的以前的字符



截取网卡MAC地址：



1. **test [ ]**

”&&“是”并且“的意思，连接两个命令，前一个命令执行成功后才会执行后面的命令。

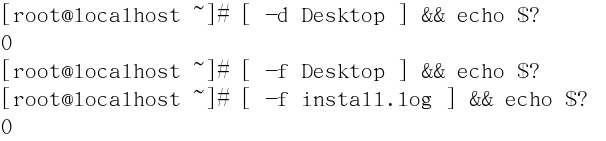
文件

测试一个对象是否为目录： [ -d 对象 ]

测试一个对象是否为存在： [ -e 对象 ]

测试一个对象是否为文件： [ -f 对象 ]

测试当前用户对一个对象是否有读取/写入/执行权限：[ -r/-w/-x 对象 ]

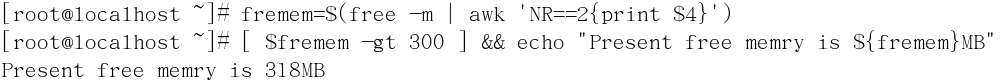


整数值比较

第一个数等于/大于/小于第二个数：-eq / -gt / -lt

第一个数不等于/<==/>==第二个数：-ne / -le / -ge

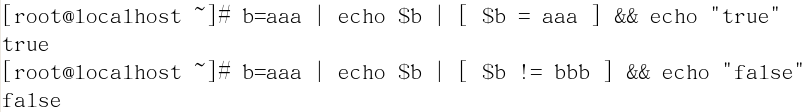
判断当前剩余物理内存是否大于300MB，若大于300MB则显示剩余内存大小



字符串比较

第一个字符串与第二个字符串相同/不相同：= / !=

第二个字符串是否为空：-z

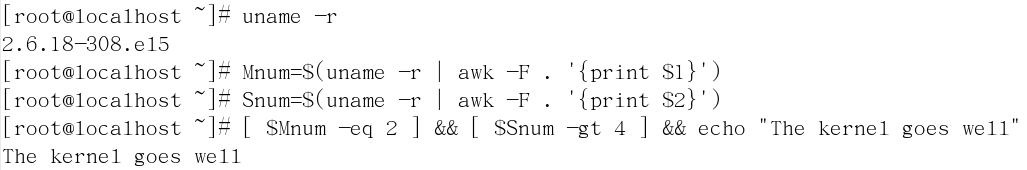


&& ！

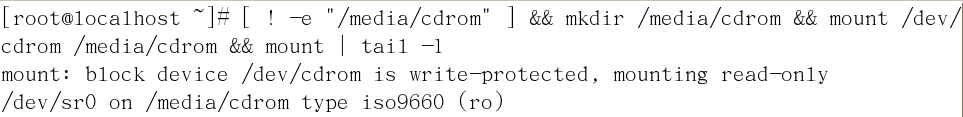
逻辑测试用来判断条件之间的依赖关系，使用时放在不同的测试语句或命令之间。

&& ：逻辑与，”并且“；|| ：逻辑或，”或者“； ！ ：逻辑否，”不“。

判断当前系统内核版本是否大于2.4：



判断/medir/cdrom目录是否存在，如果不存在就建立/medir/cdrom目录，自动挂载光盘镜像，并显示挂载状态



1. **函数（）{}**

函数是拥有名字的代码块。当我们定义代码块，我们就可以在我们的脚本调用函数名字，该块就会被执行。Shell函数概述:在编写Shell脚本程序时，将一些需要重复使用的命令操作，定义为公共使用的语句块，即可称为函数。

定义新的函数：

function 函数名 {

　　命令序列

}

或

函数名() {

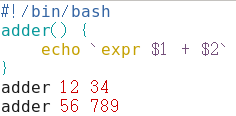
　　命令序列

}

调用已定义的函数：函数名

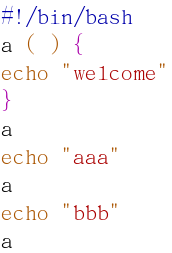
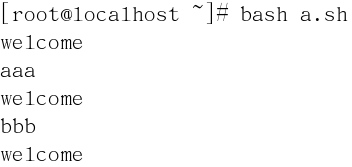
向函数内传递参数：函数名 参数1 参数2 ...

在脚本中定义一个加法函数，用于计算2个整数的和，调用该函数计算（12+34）、（56+789的和：





自定义函数可以看做多个命令的集合，调用某个函数，就是执行这个集合中的所有命令。自定义函数的格式是“函数名称 () { 命令 }”，调用函数的方法是将函数名称独置一行。

脚本执行的结果：

1. **If**
2. 单分支的if语句

if 条件

then 命令

fi

1. 双分支的if语句

if 条件1

then 命令1

elif 条件2 (如果条件1不成立则判断条件2)

then 命令2

else 命令3

fi

1. 多分支的if语句

if 条件1

then 命令1

else 命令2

1. **$? $1 $\* $#**

$? 是系统域定义变量，表示前一条命令执行后的返回状态，“echo $?”返回值为0表示执行正确，返回非0值表示执行出现错误。有时候一条命令被执行了，但命令包含的错误信息，“echo $?”返回值为0只表示命令成功执行了，不能说明命令判断的结果是对的。这时我们可以在if判断语句中“then”或“else”的命令行中追加”exit 1”说明命令判断得到的结果为不理想的，当该if语句执行完成后“echo $?”返回值就为1了。

$1是位置变量，表示用户执行该命令时赋予的第一个对象，$0表示被操作的命令或程序（如ll、cat），$2….$9依次表示当执行命令（或脚本，如ll、cat）时用户指定的操作对象中的第二个….第9个。$0是预定义变量($0、$#、$\*、$?，4个预定义变量都有具体的含义)，$1、$2是位置变量（$1、$2….$9）。位置变量来确定一个命令执行的结果的字符串，常和awk搭配使用。awk默认被筛选对象的多个字符串以空格隔开，被空格隔开的第一个字符即$1、第二个字符即$2…第9个字符为$9。以下命令是对一行以空格隔开的字符串安装指定方式排序：



自定义排序规则需注意，空格、冒号、横杠都必须被双引号括起来。

$\*是系统预定义变量，表示所有位置参数的内容。

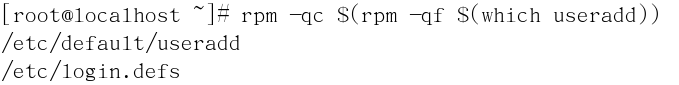
$#是系统预定义变量，表示命令行中位置参数的个数。



1. **反撇号 ` 与 $( )**

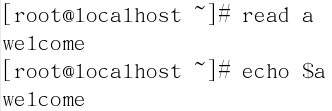
反撇号括起来的对象必须是能执行的命令，将一个命令执行的结果赋值给指定变量、两个命令写在一行具有先后、嵌套的关系时用“ ` 命令` ”。

一行命令嵌套多个命令时反撇号将不适用，“ $() ”更合适。查询提供useradd命令程序的软件包所安装的配置文件路径，使用多个$()



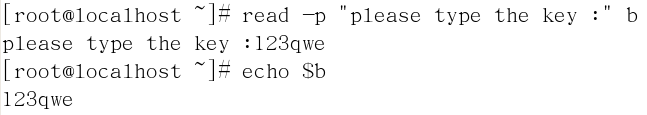
1. **read、 read –p、while read**

echo静态赋值，”read”是交互式赋值命令。“read 变量名称”命令提示用户为该变量赋值：

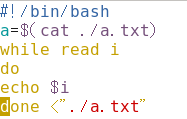


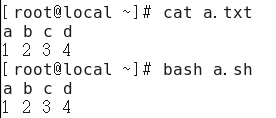
“read “提示信息“ -p 变量”命令产生提示信息、更方便用户输入，将输入的内容赋值给变量





“while read”是使用while循环，以行的形式读取容器a.txt中的内容赋值给指定的变量i





read -p输入时间限制

默认read -p永久等待输入，加入-t选项指定输入限制时间

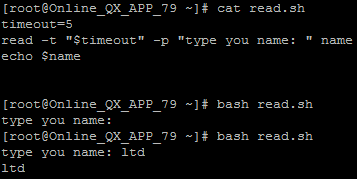
实例代码：

timeout=5

read -t "$timeout" -p "type you name: " name

echo $name

限时5秒无输入自动exit



1. **for while**

for循环语法结构：

for 变量名称 in 容器（读取的对象的集合）

do

命令

done

For语句以空格为分隔符依次读取容器中的对象，将读取的对象赋值为指定变量。

While循环常与数字循环加运算连用

i=1 #赋予变量i的初始值为1

while [ $i –le 100 ] #当i的值小于100时执行循环体

do

命令

let i++ #一次循环完毕让i加1

done

“while read ”以行的形式循环读取

while read 变量

do

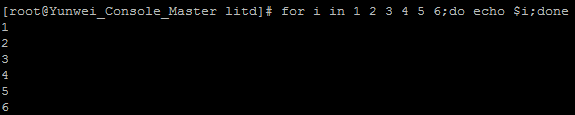
命令

done <”文件路径“

let i++是将变量i依次“加1”，相同的表达式有let "i = i + 1" 或i=$（（$i＋1））

for循环举例：

for i in 1 2 3 4 5 6;do echo $i;done



1. **case分支语句**

case语句的结构

case 变量值 in

模式1）

命令1

；； #表示命令序列的结束

模式2）

命令2

；；

…..

\*) #默认模式，在“service 服务名称 操作”命令中，指定的服务名称即case语句中的变量值，操作即模式，若该操作在该服务的脚本case语句中没有对应的模式，即针对该服务的指令不存在，机会执行”\*)“块的默认命令，如输出帮助信息。

默认命令序列

esac

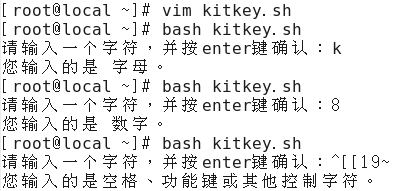
Case分支语句在Linux系统的/etc/rc.d/init.d/（即/etc/init.d)目录下几乎所有的系统服务脚本都使用了，控制系统服务start、stop、restart的命令对应不同的模式。模式字符串中，可以用方括号表示一个连续的范围，如”[0-9]“，还可以用竖杠符合”|“表示或，如”A|B”

以下的脚本通过交互式检查用户输入字符是字母还是数字，或者其他字符



执行脚本，依次输入字母k、设数字8、f8键





1. **break Continue**

break语句:在for、while、until等循环语句中，用于跳出当前所在的循环体，执行循环体后的语句。break即“中断”的意思，用于跳出当前所在的循环体，但是并不退出程序 。执行break命令后将跳到done语句之后，执行done之后的语句，不再执行循环体do…done间的命令。break命令一个简单的用途是退出执行中的循环。我们可以在while和until循环中使用break命令跳出循环。

Continue语句：在for、while、until等循环语句中，用于跳过循环体内余下的语句，重新判断条件以便执行下一次循环。continue即“继续”的意思，用于暂停本次循环，跳转至循环语句的顶部重新测试条件，本次执行过程中continue后的命令序列将被忽略。

continue和break语句都是用在for、while或until循环语句以及case语句中，并且一般都是配合着条件判断语句一起使用。

1. **until**

until：直到”测试条件“成立时终止循环，而while是：当”测试条件“成立时进行循环；until 循环的结构与while命令类似，“until通过检测其后接命令的返回值“$?”来判断是否退出循环，即：until在测试条件为假（非0）时执行循环，条件为真时（0）退出循环，正好与while循环相反

until 条件测试命令

do

命令序列

Done

1. **shift**

用于迁移位置变量，将 $1~$9 依次向左传递。例如，若当前脚本程序获得的位置变量如下：

$1=file1、$2=file2、$3=file3、$4=file4

则执行一次shift命令后，各位置变量为：

$1=file2、$2=file3、$3=file4

再次执行shift命令后，各位置变量为：

$1=file3、$2=file4

通过命令行参数传递多个整数值，并计算总和：

[root@localhost ~]# vi showday.sh

#!/bin/bash

Result=0

while [ $# -gt 0 ]

do

Result=`expr $Result + $1`

shift

done

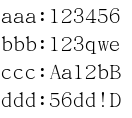
echo "The sum is : $Result"

[root@localhost ~]# ./sumer.sh 12 34 56

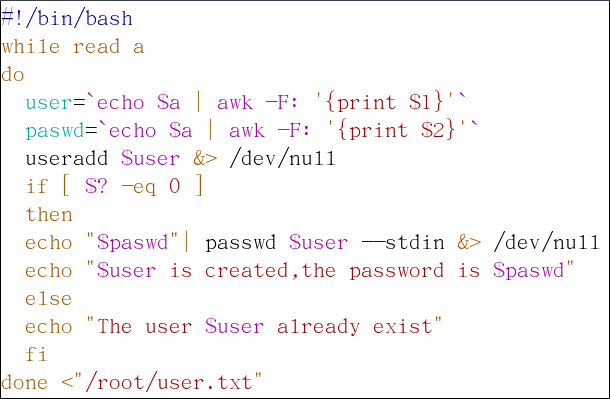
The sum is : 102

1. **批量建用户、设密码**

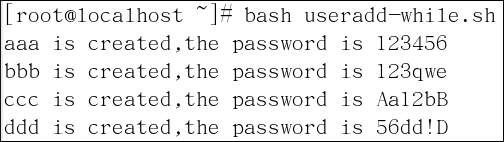
passwd 设置用户密码时“--stdin”参数可以无交互式。useradd-while.sh脚本是首先在/root/user.txt以“用户名：密码”的形式设置用户及密码，然后通过“while read”命令调用user.txt文件



自动建立用户，并设置密码。脚本的内容如下：



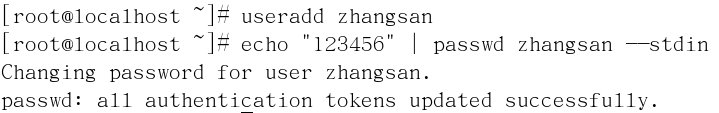
脚本执行的结果：



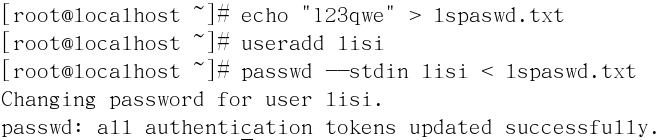
“while read”是指从“done <文件/变量”指定的对象以**行**的形式（不是以空格为分隔符，而是将一行字符串看做一个元素）依次读取赋值为变量“a”，使用“awk”从变量a中筛选字符串，“-F：”是指定原文件user.txt文件中字符串以“：”为分隔符。

取消交互式为用户设置密码，使用管道符“|”或重定向输入符号“<”，方法分别如下：

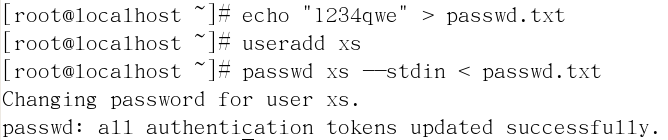
1）echo “密码“ | passwd 用户 --stdin



2）首先将密码保存在lspaswd.txt文件中，passwd --stdin 用户 < lspaswd.txt



或者passwd 用户 --stdin < 密码文件

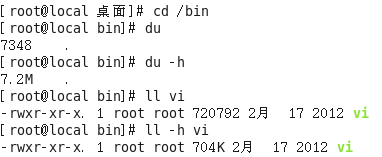


1. **ll ls dd find –size**

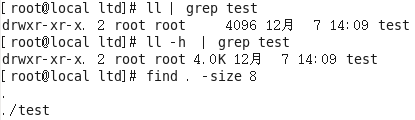
文件类型有七种，它可以从 ls -l命令所列出的结果的第一位看出，这七种类型是：

d 目录。 l 符号链接(指向另一个文件)。 s 套接字文件。 b 块设备文件。 c 字符设备文件。 p 命名管道文件。 - 普通文件，或者更准确地说，不属于以上几种类型的文件。

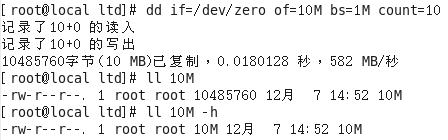
Linux系统使用“ll”命令查看文件大小以**字节B**为单位，“du”命令统计目录大小以**千字节K**为单位；使用“ll –h”查看文件大小以1K=1024B换算，使用“du –h”查看目录大小以1M=1024K换算



