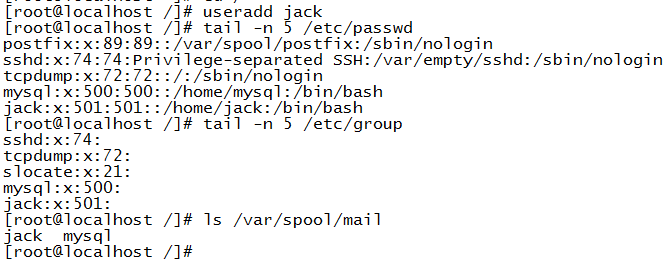
Linux学习随笔

# 一、用户和用户组

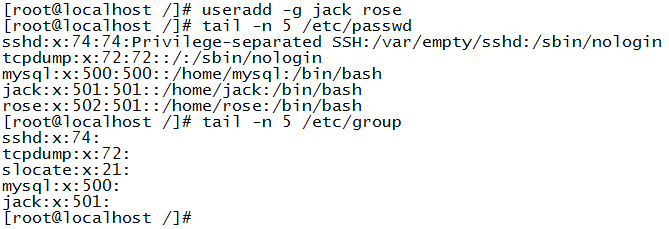
## 1. 添加用户:useradd [option] [username]

useradd jack

添加一个jack用户，信息保存在/etc/passwd文件中，同时会建立一个电子邮箱文件在/var/spool/mail路径下，以jack命名；同时系统会默认添加一个jack的用户组，信息保存在/etc/group文件中如下：

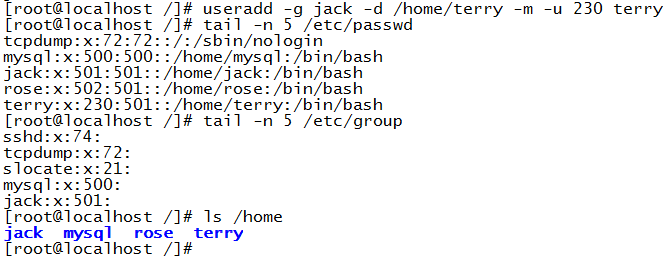


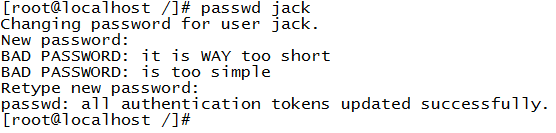
useradd –g jack rose ：添加一个rose用户，把该用户归属于用户组jack



useradd -g grouptest -d /home/test -m -u 230 test

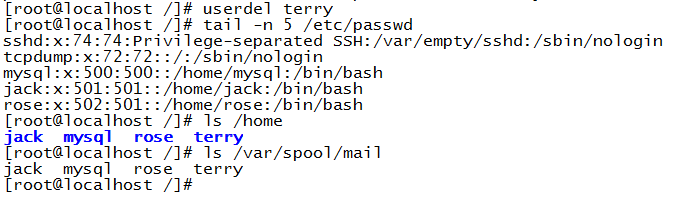
添加一个test用户，把该用户归属于用户组groupname，-d指定test的默认路径为/home/test,-m不存在路径则创建，-u指定用户号。

passwd jack ：设置jack用户的密码。

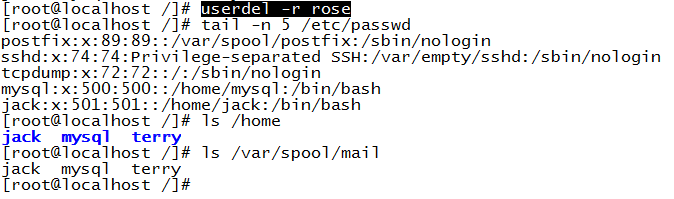


## 2. 删除用户:userdel [option] [username]

userdel terry ：删除terry用户。此时，该用户默认的登录路径没有被删除，系统email邮箱也未被删除。

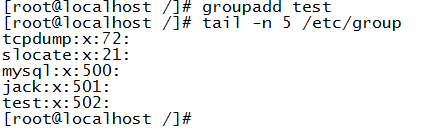


userdel –r rose ：删除rose用户。并删除rose用户在/home路径下的目录，/var/spool/mail下的email邮箱文件。当账户在使用时，会删除失败。