Find -size命令查看文件的大小，只用数字时以**块（一块等于5 1 2字节）**为单位统计文件大小



dd命令创建指定大小的空文件：



1. **chmod chown chgrp**

符号模式c h m o d命令的一般格式为：

**chmod [who] operator [permission] filename**

w h o的参数有：u 文件属主权限。 g 同组用户权限。 o 其他用户权限。a 所有用户(文件属主、同组用户及其他用户 )。

o p e r a t o r的含义：+ 增加权限。- 取消权限。= 设定权限。

p e r m i s s i o n的含义： r 读权限。w 写权限。x 执行权限。s 文件属主和组s e t - i d。t 粘性位\*。l 给文件加锁，使其他用户无法访问。u,g,o 针对文件属主、同组用户及其他用户的操作。

\*在列文件或目录时，有时会遇到“ t” 位。“ t” 代表了粘性位。如果在一个目录上出现 “ t” 位，这就意味着该目录中的文件只有其属主才可以删除，即使某个同组用户具有和属主同等的权限。不过有的系统在这一规则上并不十分严格。

绝对模式c h m o d命令绝对模式的一般形式为：**chmod [mode] file**

其中m o d e是一个八进制数，4表示读取权限，2表示写入权限，1表示执行权限。

**c h o w n**命令的一般形式为：**chown -R -h owner file**

- R选项意味着对所有子目录下的文件也都进行同样的操作。 - h选项意味着在改变符号链接文件的属主时不影响该链接所指向的目标文件。

**c h g r p**命令改变文件的属组，格式和c h o w n命令一样。

1. **suid/guid**

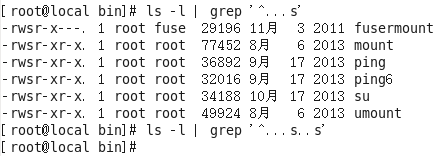
如果root用户的某一个脚本设置了suid/guid的权限，那么其他普通用户在执行它的期间也同样具有该文件所属用户组中root用户的权限。Root用户管理大型的数据库系统，写了几个脚本，并设置了它们的 g u i d，指定的普通用户通过执行这些脚本，他们可以完成数据库备份及其他管理任务，而无须以root的身份登录，在这些脚本运行结束之后，他们又回复到普通用户的权限。

进入 / b i n或/ s b i n目录，执行下面的命令查找s u i d文件：

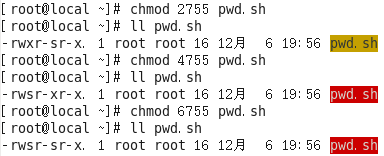
**$ ls -l | grep '^...s'**

查找g u i d文件：

**$ ls -l | grep '^...s..s'**



设置 s u i d，就将相应的权限位之前的那一位设置为 4；设置g u i d，将相应的权限位之前的那一位设置为2；如果希望两者都置位，将相应的权限位之前的那一位设置为6。

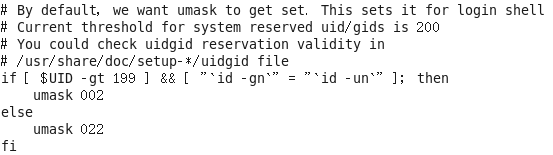


还可以使用符号方式来设置 s u i d / g u i d。如果某个文件具有这样的权限： rwx r-x r- x，那么可以这样设置其s u i d：

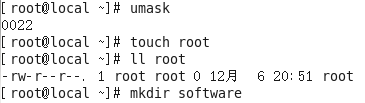
chmod u+s <filename> 于是该文件的权限将变为：rws r-x r-x

1. **umask**

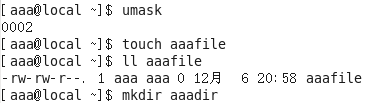
Umask是用户创建文件的缺省模式，可以理解为创建文件是缺省的相反权限。Umask值在所有用户登录系统时都会加载的/etc/profile配置文件中设定：



系统不允许用户创建的文件默认具有可执行权限，却允许创建的目录有可执行权限。因此文件的umask值最大为6，目录的umask值最大为7。属组不同的用户umask值不同：









修改当前用户umask值的方法是：使用“umask 0xx”命令。

1. **vim**

非编辑模式下按u撤销编辑 ctrl+r 反撤销、按一下恢复一下、按u是撤销、一直按u回到你打开这个文件滴最开始状态、且下方有提示！

**当你按u撤销一次操作后，再按ctrl+r就可以恢复到按u前的状态！**

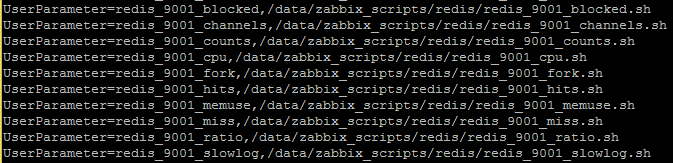
ctrl+r 重复前一个动作

vim –将用在 pipe 里

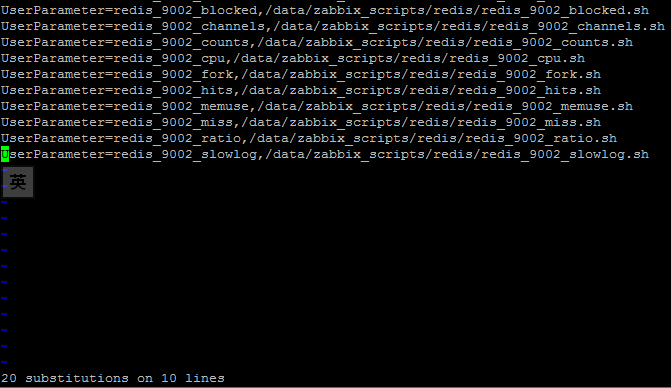


全局替换：，将local字符串全部替换为generic。

从当前行替换到最后一行：：.，$s/原字符串/替换内容/g





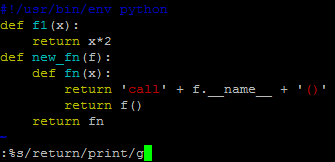


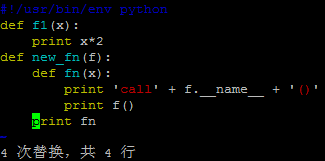
按^跳到行首，按$跳到行尾

按G跳到最后一行

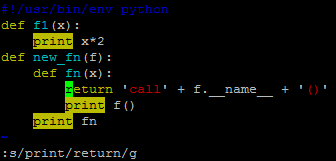
搜索多个内容时，按n翻下一行

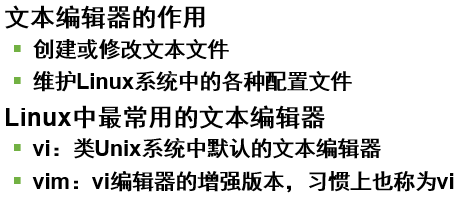
全局替换：:%s/内容/新内容/g





替换当前行：:s/内容/新内容/g





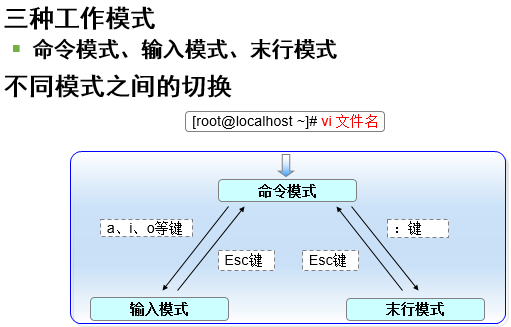
[root@localhost ~]# **alias vi='/usr/bin/vim'**

[root@localhost ~]# **which vi**

alias vi='/usr/bin/vim'

/usr/bin/vim

**设置别名，以便执行vi时自动调用增强版编辑器**



vi编辑器的三种工作模式，相当于图形软件窗口中的不同界面，不同的模式中能够对文件进行的操作也不相同

**1）命令模式**：启动vi编辑器后默认进入命令模式，该模式中主要完成如光标移动、字符串查找，以及删除、复制、粘贴文件内容等相关操作

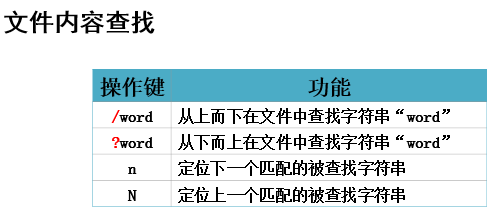
**2）输入模式**：该模式中主要的操作就是录入文件内容，可以对文本文件正文进行修改、或者添加新的内容。处于输入模式时，vi编辑器的最后一行会出现“-- INSERT --”的状态提示信息

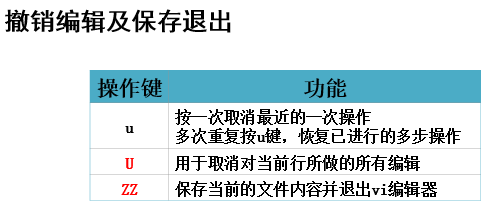
**3）末行模式**：该模式中可以设置vi编辑环境、保存文件、退出编辑器，以及对文件内容进行查找、替换等操作。处于末行模式时，vi编辑器的最后一行会出现冒号“**:**”提示符

**命令模式的基本操作**



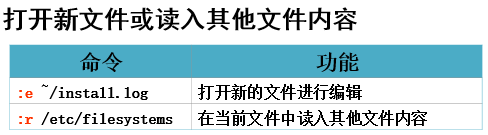


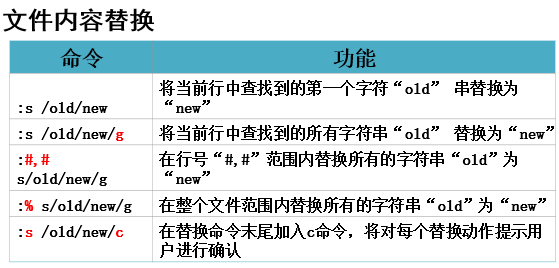




**末行模式的基本操作**







1. **crontab**

Crontab域分为6个列：第1列 分钟1～5 9；第2列 小时1～2 3（0表示子夜）；第3列 日1～3 1；第4列 月1～1 2；第5列 星期0～6（0表示星期天）；第6列 要运行的命令。只支持\* - ,通配符，如每天下午6点到11点每隔15分钟：0,15,30,45 18-23。星期一、星期三、星期五每隔30分钟：0,30 \* \* \* 1,3,5。

设置crontab任务计划的方式有两种：1）以”分钟 小时 日 月 星期 命令/脚本“的格式将”crontab –e”内容保存在一个文件中file中，然后执行“crontab file”命令添加任务计划；2）执行“crontab –e”命令，手动添加任务计划。

每10分钟执行一次/data/install/phpcluster/RedBack.sh脚本

\*/10 \* \* \* \* bash /data/install/phpcluster/RedBack.sh

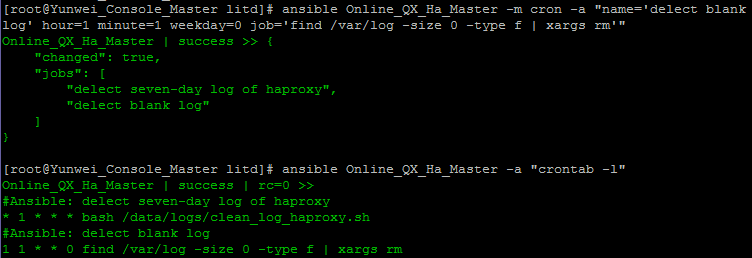
每天的8点、14点、19点、24点执行数据库备份脚本(注：crontab的hour是0-23，0代表24点)

0 0,8,14,19 \* \* \* /home/li475/scripts/backup\_mysql.sh

Cron设置：

1）每周凌晨1点删除所有空日志

设置cron任务计划：ansible Online\_QX\_Ha\_Master -m cron -a "name='delect blank log' hour=1 minute=1 weekday=0 job='find /var/log -size 0 -type f | xargs rm'"



2）每天凌晨1点清理7天前的日志

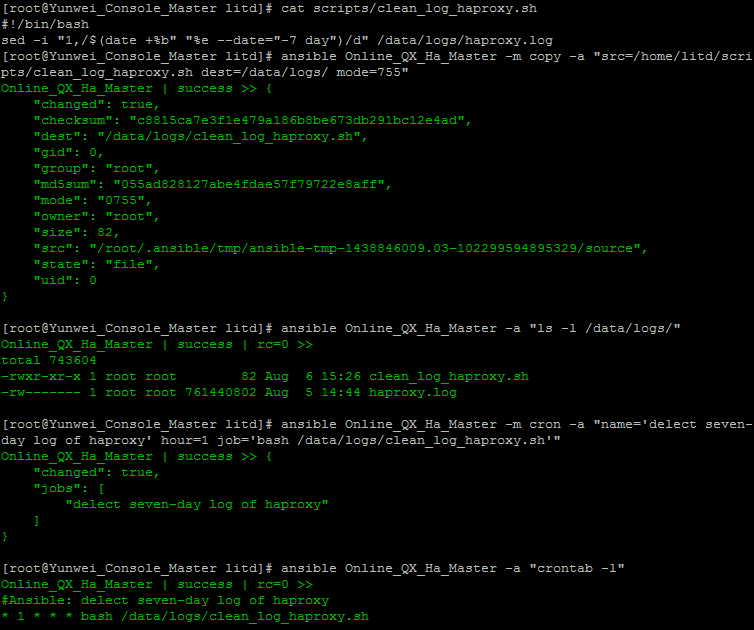
脚本内容：

#!/bin/bash

sed -i "1,/$(date +%b" "%e --date="-7 day")/d" /data/logs/haproxy.log

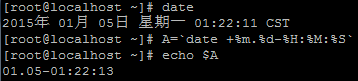
分发：ansible Online\_QX\_Ha\_Master -m copy -a "src=/home/litd/scripts/clean\_log\_haproxy.sh dest=/data/logs/ mode=755"

设置cron任务计划：ansible Online\_QX\_Ha\_Master -m cron -a "name='delect seven-day log of haproxy' hour=1 job='bash /data/logs/clean\_log\_haproxy.sh'"



1. **date**

格式为月.日-小时：分钟：秒



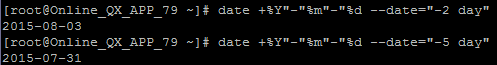
date +%Y%m%d%H%M



date +%Y"-"%m"-"%d



date +%Y"-"%m"-"%d --date="-2 day"



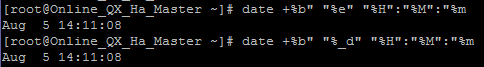
显示系统时间戳

date +%b" "%d" "%H":"%M":"%m

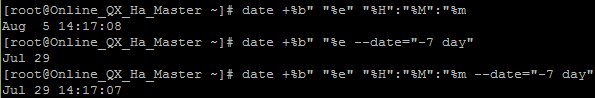


date +%b" "%e" "%H":"%M":"%m

date +%b" "%\_d" "%H":"%M":"%m



date +%b" "%e --date="-7 day"



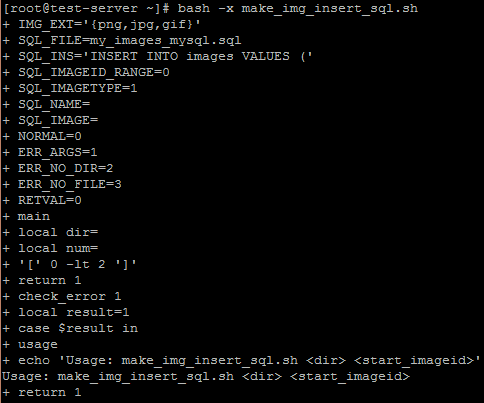
**24. 标准输出和错误输出重定向**

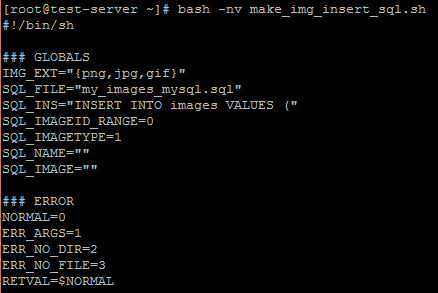
将标准输出和错误输出同时重定向到同一位置



**25. 调试shell脚本**

使用’-x’参数可以调试shell脚本，另一个种方法是使用‘-nv’参数





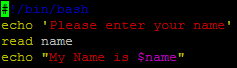
**26. 测试文件**

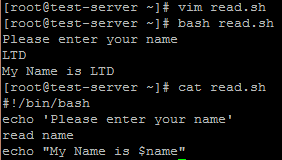
test命令可以用来测试文件。基础用法如下表格：



**27. 让 shell 就脚本得到来自终端的输入**

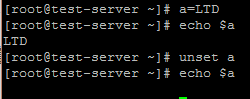
read命令可以读取来自终端（使用键盘）的数据。read命令得到用户的输入并置于你给出的变量中。





**28. unset**

unset”命令用于取消变量或取消变量赋值

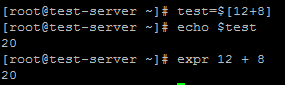


**29. 算术运算**

有两种方法来执行算术运算：

1.使用expr命令

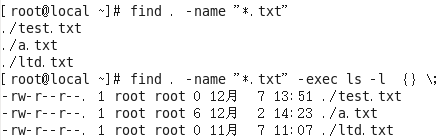
2.用一个美元符号和方括号（$[ 表达式 ]）



**进阶篇**

**1.find**

-exec find命令对匹配的文件执行该参数所给出的 s h e l l命令。相应命令的形式为 ' command' {} \;，注意{ }和\；之间的空格。



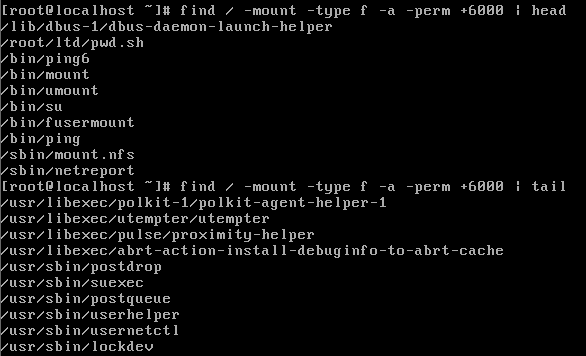
f i n d命令有很多选项或表达式，每一个选项前面跟随一个横杠 -：

-name 按照文件名查找文件。 在当前目录及子目录中查找所有的‘ \*.txt’ 文件:$ find . -name "\*.txt" ；在当前目录及子目录中查找文件名以一个大写字母开头的文件：$ find . -name "[A-Z]\*" ；在/etc目录中查找文件名以host开头的文件:$ find /etc -name "host\*" ；在当前目录查找文件名以两个小写字母开头、跟着是两个数字、最后是“ \* . txt”的文件（如 ax37.txt）：$ find . -name "[a-z][a-z][0--9][0--9].txt"

-perm 按照文件权限来查找文件。如果明确指定文件属主、属组、其他用户的详细权限，使用八进制权限表示法比如：find . –perm 755查找当前目录具有755权限的文件；如果只针对属主或者属组或者其他用户的特定权限查找，在八进制数组前加上“-”，比如：find –perm -007查找当前目录下其他用户具有读、写、执行权限的所有文件。

**find / -mount -type f -a -perm +6000**

查找系统中具有set位权限的文件，“+6000”中000表示不检查文件的基本权限，6表示只要匹配“4”或”2“权限即可

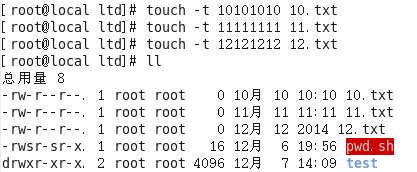


-prune 指出需要忽略的目录。这个命令暂不研究。

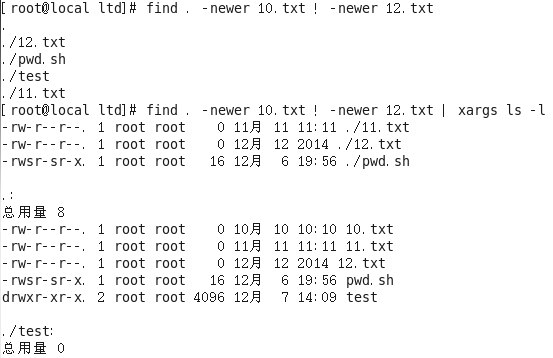
-user 按照文件属主来查找文件。在/etc目录下查找文件属主为uucp的文件：$ find /etc -user uucp；查找无有效属主的文件即属主在/etc/passwd文件中没有有效帐户的文件：$ find /home –nouser； -group 按照文件属组来查找。配合nogroup（查找无有效所属组的文件，即该文件所属的组在 / e t c / g r o u p s中不存在）参数，用法同user、nouser。

-mtime -n +n 按照文件的更改时间来查找文件， - n表示文件更改时间距现在n天以内，+ n 表示文件更改时间距现在 n天以前。在系统根目录下查找更改时间在 5日以内的文件：$ find / -mtime -5 ；在/var/adm目录下查找更改时间在3日以前的文件：$ find /var/adm -mtime +3 。F i n d命令还有- a t i m e和- c t i m e选项，但它们都和- m t i m e选项相似。

-newer file1 ! file2 查找更改时间比文件f i l e 1新但比文件f i l e 2旧的文件。首先我使用“touch -t 时间戳（月日时分） 文件”命令创建指定更新时间的空文件:

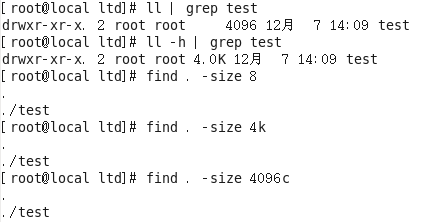


查找在10.txt文件的更新时间和12.txt文件的更新时间之间更新的文件

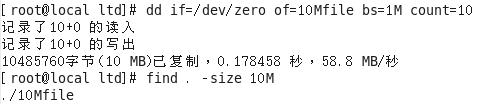


-type 查找某一类型的文件。f普通文件、d目录、l符号链接文件、c字符设备文件b块设备文件、p管道文件。在 / e t c目录下查找所有的目录：$ find /etc -type d –print；在当前目录下查找除目录以外的所有类型的文件：$ find . ! -type d –print；在/ e t c目录下查找所有的符号链接文件：$ find /etc -type l -print

-size n[c] 按照文件长度来查找文件。只用数字n则以块计量文件长度（1块=512B），以字节计量文件长度用 n c（不是b或B），以千字节统计用k（小写k），以M为单位统计用M（大写M）、G(大写G):



用dd命令创建一个10M大小的文件，find的-size跟上M参数：



-depth 在查找文件时，首先查找当前目录中的文件，然后再在其子目录中查找。

-fstype 查找位于某一类型文件系统中的文件，这些文件系统类型通常可以在配置文件/etc/f stab中找到，该配置文件中包含了本系统中有关文件系统的信息。

-mount 在当前的文件系统中查找文件（不进入其他文件系统）。如下例子：

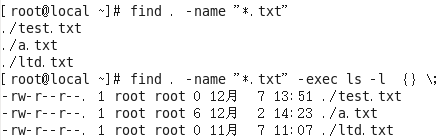


-follow 如果f i n d命令遇到符号链接文件，就跟踪至链接所指向的文件。

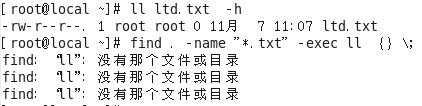
-cpio 对匹配的文件使用cpio命令，将这些文件备份到磁带设备中。

**2.-exec与xargs**

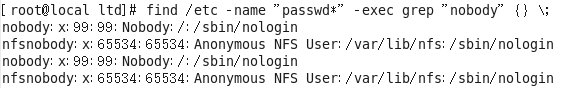
-exec find命令对匹配的文件执行该参数所给出的 s h e l l命令。相应命令的形式为 ' shell' {} \;，注意{ }和\；之间的空格。



但是-exec不能与“ll”命令结合使用



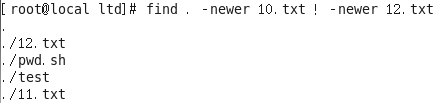
-exec与grep结合使用

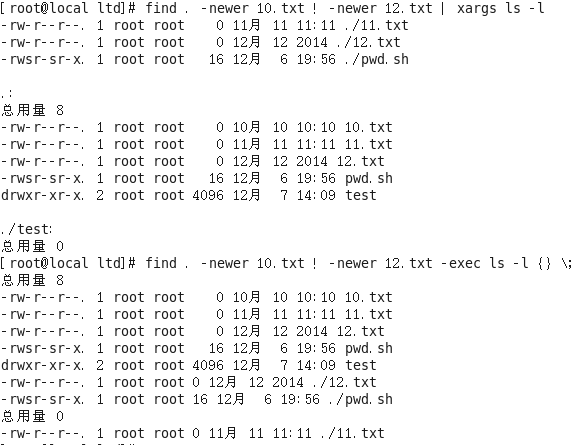


-exec与rm结合使用，”-ok”可代替”-exec”，交互式删除不必要的文件，输nu删除



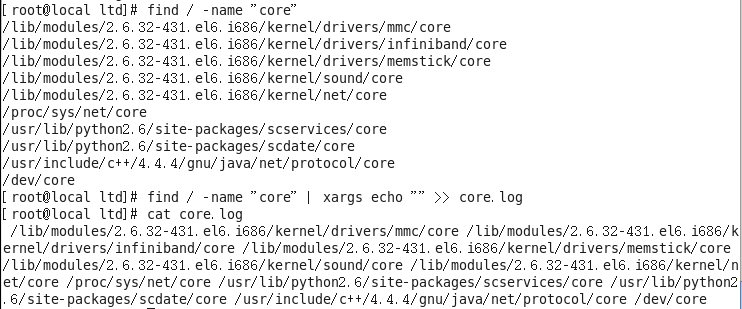
-exec和 | xargs不适用于所有后续shell命令，而且输出内容有所不同：



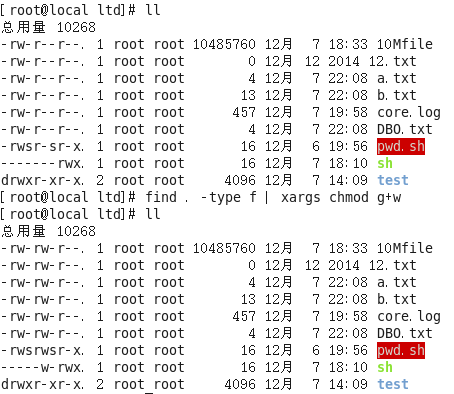


f i n d和-exec同用时，find命令将所有匹配到的文件一起传递给exec执行，-exec选项会为处理每一个匹配到的文件而发起一个相应的进程，并非将匹配到的文件全部作为参数一次执行；这样在有些情况下就会出现进程过多、系统性能下降的问题；而使用 xargs命令则只有一个进程。有些系统对能够传递给 exec的命令长度有限制，这样在 find命令运行几分钟之后，就会出现溢出错误。错误信息通常是“ 参数列太长” 或“ 参数列溢出”。find命令把匹配到的文件传递给 xargs命令，xargs命令每次只获取一部分文件，先处理最先获取的一部分文件，然后是下一批。使用 xargs命令时，究竟是一次获取所有的参数，还是分批取得参数，以及每一次获取参数的数目都会根据该命令的选项及系统内核中相应的可调参数来确定。

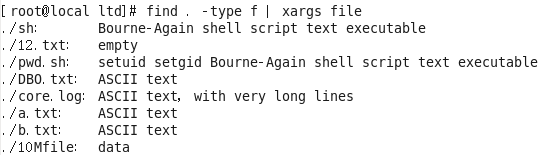
xargs与echo命令同用：



xargs与chmod同用：



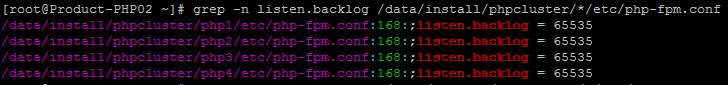
xargs与file命令同用查看一个文件的类型



**3.grep**

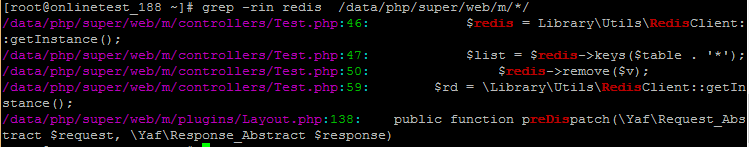
查找文件包含的字符串并打印出行号

grep -n listen.backlog /data/install/phpcluster/\*/etc/php-fpm.conf

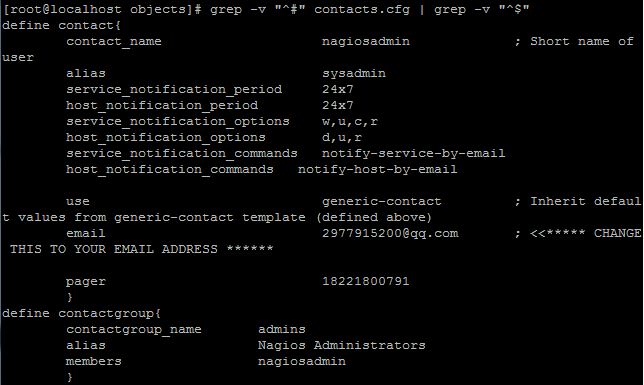


查找一个目录下的所有文件，匹配字符串并打印其所在的行号

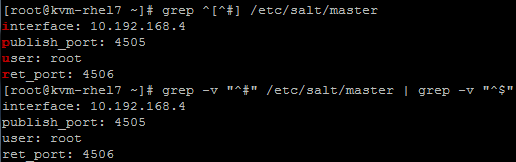
grep -rin redis /data/php/super/web/m/\*/



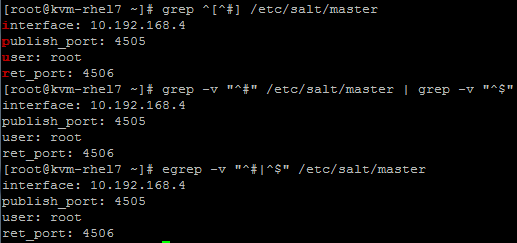
过滤注释行和空行：grep –v “^#” a.txt | grep –v “^$”



还有一个更好的命令：grep ^[^#] /etc/salt/master相当于grep -v "^#" /etc/salt/master | grep -v "^$"

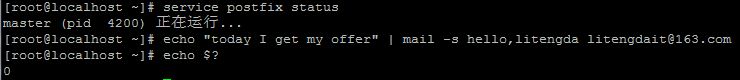


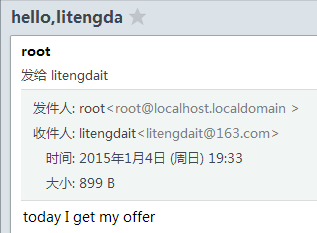
或者egrep -v "^#|^$" /etc/salt/master



**4.mail -s**

Rhel 6.5预安装postfix，首先postfix服务正在运行，输入echo “aaa” | mail –s test [litengdait@139.com](mailto:litengdait@139.com)，即可发送一封标题为test、正文为aaa的邮件。

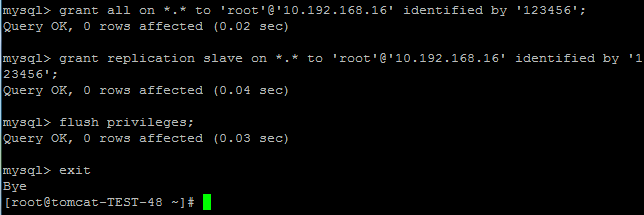




**5.命令行操作mysql – show**

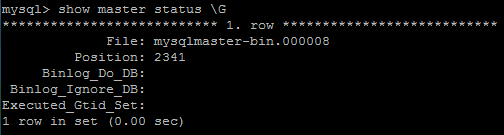
1）查看mysql主服务器的日志文件、position

首先在mysql主服务器授权监控服务器10.192.168.16的root用户对数据库具有replication权限，这里我为了方便，授予用户所有权限



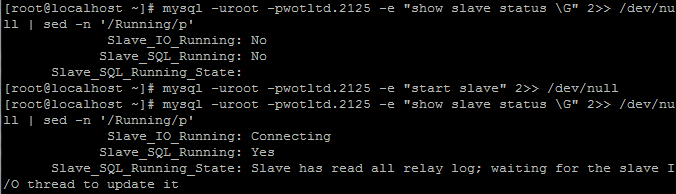
主要命令：mysql –h 10.192.168.62 –uroot –pwotltd.2125 –e “show master status \G” 2>> /dev/null | sed –n ‘2,3p’