## 3. 添加用户组:groupadd [option] [groupname]

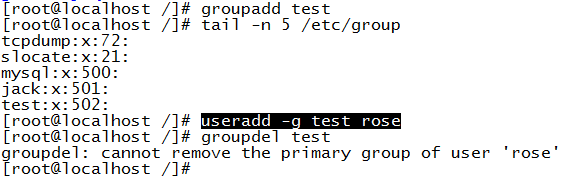
groupadd test ：添加一个test用户组，可以通过cat /etc/group查看相关信息。



## 4. 删除用户组

groupdel test ：删除用户组test

在删除用户组时，一定要确认是否还有用户在该用户组下，否则会报错。



## 5. 显示服务器上登录用户以及其线程

ps -ef|grep pts/2 ：显示匹配”pts/2”的所有线程



## 6. 杀死目标用户的线程，阻止用户登录

kill –9 [线程ID]

ps -ef | grep "root" | tail -n 5 ：使用ps查看root的进程，显示最后5行

su rose ：切换rose账户

su root ：切换回root账户，但是需要输入密码，注意：其他任何账户切换到root账户都需要输入密码

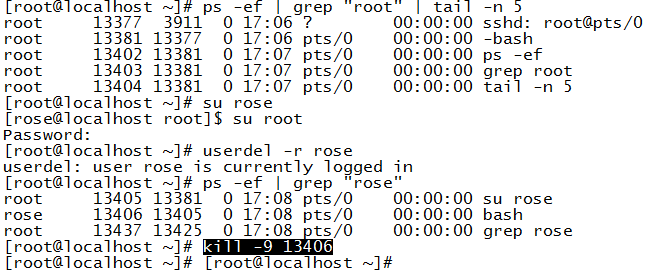
userdel -r rose ：删除rose账户，提示rose账户正在使用。

ps -ef | grep "rose" ：使用ps查看使用rose账户的进程

kill -9 13406 ：强制杀死进程。

执行kill后，再敲回车会提示rose账户loginout，连接重启。

此时再删除rose账户即可成功。



# 二、目录和文件管理

## 1. 创建目录

mkdir [路径]

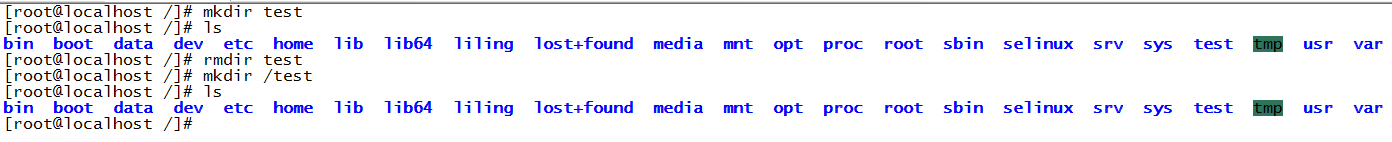
mkdir /usr/local/test

在/usr/local目录下建立test目录，使用绝对路径建立目录，同以下

使用相对路径建立目录，先到目录/usr/local下，直接在目录下建立test目录

cd /usr/local ：切换到路径/usr/local

mkdir test ：创建路径test



## 2. 删除目录

rmdir test ：删除test目录，rmdir只能删除空目录

rm –fr /usr/local/test

rm为删除目录命令，参数f，删除时不提醒，r循环删除子目录和文件。

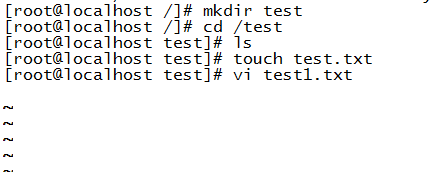


## 3. 创建文件

在usr/local/目录下创建test.txt文件，可以使用touch，vi命令。touch命令的适用范围更广。

touch usr/local/test.txt ：创建一个文件，不打开。如果该文件存在，则会改变文件的修改时间。

vi usr/local/test.txt ：创建一个文件，并打开，进入vi编辑模式。

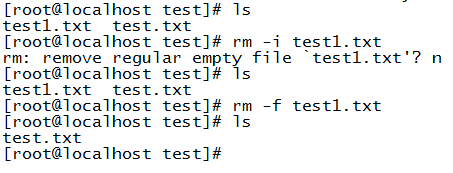




## 4. 删除文件

rm –f /usr/local/test.txt ：删除目录/usr/local/下的test.txt文件，参数f是强制删除，不提示。

rm –i /usr/local/test.txt ：删除目录/usr/local/下的test.txt文件，参数i是强制删除，提示输入，y是确认删除，n是不删除。