2）停止、启动slave服务

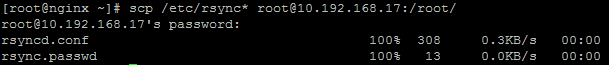
主要命令：mysql –uroot –pwotltd.2125 –e “stop slave”2>> /dev/null



**6.sleep**

在有的shell（比如linux中的bash）中sleep还支持睡眠（分，小时）  
sleep 1 睡眠1秒  
sleep 1s 睡眠1秒  
sleep 1m 睡眠1分  
sleep 1h 睡眠1小时

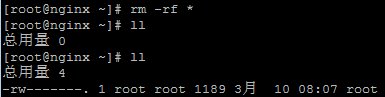
**7.scp**



通过指定ftp端口上传：scp 文件 –p 13579 用户名@IP:目录 或者 sftp文件 –p 13579 用户名@IP:目录

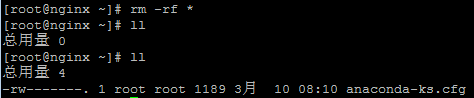
注意目标路径！10.192.168.16：root表示把对象复制为root文件存放在对应家目录：





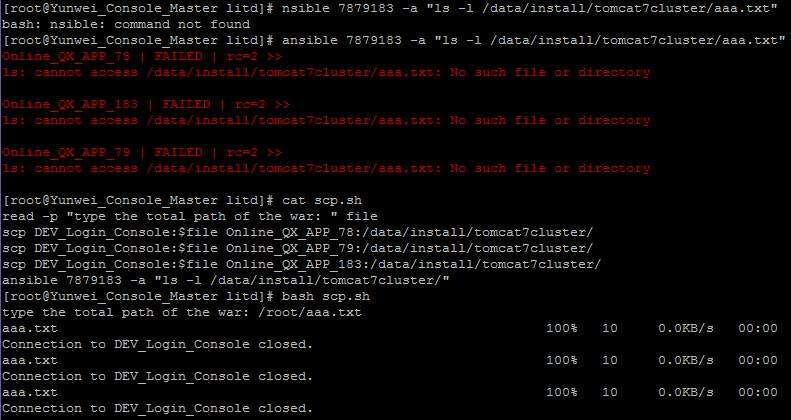
10.192.168.16：/root表示把对象存放在对应root目录保持原文件名：

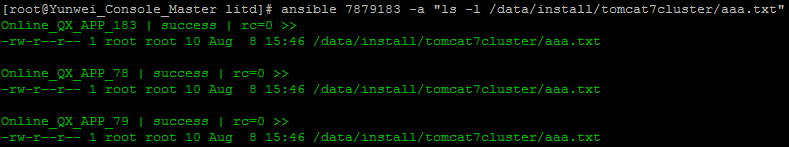




不同主机之间scp

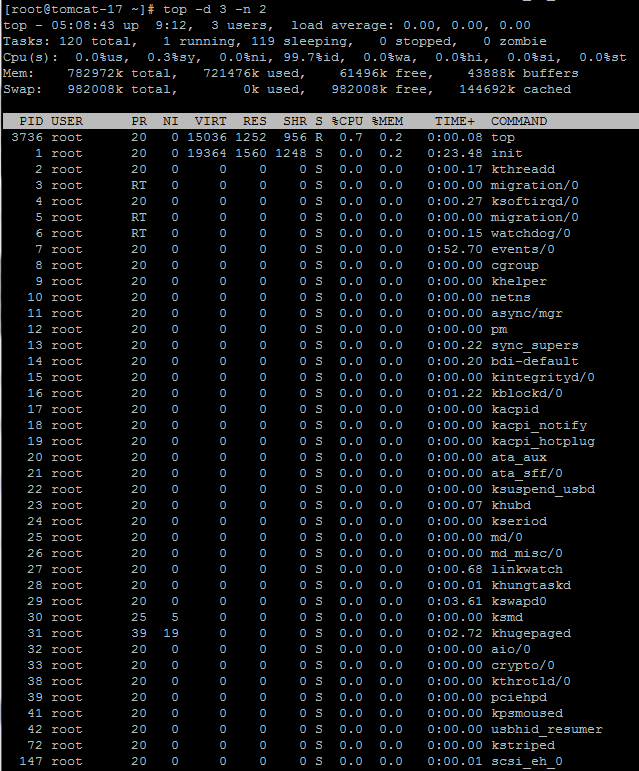
scp DEV\_Login\_Console:$file Online\_QX\_APP\_78:/data/install/tomcat7cluster/





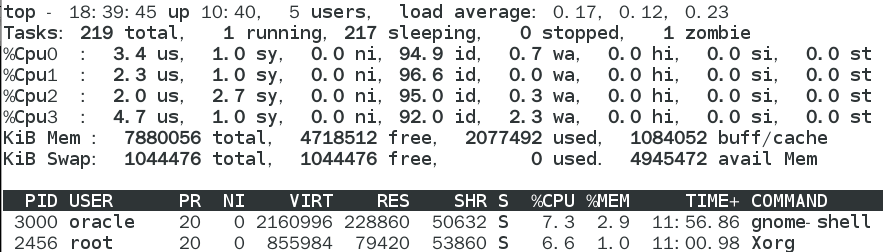
**8.top-僵死进程**

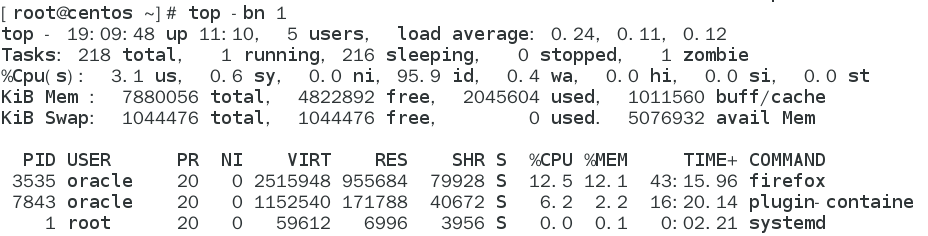
top –d 3 –n 2 隔3秒刷2次

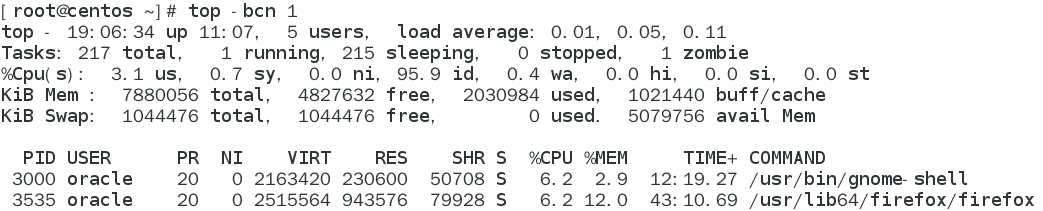


僵死进程的标志是，第8列的“S”是“Z”，“S”表示睡眠。

top进入后，按1可查看CPU的所有核的负载

top -bn 1只统计1次

top -bcn 1详细查看进程执行的命令



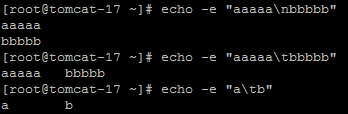
通过top统计每个进程占用的内存 ，单位是MB

for i in `top -bn 1 | grep php-fpm | awk '{print $10}'`;do echo $i\*80580.56/1024 | bc;done



9.**echo \n**

echo –e “…\n…“换行



**10.echo 颜色**

echo可显示黑、红、绿、黄、紫、蓝、白7中基色

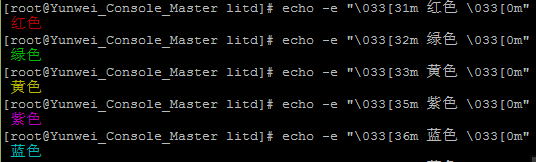
echo -e "\033[31m 红色 \033[0m"

echo -e "\033[32m 绿色 \033[0m"

echo -e "\033[33m 黄色 \033[0m"

echo -e "\033[35m 紫色 \033[0m"

echo -e "\033[36m 蓝色 \033[0m"



这里使用 tput 来配置输出颜色，输出文本，最后再恢复默认输出颜色。如果想对 tpu 了解更多，参看 prompt-color-using-tput 。

#!/bin/bash

NORMAL=$(tput sgr0)

GREEN=$(tput setaf 2; tput bold)

YELLOW=$(tput setaf 3)

RED=$(tput setaf 1)

function red() {

echo "$RED$\*$NORMAL"

}

function green() {

echo -e "$GREEN$\*$NORMAL"

}

function yellow() {

echo -e "$YELLOW$\*$NORMAL"

}

read -p "type a: " a

read -p "type b: " b

read -p "type c: " c

# To print success

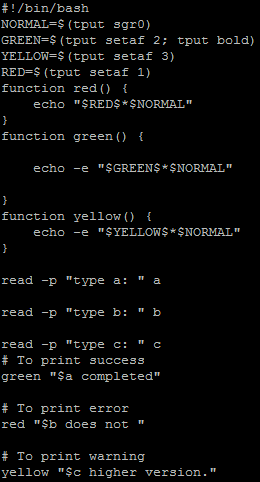
green "$a completed"

# To print error

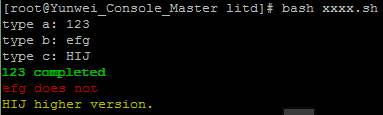
red "$b does not "

# To print warning

yellow "$c higher version."

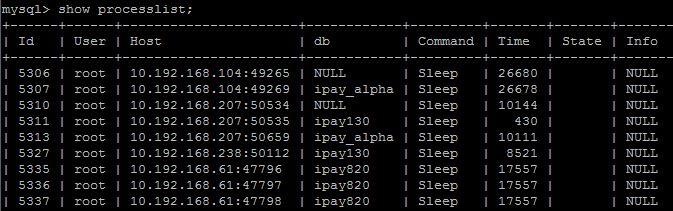


执行结果：



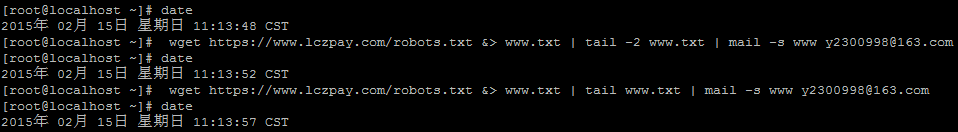
**10.mysql死锁**

show processlist;



**11.wget**

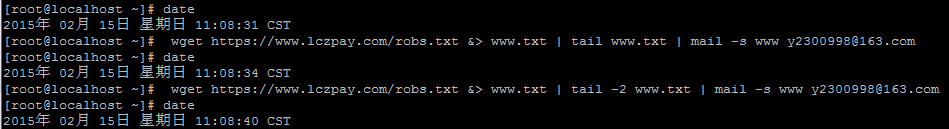
wget <https://www.lczpay.com/robots.txt> &> [www.txt](http://www.txt/) | tail [www.txt](http://www.txt/) | mail –s www y2300998@163.com

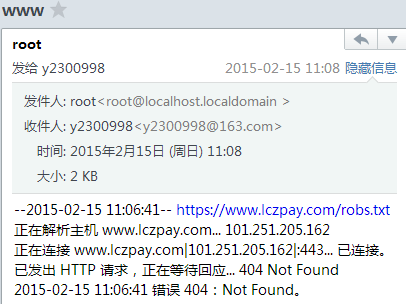


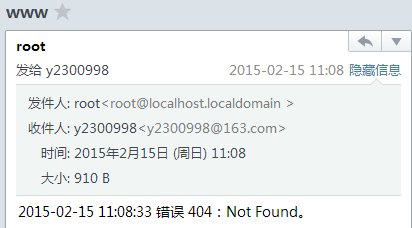




如果网站打不开，收到的邮件会是这样







**12. sort wc uniq**

sort

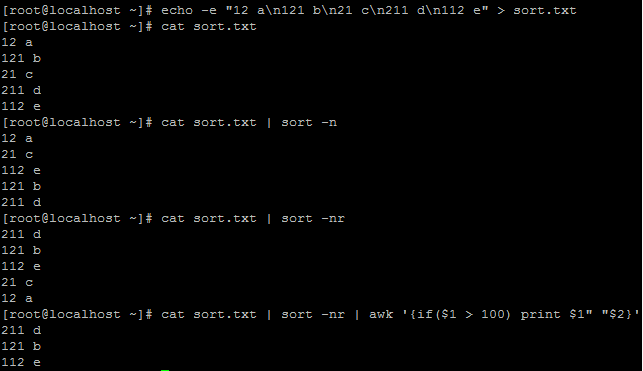
-n 按照数字排序

-r 按照逆序排序

-k 表示排序列

-t 指定分隔符

Sort默认按ASCII码值进行比较 -n 指定分类是域上的数字分类，-r 对分类次序或比较求逆



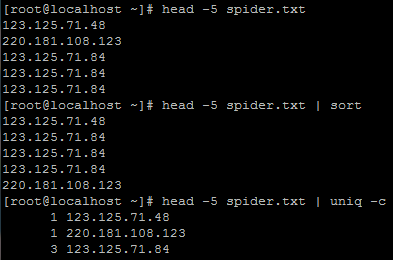
wc

-l 统计文件中行数

-c 统计文件字节数

-L 查看最长行长度

sort按照第1个数字大小排序，uniq –c统计重复出现的次数



uniq

-c 查看该行内容出现的次数

-u 只显示出现一次的行

-d 只显示重复出现的行

**13. expect**

使用expect实现shell自动交互

auto\_smart\_ssh () {

expect -c "set timeout -1;

spawn -noecho ssh-copy-id $2 ${@:3};

expect {

\*assword:\* {send -- $1\r;

expect {

\*denied\* {exit 2;}

eof

}

}

eof

{exit 1;}

}

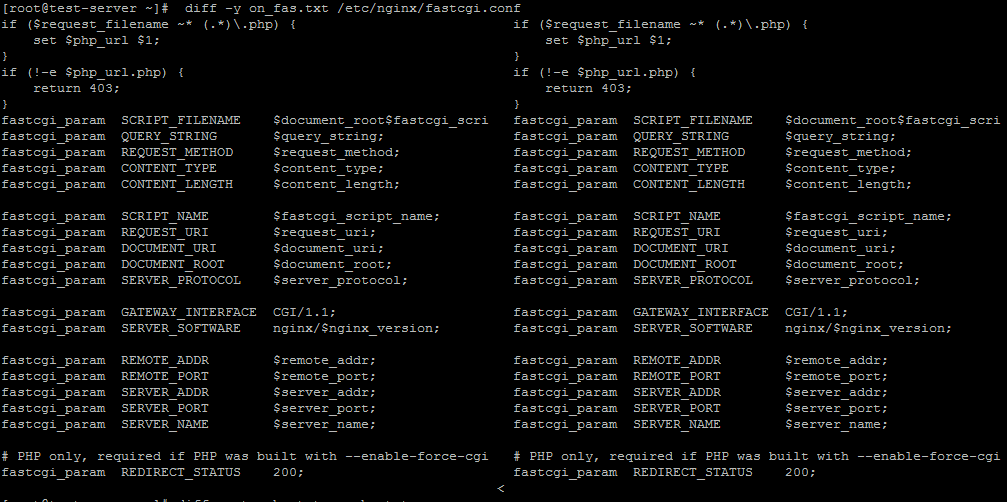
"

#return $?

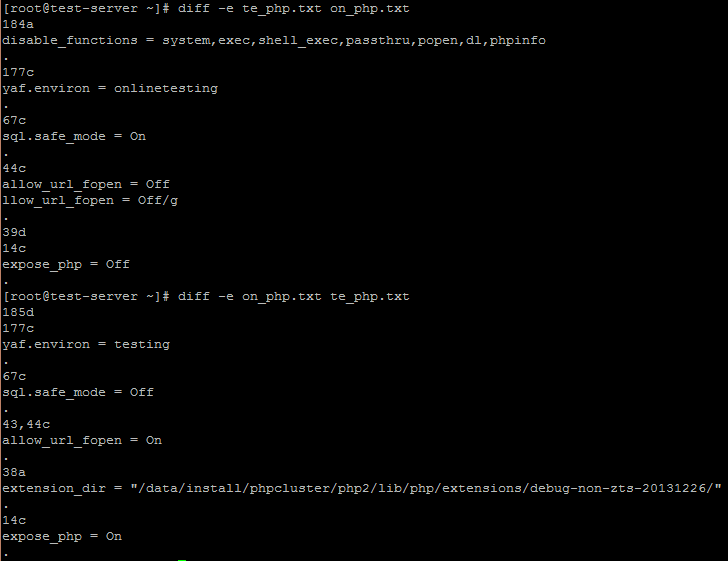
}

**14. diff比较文件差异**

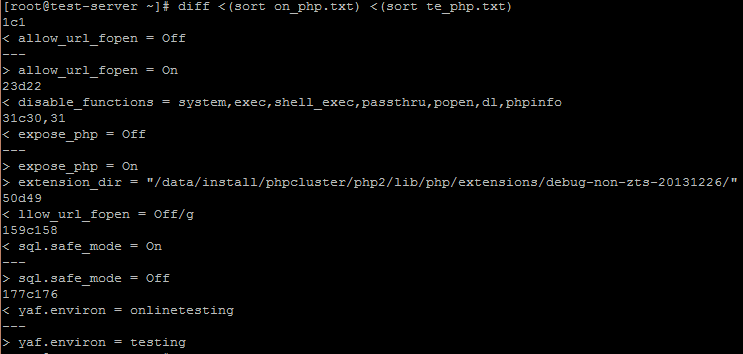
diff -y file1 file2 并排打印文件所有内容



diff -e file1 file2 只显示右边文件file2的不同之处

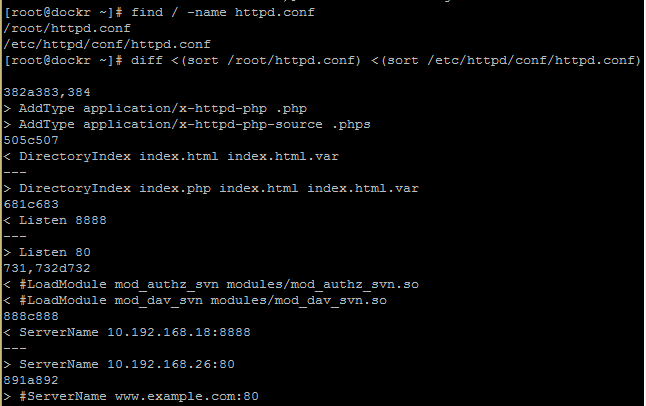


diff <(sort file1) <(sort file2) 以上下文的形式显示左右两边文件的不同之处



比较apache的配置文件，修改前和修改后的差别

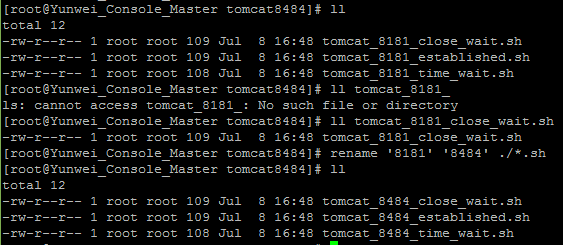
diff <(sort /root/httpd.conf) <(sort /etc/httpd/conf/httpd.conf)



**15.rename批量修改文件名**

格式：rename ‘文件名包含的字符串’ ‘目标字符串’ 文件

rename '8181' '8484' ./\*.sh

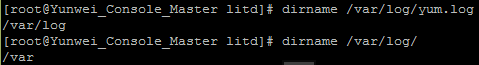


**16.获取文件名、目录名**

dirname可以获取文件、目录的上一级路径

dirname /var/log/yum.log

dirname /var/log/



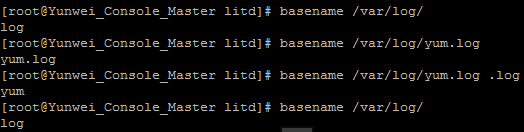
basename可以获取文件的扩展名、去扩展名

basename /var/log/

basename /var/log/yum.log

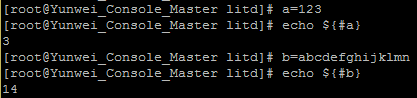
basename /var/log/yum.log .log

basename /var/log/



**17.检查字符串长度**

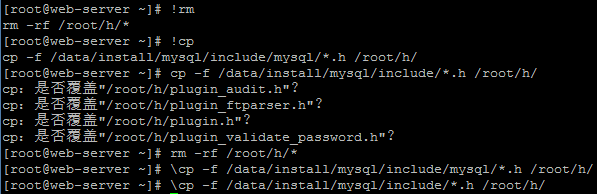
echo ${#变量名称} 可以给出字符串的长度



**18.cp直接覆盖**

\cp -f /data/install/mysql/include/mysql/\*.h /root/h/

\cp -f /data/install/mysql/include/\*.h /root/h/



用了\cp，整个世界都清净了

19.tar打包、解压、查看包文件

总结

1、\*.tar 用 tar -xvf 解压

2、\*.gz 用 gzip -d或者gunzip 解压

3、\*.tar.gz和\*.tgz 用 tar -xzf 解压

4、\*.bz2 用 bzip2 -d或者用bunzip2 解压

5、\*.tar.bz2用tar -xjf 解压

6、\*.Z 用 uncompress 解压

7、\*.tar.Z 用tar -xZf 解压

8、\*.rar 用 unrar e解压

9、\*.zip 用 unzip 解压

10.\*.tar.xz用tar xpvf解压

压缩成gzip文件：

tar -zcvf shell.tar.gz shell/

将gzip文件解压：

tar -zxvf shell.tar.gz

压缩为bz2文件：

tar -jcvf shell.tar.bz2 shell/

将bz2文件解压：

tar -jxvf shell.tar.bz2

如果只想将文件打成tar包，不做压缩，同样很简单：

tar -cvf shell.tar shell/

同样，解压你也就知道了

tar -xvf shell.tar

将当前目录下所有.txt文件打包并压缩归档到文件this.tar.gz，我们可以使用

tar czvf this.tar.gz ./\*.txt

将当前目录下的this.tar.gz中的文件解压到当前目录我们可以使用

tar xzvf this.tar.gz ./

调用格式：tar -选项 文件目录

功能： 对文件目录进行打包备份或解包

选项：

-c 建立新的归档文件

-r 向归档文件末尾追加文件

-x 从归档文件中解出文件

-O 将文件解开到标准输出

-v 处理过程中输出相关信息

-f 对普通文件操作

-z 调用gzip来压缩归档文件，与-x联用时调用gzip完成解压缩

-Z 调用compress来压缩归档文件，与-x联用时调用compress完成解压缩

tar [-cxtzjvfpPN] 文件与目录

参数：

-c :建立压缩文件的参数命令（creat的意思）

-x :解压缩文件的参数命令

-t :查看tar包里文件的命令特别注意，在使用参数时,c/x/t只能有一个，不能同时存在

因为不可能同时压缩与解压缩。

-z :是否同时具有gzip的属性，即是否需要用gzip压缩

-j :是否同时具有bz2的属性，即是否需要用bzip2压缩（记不住的就是它）

-v :压缩过程中显示文件，这个常用，呵基本上我现在每次解压都会看一下里面的文件

-f :使用文件名，之后立即加文件名，不能再加别的参数

-p :使用原文件的原来属性（属性不会根据用户而变），这个从来没用过。。

-P :可以使用绝对路径来压缩

-N :比后面接的日期（yyyy/mm/dd)还要新的才会被打包进新建的文件中

–exclude FILE :在压缩的过程中，不要将FILE打包

20. 利用多核CPU来加速awk, sed, bzip2, grep, wc,make等命令

要想让Linux命令使用所有的CPU内核，我们需要用到GNU Parallel命令，它让我们所有的CPU内核在单机内做神奇的map-reduce操作，当然，这还要借助很少用到的–pipes 参数(也叫做–spreadstdin)。

bzip2是比gzip更好的压缩工具，但它很慢！以前的做法：cat bigfile.bin | bzip2 --best > compressedfile.bz2，现在这样：

cat bigfile.bin | parallel --pipe --recend '' -k bzip2 --best > compressedfile.bz2

尤其是针对bzip2，GNU parallel在多核CPU上是超级的快。你一不留神，它就执行完成了。

GREP

如果你有一个非常大的文本文件，以前你可能会这样：grep pattern bigfile.txt，现在你可以这样：

cat bigfile.txt | parallel --pipe grep 'pattern'

或者这样：

cat bigfile.txt | parallel --block 10M --pipe grep 'pattern'

这第二种用法使用了 –block 10M参数，这是说每个内核处理1千万行——你可以用这个参数来调整每个CUP内核处理多少行数据。

AWK

下面是一个用awk命令计算一个非常大的数据文件的例子。 常规用法：cat rands20M.txt | awk '{s+=$1} END {print s}'。现在这样：

cat rands20M.txt | parallel --pipe awk \'{s+=\$1} END {print s}\' | awk '{s+=$1} END {print s}'

这个有点复杂：parallel命令中的–pipe参数将cat输出分成多个块分派给awk调用，形成了很多子计算操作。这些子计算经过第二个管道进入了同一个awk命令，从而输出最终结果。第一个awk有三个反斜杠，这是GNU parallel调用awk的需要。

WC

想要最快的速度计算一个文件的行数吗？

传统做法：wc -l bigfile.txt

现在你应该这样：

cat bigfile.txt | parallel --pipe wc -l | awk '{s+=$1} END {print s}'

非常的巧妙，先使用parallel命令‘mapping’出大量的wc -l调用，形成子计算，最后通过管道发送给awk进行汇总。

SED

想在一个巨大的文件里使用sed命令做大量的替换操作吗？常规做法：sed s^old^new^g bigfile.txt，现在你可以：

cat bigfile.txt | parallel --pipe sed s^old^new^g

然后你可以使用管道把输出存储到指定的文件里。

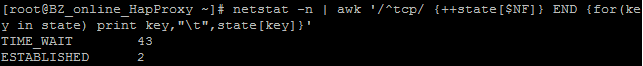
make

make -j `grep -c ^processor /proc/cpuinfo`



21. netstat结合awk详细统计网络状态

netstat -n | awk '/^tcp/ {++state[$NF]} END {for(key in state) print key,"\t",state[key]}'



先来看看netstat：

netstat -n

Active Internet connections (w/o servers)

Proto Recv-Q Send-Q Local Address Foreign Address State

tcp 0 0 123.123.123.123:80 234.234.234.234:12345 TIME\_WAIT

再来看看awk：

/^tcp/

滤出tcp开头的记录，屏蔽udp, socket等无关记录。

state[]

相当于定义了一个名叫state的数组

NF

表示记录的字段数，如上所示的记录，NF等于6

$NF

表示某个字段的值，如上所示的记录，$NF也就是$6，表示第6个字段的值，也就是TIME\_WAIT

state[$NF]

表示数组元素的值，如上所示的记录，就是state[TIME\_WAIT]状态的连接数

++state[$NF]

表示把某个数加一，如上所示的记录，就是把state[TIME\_WAIT]状态的连接数加一

END

表示在最后阶段要执行的命令

for(key in state)

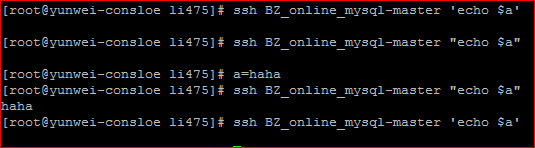
遍历数组

print key,"\t",state[key]

打印数组的键和值，中间用\t制表符分割，美化一下。

22.ssh和ansible变量

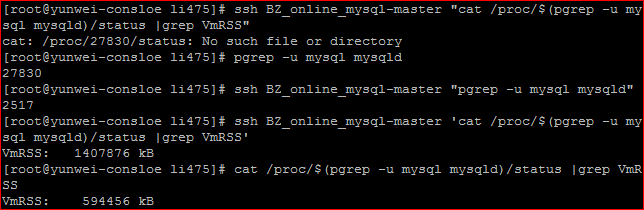
为什么ansible会将远程主机和本地的某些环境变量混淆？那是因为ssh的缘故



在用ssh、ansible执行远程命令带有变量时，不要使用双引号，而是使用单引号

比如，获取BZ\_online\_mysql-master以mysql用户运行的mysqld服务的PID号、然后计算内存

ssh BZ\_online\_mysql-master 'cat /proc/$(pgrep -u mysql mysqld)/status |grep VmRSS'| awk '{print $2/1024}'





1. 匹配字符串并打印所在行号

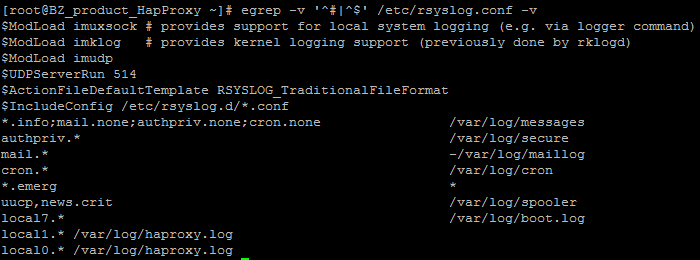
思路：全文匹配，单独用grep、awk、sed肯定会查找全文，将cat -n显示行号，再sed、grep筛选，即可

cat -n xcache.ini | sed -n -e "/protection/"p



1. 去除空行、注释行

egrep -v '^#|^$' /etc/rsyslog.conf -v



1. sort

sort -u 排序后在输出行中去除重复行

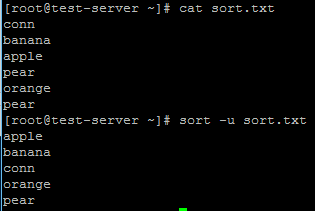
sort -n sort排序数字时默认按字符排序，所以“10”会排在“2”的后面，加上“-n”参数就可以了

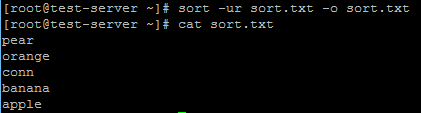
sort -r sort默认的排序方式是升序，-r是按照降序顺序排列

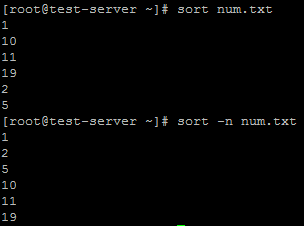
sort -o 和-ru一起使用，将重新排序后的内容，导入到原文件，比使用“>”重定向导入靠谱多了

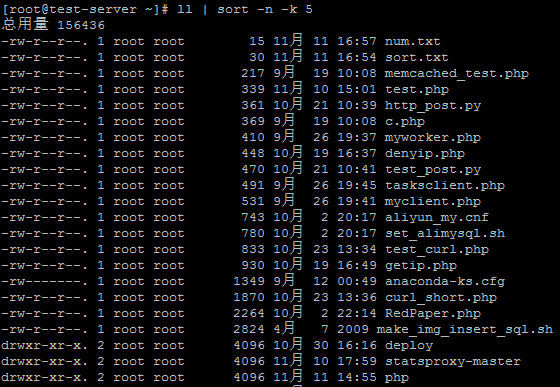
sort -k 参数–k可以指定进行排序的列，默认还是以空格为分隔符。比如要将/root目录下所有的文件按照文件大小（目录的大小都是4096）进行排序，就使用ll /root/ | sort -n -k 5。其中-k 5表示“以空格为分隔符、第5列的数字排序”。