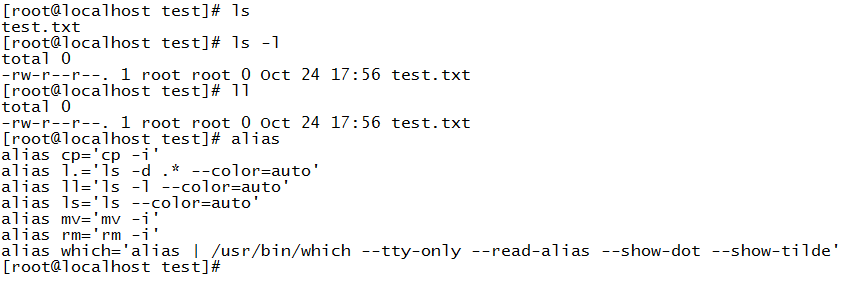
## 5. 查看目录下的文件

ls显示当前目录下的所有文件

ls –l 显示当前目录下所有文件的详情

注意：

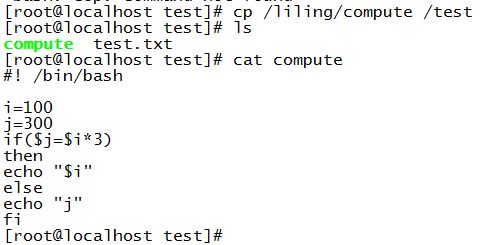
由于alias下，批注了别名ll等价与ls –l ，ll即是显示当前目录下所有文件的详情



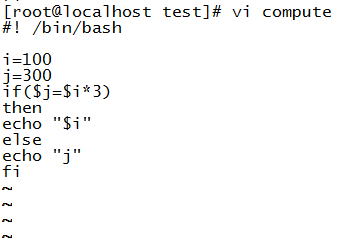
## 6. 查看文件

查看/usr/local/test.txt文件

cat compute ：只能查看文本文件，如果查看二进制文件则会出现乱码



vi /usr/local/test.txt ：查看compute文件，并进去vi编辑模式



more –s compute

查看compute文件的所有内容，只能向后翻页

more +2 compute

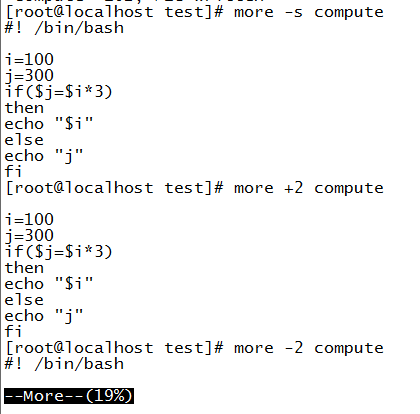
从第2行起，查看compute文件的内容

more -2 compute

从第一行开始，只显示2行内容，如果内容没有显示完，会提示

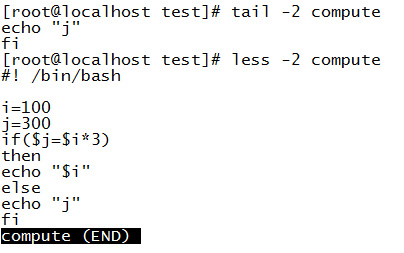
按i键，再敲回车键会翻下一行。

按i键，再敲空格键会翻下一整屏。



tail -2 /usr/local/test.txt ：从后往前读，显示2行文件内容，默认为显示10行

less -2 /usr/local/test.txt ：从后往前读，显示10行文件内容，默认为显示10行，可以向前向后翻页



## 7．搜索文件find

.代表当前目录；-name,- option为选项，可以有多个选项,选项后面跟参数，即filename,paras之类的参数；exec,print,ok是操作，commond是命令，如例子中的 –exec ls –l \{\} \，ls是命令，-l是命令的参数 \{\}用来替代查询出来的结果

find . –name ‘filename’ –option ‘paras’ …… -[exec|print|ok] commond \{\} \;