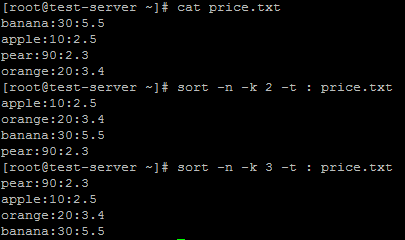
sort -t 和awk -F的作用相同，-t参数可以指定排序对象的分隔符。和-n、-k一起使用，很容易将规律一致但是连续的字符串排序











其他的sort常用选项

-f会将小写字母都转换为大写字母来进行比较，亦即忽略大小写

-c会检查文件是否已排好序，如果乱序，则输出第一个乱序的行的相关信息，最后返回1

-C会检查文件是否已排好序，如果乱序，不输出内容，仅返回1

-M会以月份来排序，比如JAN小于FEB等等

-b会忽略每一行前面的所有空白部分，从第一个可见字符开始比较。

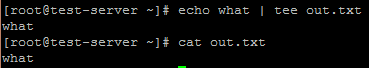
1. 删除软链接

安装mysql时，把mysql的头文件链接到/usr/include目录下，卸载mysql时，怎么删除呢？首先ll /data/install/mysql/include/mysql/ | wc –l确定要删除多少文件，再写个for循环，删除

for i in `ll /data/install/mysql/include/mysql/ | awk '{print $9}' | tail -23`;do rm -rf /usr/include/$i;done

27. tee

tee 就能做到又输出到屏幕上，又同时重定向到文件

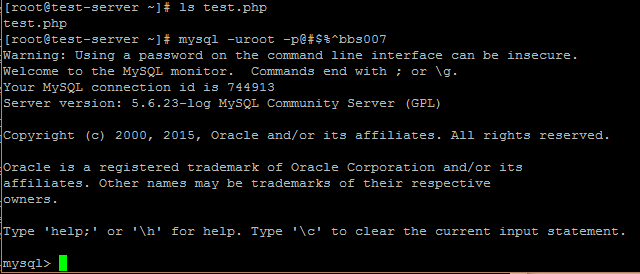


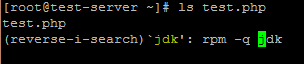
28.<ctrl+r\>

你肯定知道用键盘上下方向键可以来回找你的历史，那么比如有一个很久之前打过的命令你可能需要猛按上才能找到。其实这时候只要输入一部分然后按 <ctrl+r\>bash 就会帮你往回搜索。连续按 <ctrl+r\> 可以按顺序往前搜。(或者直接按 <ctrl+r\> 再进行输入) 。

按ctrl+r输入mysql，会自动弹出曾经用过的关于mysql的命令，直接回车，就执行了该命令





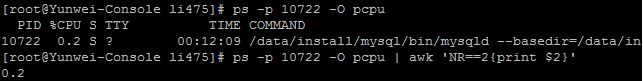


有往前搜索那么一般就有往后搜索。可惜的是往后搜索的快捷键是 <ctrl+s\>，如果你试着按一下的就会发现…好像机器没反应了。这个是因为 <ctrl+s\> 大部分情况下默认是 XOFF，代表暂停接收输入。按 <ctrl+q\> 就能恢复。当然你可以把向前搜索绑定到别的键上，请自行搜索。

29.通过pid获取CPU、内存使用量

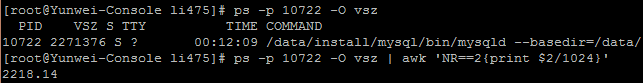
CPU: ps -p PID -o pcpu

ps -p PID -O pcpu | awk 'NR==2{print $2}'

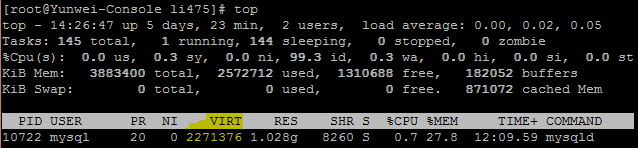


内存：ps -p PID -O vsz

ps -p PID -O vsz | awk 'NR==2{print $2/1024}'



这个内存是虚拟内存，不是实际的使用内存



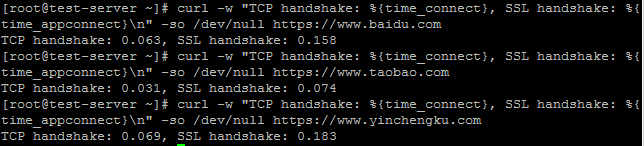
30.curl

curl有一个w参数，可以用来测量TCP握手和SSL握手的具体耗时

curl -w "TCP handshake: %{time\_connect}, SSL handshake: %{time\_appconnect}\n" -so /dev/null 站点

上面命令中的w参数表示指定输出格式，time\_connect变量表示TCP握手的耗时，time\_appconnect变量表示SSL握手的耗时（更多变量请查看文档），s参数和o参数用来关闭标准输出。

从运行结果可以看到，SSL握手的耗时（64毫秒）大概是TCP握手（22毫秒）的三倍。也就是说，在建立连接的阶段，HTTPs链接比HTTP链接要长3倍的时间，具体数字取决于CPU的快慢和网络状况。



[Shell脚本编程的常识](http://mprc.pku.edu.cn/mentors/training/TrainingCourses/material/ShellProgramming.HTM)

**grep专题**

Unix的grep家族包括grep、egrep和fgrep，grep: 默认支持基本正则表达式； egrep: 扩展正则表达式； fgrep: 不支持正则表达式元字符，搜索字符串的速度快。

grep的常用选项

-color=auto export GREP\_COLOR=’01;36′

-v: 反向选取，只显示不符合模式的行；

-o: 只显示被模式匹配到的字串，而不是整个行；

-i: 不区分字符大小写；

-A #：显示匹配到的行时，顺带显示其后面的#个行；

-A 2 -B #：前面的#行；

-C #：前后的#行；

-E: 使用扩展的正则表达式 grep -E = egrep

grep正则表达式基本元字符集

元字符：不表示字符本身的意义，而用于额外功能性的描述

^：锚定行首的符合条件的内容

# grep “^root” /etc/passwd //搜索以root开头的行

$: 锚定行尾的符合条件的内容

#grep “bash$” /etc/passwd //搜索以bash结尾的行

.: 匹配任意单个字符

#grep “s.l” 文件 匹配s和l之间有单个任意字符的内容

\*：匹配紧挨在其前面的字符任意次(包含0次)

#grep “s\*k文件 匹配k前面有0个或任意个s

[]:匹配指定范围内的任意单个字符

#grep “[Dd]..n” 文件 匹配D或d中间包含两个任意字符并以n结尾的行

[^]:匹配指定范围外的任意单个字符

#grep “[^A-Z].\*a” text –color=auto //匹配任意一个非大与字母后面面跟0个或多个任意字符以a结尾的行

\?: 匹配紧挨在其前面的字符0次或1次；

#grep “a\?b” 文件 匹配b之前有0个或1个a

#grep “a.\?b” 文件 匹配a和b之前有0个或1个任意字符

\{m,n\}: 匹配其前面的字符至少m次，至多n次；

\{0,n\}: 至多n次；0-n次；

\{m,\}：至少m次 \{m\}: 精确匹配m次；

#grep “a{1,2\}b” 文件 匹配b 前面的a 至少1次至多2次

\<: 锚定词首，用法格式：\<pattern \b: \bpattern

#grep “\<root” 文件 搜索以root为词首的行

\>: 锚定词尾，用法格式：pattern\> \b: pattern\b

#grep “root\>” 文件 搜索以root为词尾的行

\<pattern\>:锚定单词

#grep “\<root\>” 文件 搜索包含root单词的行

\(\): 分组，用法格式: \(pattern\)

\1 :可以引用第一个分组匹配到的内容 \2 :可以引用第二个分组匹配到的内容

#grep “\([Rr].D\).\*\1er” 文件 搜索R或r和d之间出现两个任意字符而后又跟0个或多个任意字符 ，并在其后引用匹配到的内容

字符集合

[:digit:] 所有数字 0-9

[:lower:] 所有小写字母 a-z

[:upper:] 所有大写字母 A-Z

[:punct:] 所有标点符号

[:space:] 表示空格或tab键

[:alpha:] 表示所有字母（包含大小写）a-zA-Z

[:alnum:] 表示所有字母和数字 a-zA-Z0-9

注：非需要这样表示[^[] ]， 如[[：space :]]表示空格 [^[：space：]] 表示非空

#grep “^[[:space:]]\{1,\}” 文件 搜索/boot/grub/grub.conf 以空格开头的行

**awk专题**

**参考：**[**AWK入门指南**](http://awk.readthedocs.org/en/latest/chapter-one.html)

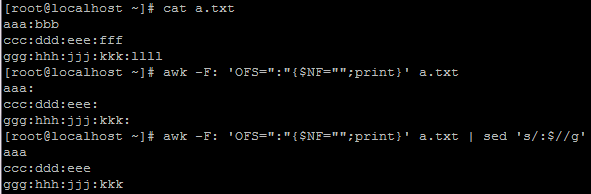
**1.awk截取、删除最后一列**

截取最后一个字段awk ‘{print $NF}’



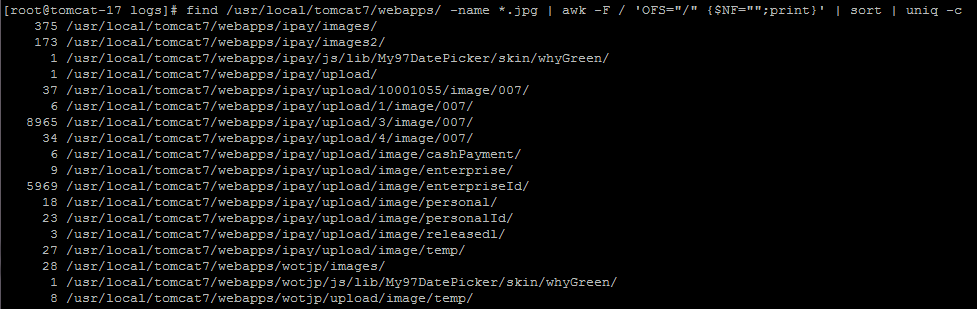
删除最后一列awk -F: 'OFS=":"{$NF="";print}'

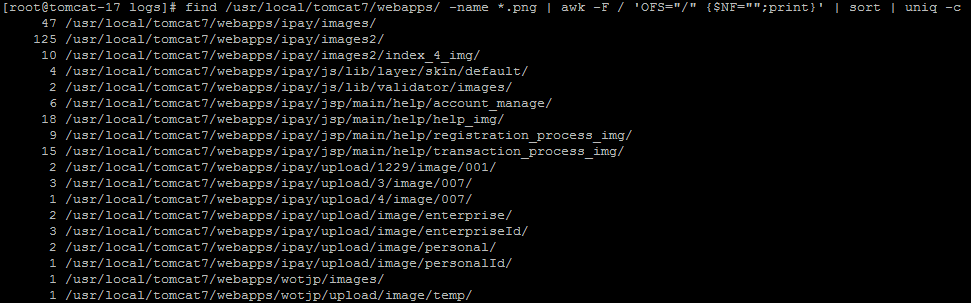
这里需要指定输出分隔符OFS，否则执行完命令以后输出分隔符就成为了空格

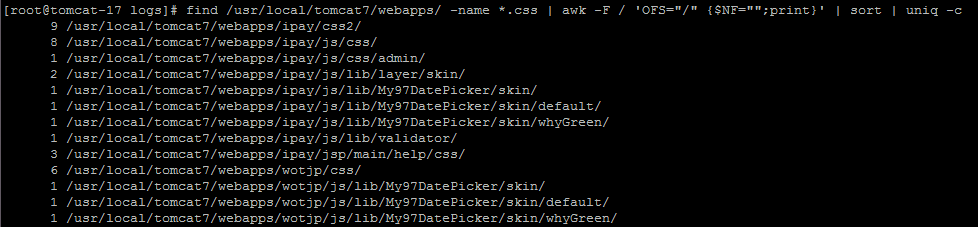


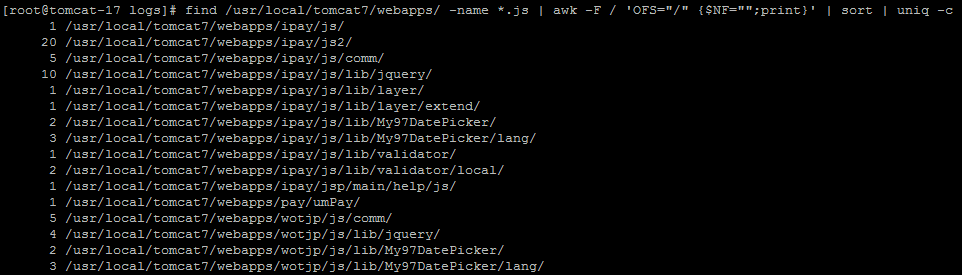
sed 's/:$//g'删除最后的分隔符

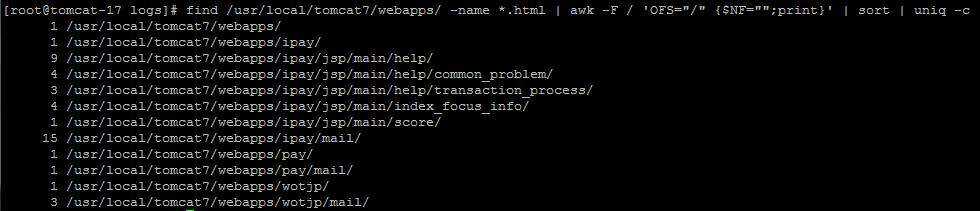
往往源代码项目目录极其子目录有很多jpg、png、css、js文件，nginx写localtion时可以使用这个方法获得root路径：



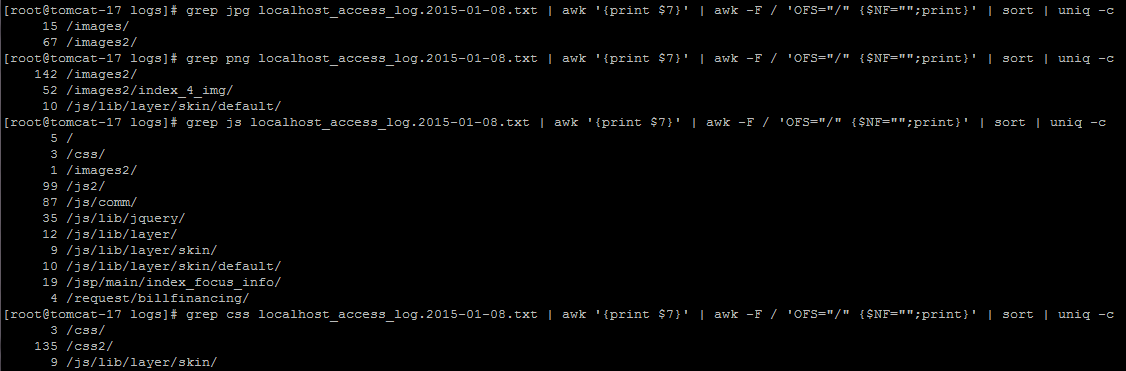






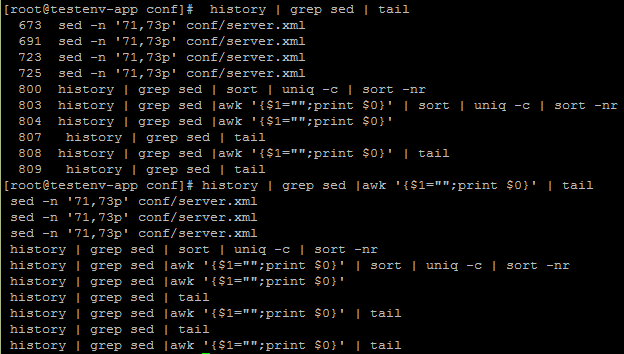


从访问日志查看



**2.awk删除第一列**

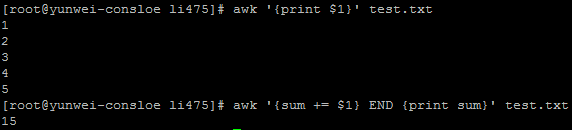
awk '{$1="";print $0}'



3.awk计算

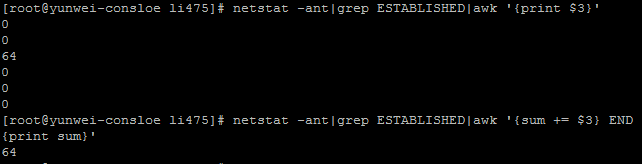
第一列数字求和

awk '{sum += $1} END {print sum}'



统计TCP连接状态为ESTABLISHED的Send-Q总数

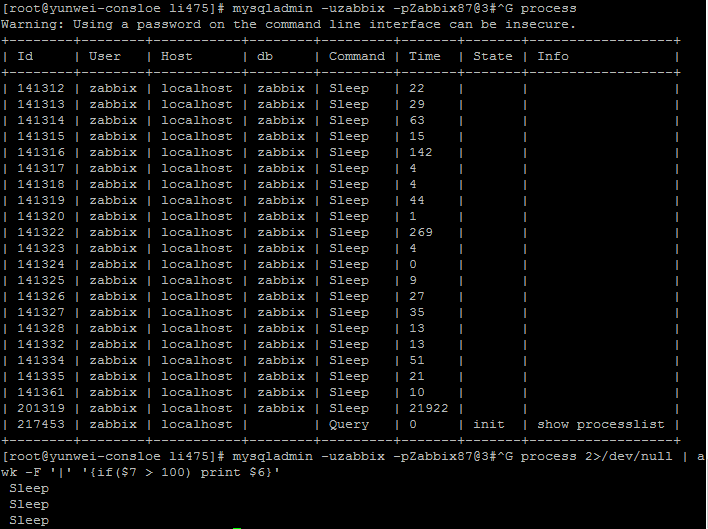
netstat -ant|grep ESTABLISHED|awk '{sum += $3} END {print sum}'



4.awk 比较数字大小

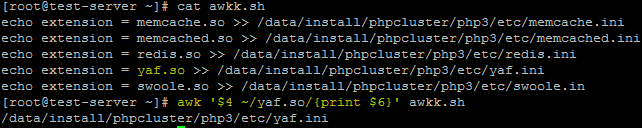
打印出mysql线程时间超过300s的状态

mysqladmin -uzabbix -pZabbix87@3#^G process 2>/dev/null | awk -F '|' '{if($7 > 100) print $6}'



5.awk比较字符串

awk '$4 ~/指定字符串/{print $6}'

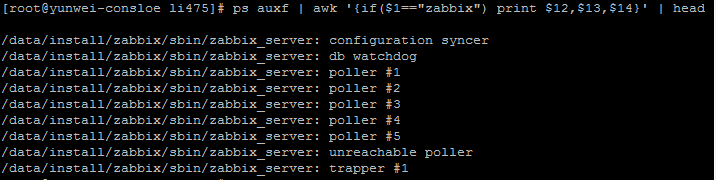


其中，字符串相等用“~”，不相等用“!~”，对比内容用“/ /”包括

参考：[玩转awk](http://mos1989.blog.51cto.com/4226977/1102445)

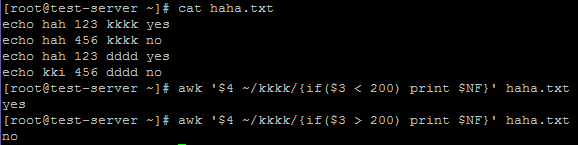
用if也可以啊

ps auxf | awk '{if($1=="zabbix") print $12,$13,$14}' | head



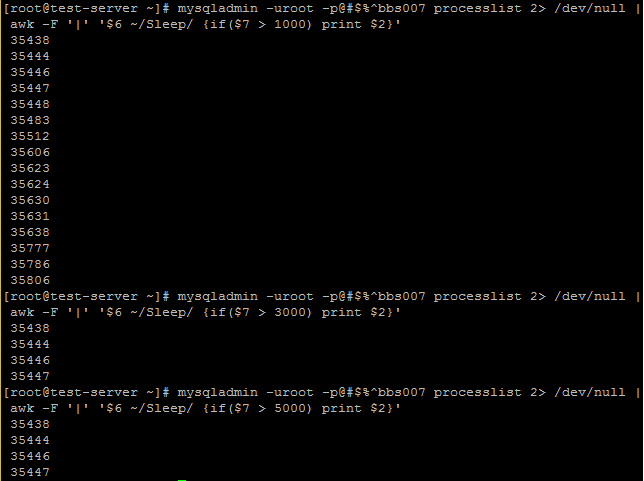
同时比较字符串是否相同、数字是否相等，用~、if组合：

awk '$4 ~/kkkk/{if($3 < 200) print $NF}' haha.txt



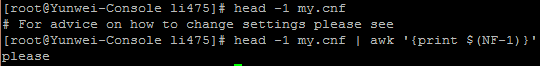
统计数据库的线程状态处于Sleep且时间大于1000s

mysqladmin -uroot -p@#$%^bbs007 processlist 2> /dev/null | awk -F '|' '$6 ~/Sleep/ {if($7 > 1000) print $2}'

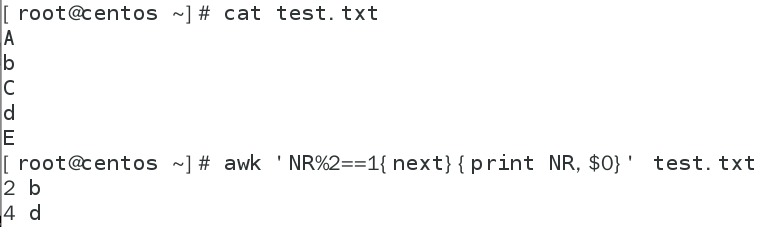


6.awk截取倒数第二列

awk '{print $(NF-1)}'



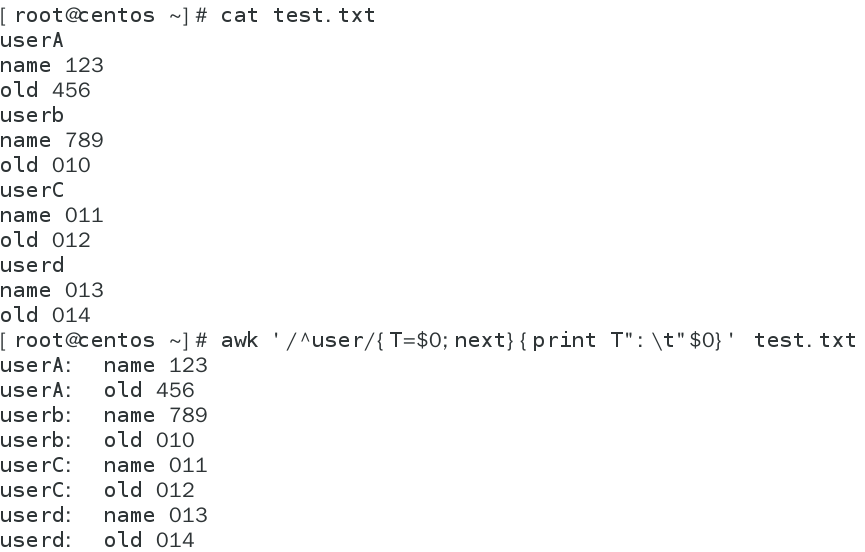
7.awk取偶数行

awk 'NR%2==1{next}{print NR,$0}' test.txt

当记录行号除以2余 1，就跳过当前行。下面的print NR,$0也不会执行。 下一行开始，程序有开始判断NR%2 值。这个时候记录行号是：2 ，就会执行下面语句块：'print NR,$0'

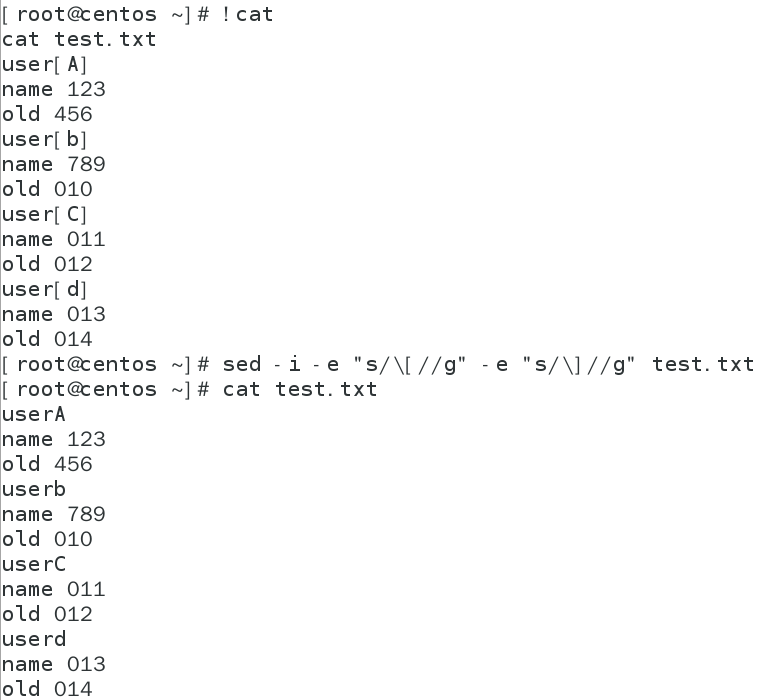
8.awk表格分析

awk '/^user/{T=$0;next}{print T":\t"$0}' test.txt

 遇到以user开头的行，就越过它直奔下一行，使用：作连接符，合 并以user开头的行和不以user开头的行

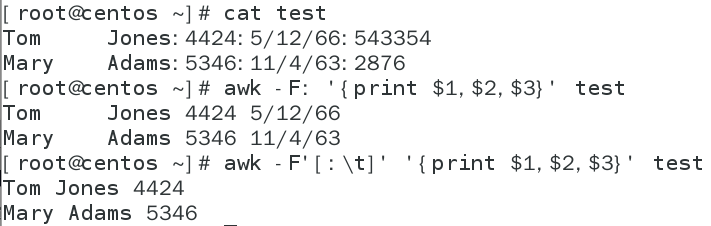
9.同时删除 [ 和 ] 字符

sed -i -e "s/\[//g" -e "s/\]//g" test.txt



10.awk使用多个字段分隔符

awk -F'[:\t]' '{print $1,$2,$3}' test



-F选项后面跟了一个位于方括号中的正则表达式，当遇到空格、冒号或制表符时，awk会把它当成字段分隔符。这个表达式两头加了引号，这样就不会被shell当成自己的元字符来解释(注意， shell使用方括号来进行文件名扩展)。

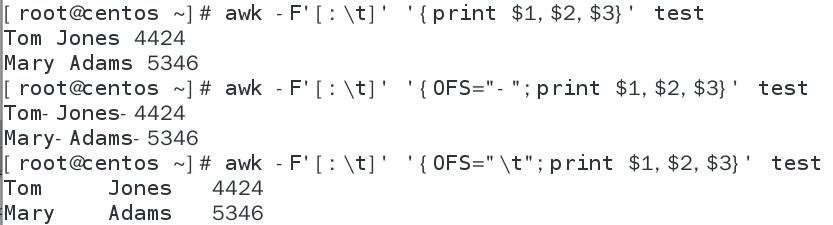
默认的输出字段分隔符是单个空格，被保存于awk的内置变量OFS中。此前的所有例子中，我们都是用[print语句](http://www.linuxawk.com/jiaocheng/73.html)把输出打印到屏幕上。因此，无论OFS如何设置，print语句中用于分隔字段的逗号，在输出时都被转换成OFS的值。如果用OFS的默认值，则$1和$2之间的逗号会被转换为单个空格，print函数打印这两个字段时会在它们之间加一个空格。

如果没有用逗号来分隔字段，则输出结果中的字段将堆在一起.

11.awk自定义输出分隔符

使用OFS可以自定义输出字段的分隔符，注意一定要有逗号‘，’

awk -F'[:\t]' '{OFS="-";print $1,$2,$3}' test

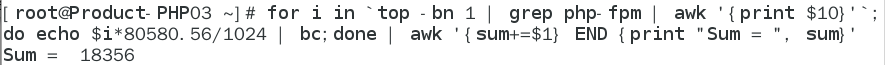


12.awk求和

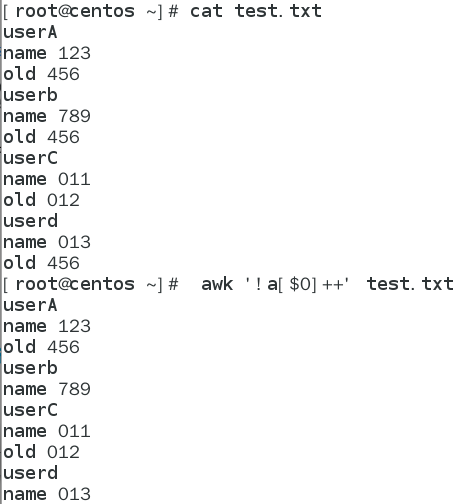
所有的行的$1都是数值，awk累加所有的值后给出总和

for i in `top -bn 1 | grep php-fpm | awk '{print $10}'`;do echo $i\*80580.56/1024 |

bc;done | awk '{sum+=$1} END {print "Sum = ", sum}'

13.awk删除重复行

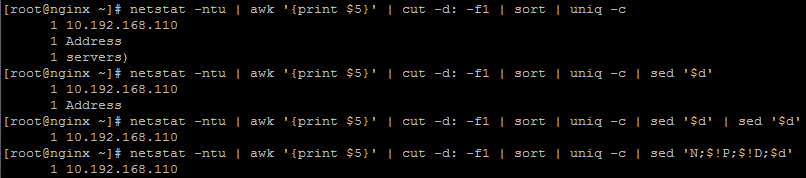
awk '!a[$0]++'



**sed专题**

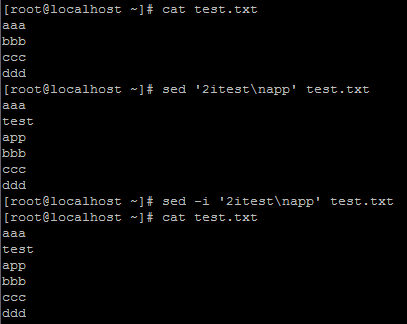
**1.sed删除最后一行、两行**

删除最后一行sed ‘$d’，删除最后两行sed -i 'N;$!P;$!D;$d'