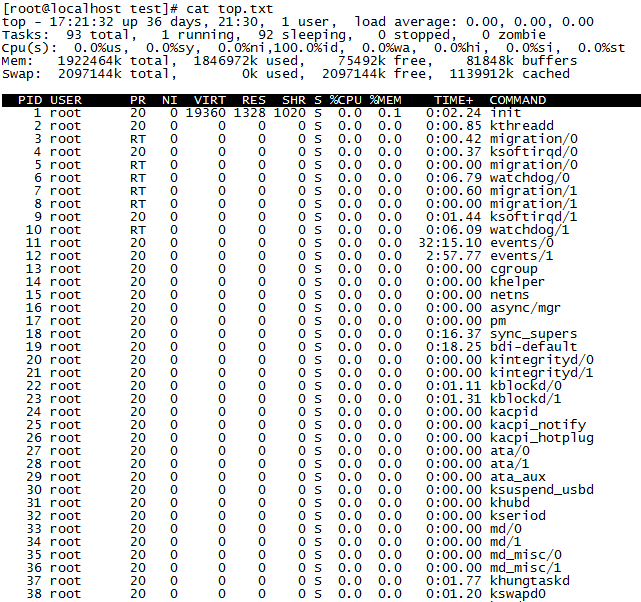
例如，查询etc 下，所有文件名带rc的普通文件，并显示出来

find /etc –name ‘rc\*’ –type f –exec ls –l \{\} \;

/etc是目录，-name是选项，rc\*是-name的参数值，-type是选项，f是-type的参数值，指普通文件，-exec是操作，ls是命令，-l是命令的参数，\{\}\;是指满足条件的结果。

find+grep+awk+sed的高级应用

/test下有一个文件top.txt，内容如下：



获取当前正在使用的内存，即Mem:行total-free-buffers，当当前内存值大于1000000时打印内存使用率，如下：

find -name "top.txt" -type f -exec cat \{\} \; | grep "Mem" | awk 'begin {FS=","} {print $3,$5,$7,$9}' | sed "s/k//g"|awk '{one=1;two=2;three=3;four=4} {if( $(one)-$(three)-$(four)>1000000) print ($(one)-$(three)-$(four))/$(one)}'

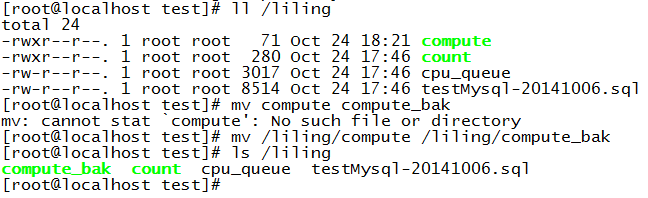
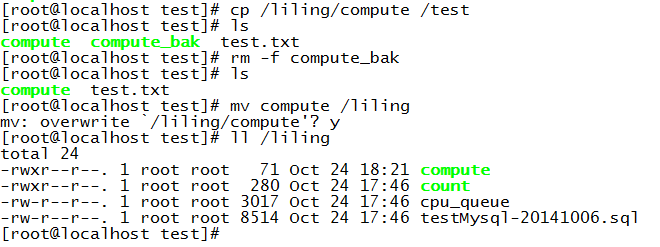
使用find 找到top.txt文件，并执行cat输出并重定向给grep，grep筛选Mem:行获取对应的分域，重定向到sed，sed使用替换k，再重定向到awk进行计算判断，最后打印结果。