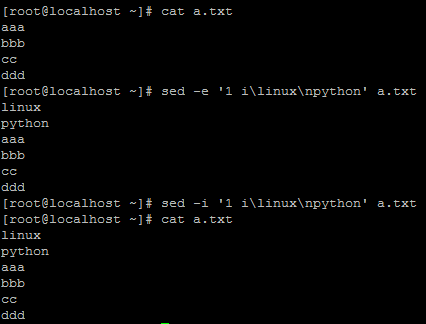


**2.sed在指定行插入多行**

sed -i '行号i内容1\n内容2' 文件



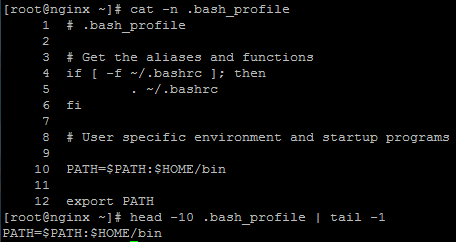
在首行插入多行内容



**3.sed查看指定行**

查看.bash\_profile文件的第10行

head -10 .bash\_profile | tail -1





**4.sed在指定行插入内容**

在第二行插入hello

sed '2 i\hello' server-start.sh

在第二行插入hello然后另起一行插入aaa

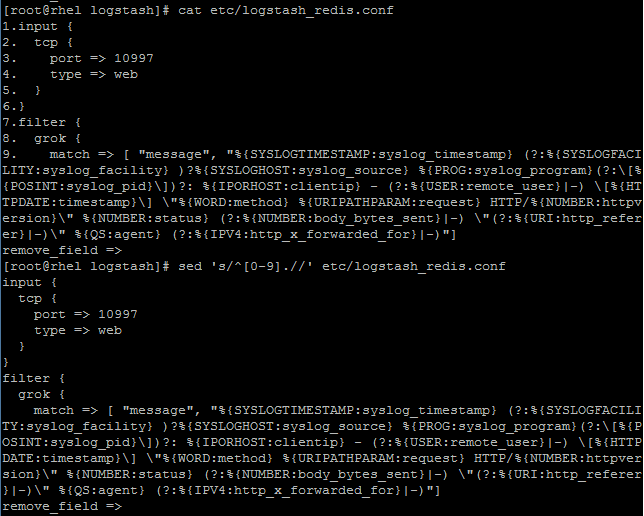
sed -i '2 i\hello\naaa' server-start.sh



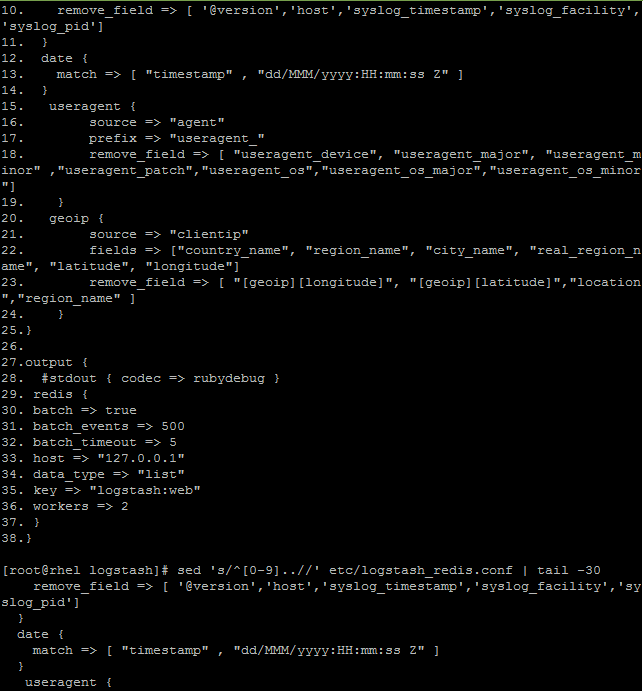
**5.sed –i修改文件内容实例**

删除每行开头的“数字.”2个或3个字符串

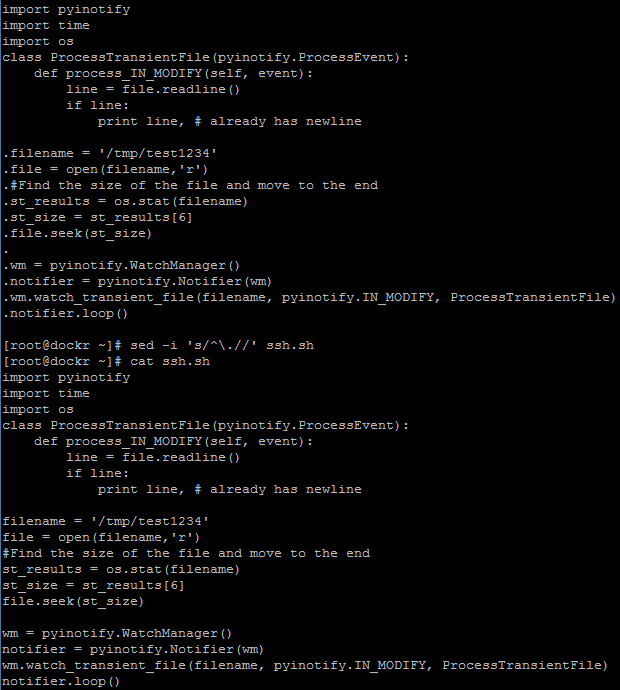
sed 's/^[0-9].//' 删除开头的1.、2.、3.、4.、、、、、9.



sed 's/^[0-9]..//' 删除开头的”10.、11.、12. 99.“



sed -i 's/^\.//' ssh.sh 删除开头的“.”



命令格式：sed -i ‘s#原内容#新内容\n另起一行#’ 文件



sed -i 's/enforcing/disabled/g' /etc/selinux/config

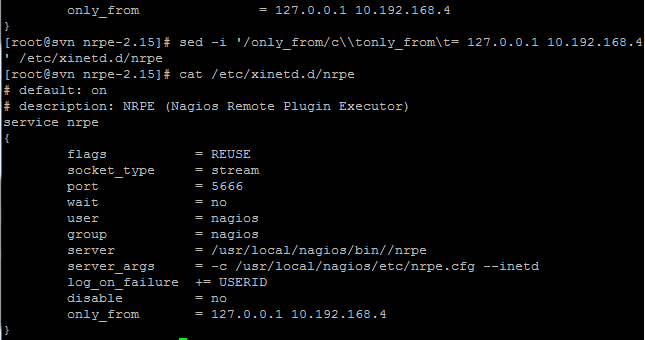


sed -i '/disable/c\\tdisable\t\t\t= no' /etc/xinetd.d/tftp

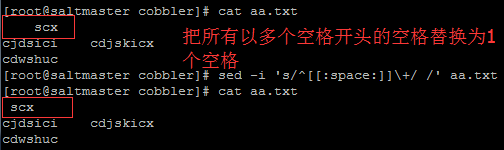


“\t“代表空格，一般的一个空格就足够了

sed -i '/only\_from/c\\tonly\_from\t= 127.0.0.1 10.192.168.4' /etc/xinetd.d/nrpe



sed -i 's/^[[:space:]]\+/ /' aa.txt

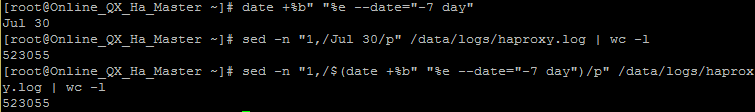


[[:space:]] 匹配一个空格 [[:space:]]+ 表示至少一个空格（>=1）

**6.sed 替换内容为变量**

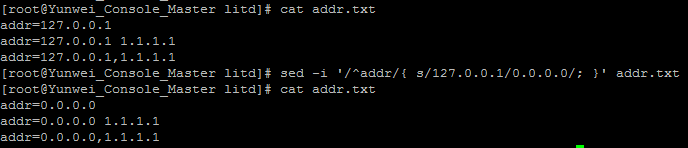
我们一般认为sed使用单引号和双引号似乎没有多大区别，但是如果要使用变量，就必须使用双引号

sed -n "1,/$(date +%b" "%e --date="-7 day")/p" /data/logs/haproxy.log | wc -l



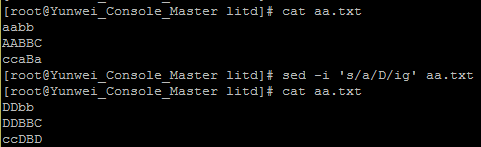
**7.sed修改指定字符串开头的行**

sed -i '/^字符串/{ s/原内容/替换的内容/; }'



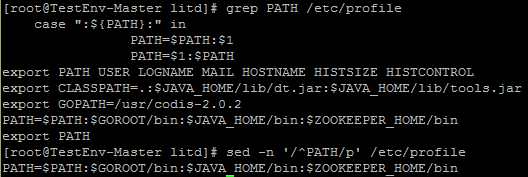
**8.sed不区分大小写修改**

sed -i 's/旧内容/新内容/ig' 文件



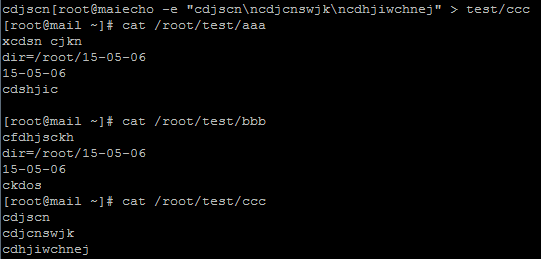
**9.sed查找以指定字符串开头的行**

sed -n '/^字符串/p' 文件



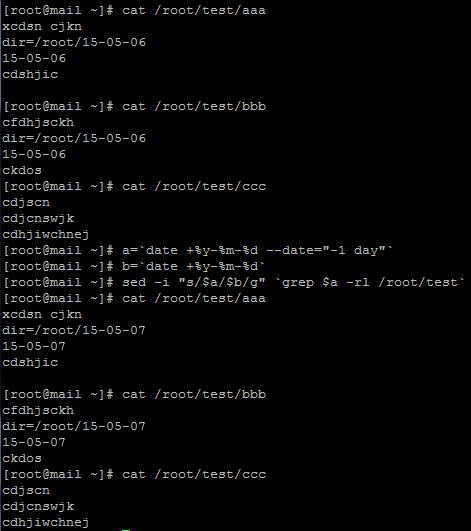
**10.sed批量替换文件内容**

我把需要批量修改字符串的所有文件放在同一目录/root/test下



核心命令sed -i "s/$a/$b/g" `grep $a -rl /root/test`

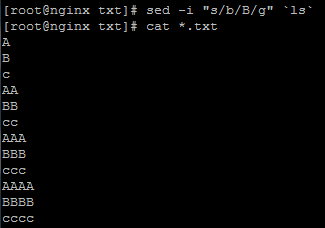
其中$a是a=`date +%y-%m-%d --date="-1 day"`，$b是b=`date +%y-%m-%d`



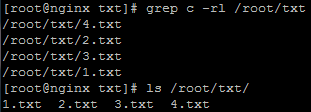
sed –i与grep命令结合， grep命令可以查找单个文件或文件夹下文件中的内容，使用参数“r”或“-R”可以递归查找指定文件夹下的所有文件；参数“l”可以让grep输出包含指定字符串的文件名，首次匹配成功后立刻停止对同文件的查询。如果不使用“l“，grep查询单个文件时输出文件中所有包含指定字符串的行，对文件夹查询时则按照“文件名:包含字符串行”的方式输出。

sed –i结合ls，ls获取的文件名都作为sed的操作对象

sed -i "s/b/B/g" `ls`



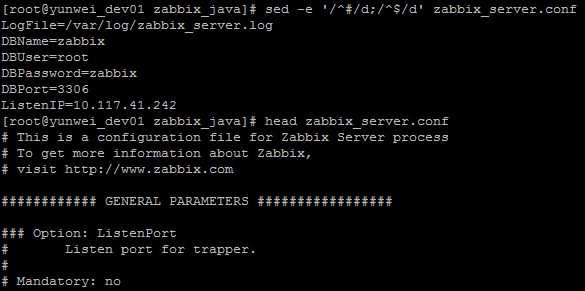
其实sed替换文件内容，操作对象可以为单个或者多个，所有只要得到所有文件的文件名即可



**11.sed筛选注释行和空白行**

习惯用grep ^[^#$]筛选注释行和空白行，但是有时候没有效果哎，试试sed！

sed -e '/^#/d;/^$/d'



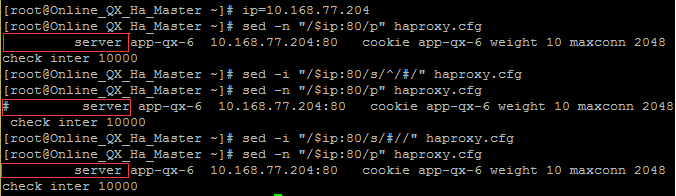
**12.sed匹配字符串后修改行开头**

用sed注释包含指定字符串的行

sed -i "/$ip:80/s/^/#/" haproxy.cfg

完了再改回来

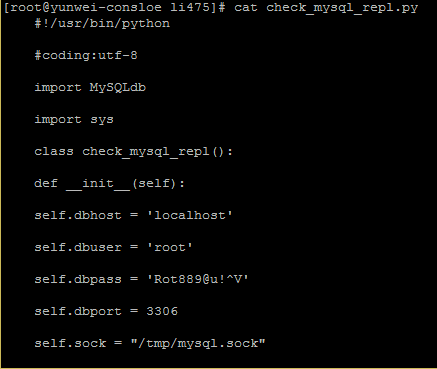
sed -i "/$ip:80/s/#//" haproxy.cfg



13.sed删除首行空格

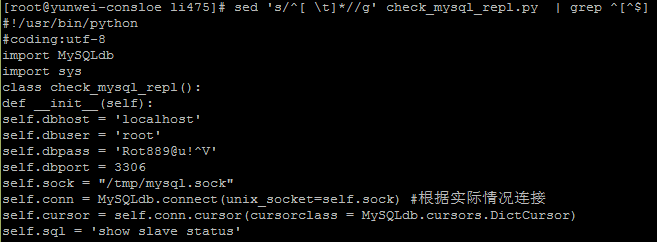
sed 's/^[ \t]\*//g'

这是原文件



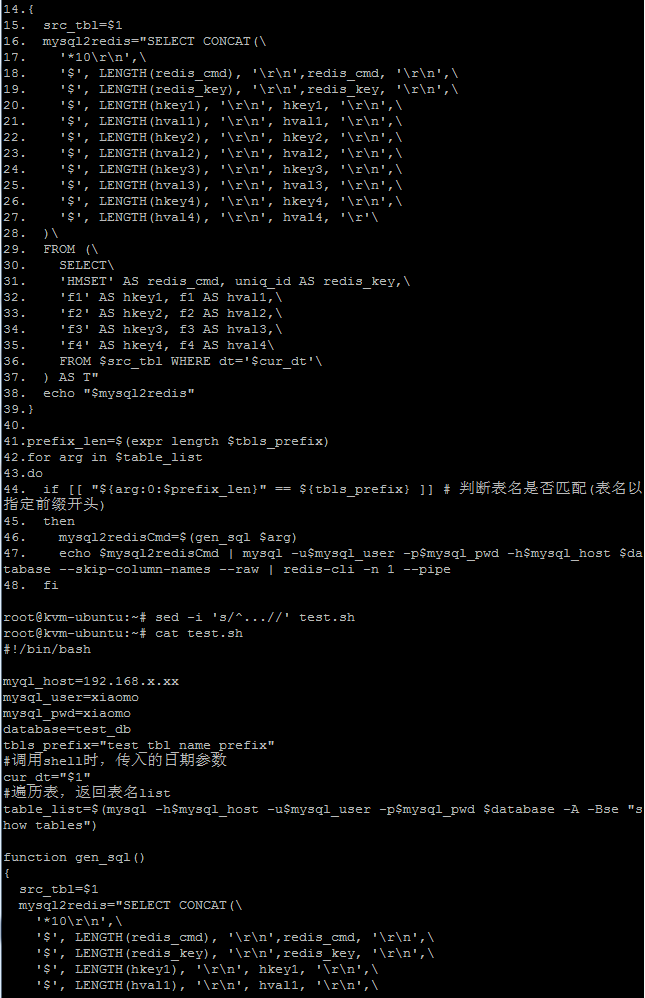
Sed删除首行空格

sed 's/^[ \t]\*//g' check\_mysql\_repl.py | grep ^[^$]



**14.**删除每一行的前3个字符串

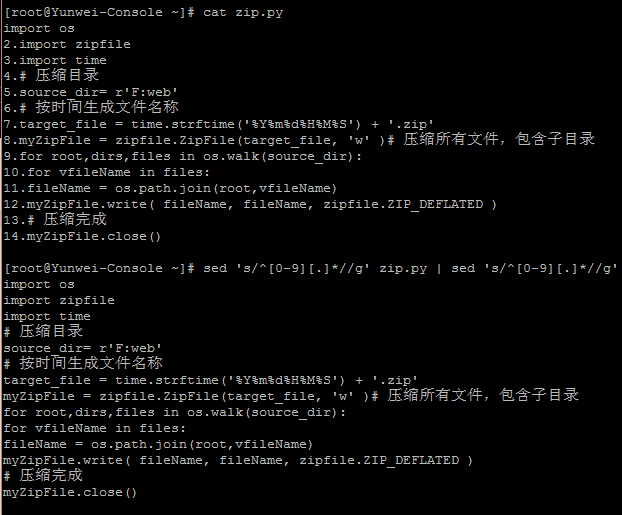
sed -i 's/^...//' test.sh，其中3个.表示3个字符串，删除n个字符串就写n个.



15.删除首行的数字和标点符号

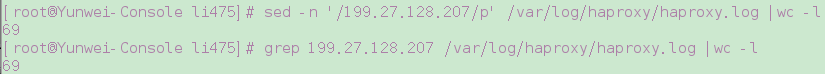
sed 's/^[0-9][.]\*//g' zip.py | sed 's/^[0-9][.]\*//g'

核心命令是^[0-9][.]\*



16.sed匹配字符串后打印行

sed -n '/字符串/p'

****