## 8．文件的复制，移动，重命名。

cp /liling/compute /test ：复制/liling/compute到/test目录下，如果未指定文件名，则为原文件名，否则为指定的文件名。如果/test目录下存在文件，则会提示是否覆盖原文件，y为覆盖，N为不覆盖。如果在cp命令后加-i提示是否覆盖，-f则强制覆盖不提示。

mv compute /liling ：剪切文件到/liling目录下，如果/liling未指定文件名，则为原文件名，否则为指定的文件名。如果/liling下有同名文件，则会提示是否覆盖原文件，y为覆盖，N为不覆盖。如果在mv命令后加-i提示是否覆盖，-f则强制覆盖不提示。

mv compute compute\_bak : 为compute文件改名为compute\_bak

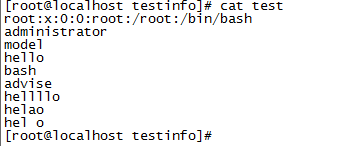


# 三、正则表达式

|  |  |
| --- | --- |
| 基本的正则表达式元字符集合及其意义 | |
| \* | 匹配0和0个以上字符 |
| . | 匹配单个字符 |
| ^ | 匹配行首，或后面字符的非 |
| $ | 匹配行末 |
| [] | 匹配字符集 |
| \ | 转定义为基本字符 |
| \<\> | 精确匹配 |
| \{n\} | 匹配字符或字符串出现n次 |
| \{n,\} | 匹配字符或字符串至少出现n次 |
| \{n,m\} | 匹配字符或字符串出现n~m次 |
| ? | 匹配0或1次在其之前出现的字符 |
| + | 匹配1次以上在其之前出现的字符 |
| () | 表示一个字符集合或用于expr中 |
| | | 表示“或”，或匹配一组 |

基本正则表达式元字符测试：

文本文件test，内容如下

****

1. grep -n "hel\*" test

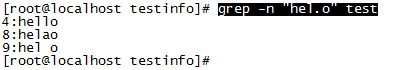
显示结果：

****

grep 文本搜索命令；-n选项，显示匹配行号；”hel\*”参数，该行字符中必须有”he”字符，并且”he”字符后可以没有”l”字符出现，或者”he”字符后必须是”l”字符同时”l”字符可以出现1次以上。

1. grep -n "hel.o" test

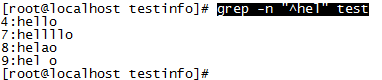
显示结果：

****

匹配字符”hel”+任意一个字符+”o”字符。如图，”hel”与”o”中间是空字符也是匹配的。

1. grep -n "^hel" test

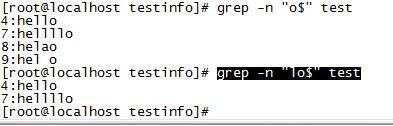
显示结果：

****

匹配以”hel”字符开头的行

1. grep -n "lo$" test与grep -n "o$" test的区别

显示结果：

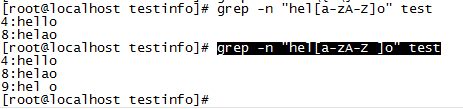
****

前一个以匹配”o”结尾的行，后一个以匹配”lo”结尾的行。

注意：“^$“连在一起使用是匹配空行。

1. “\”字符是转定义的字符，例如：”\.”等于”.”字符，而不是正则表达式中的.代表任意字符。
2. grep -n "hel[a-zA-Z ]o" test与grep -n "hel[a-zA-Z]o" test的区别

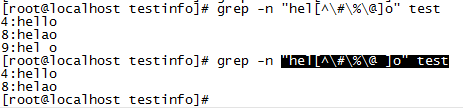
显示结果：

****

前一个匹配hel[小写a到z,大写A-Z在内的任意一个字符]o

后一个匹配hel[小写a到z,大写A-Z，空格在内的任意一个字符]o

^符号在[]中是取反，如下：

****

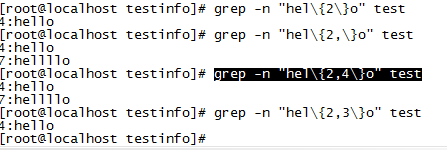
grep –n “<\hello\>” test，精确匹配

****

1. grep –n “hel\{2\}o” test，grep –n “hel\{2,\}o” test，

grep –n “hel\{2,4\}o” test，grep –n “hel\{2,3\}o” test

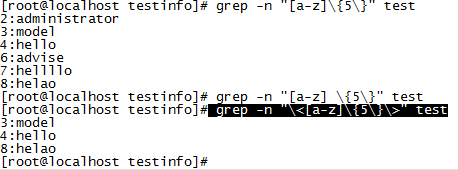
结果如下：

****

第一个匹配”hello”字符，第二个匹配hello,helllo,helll….o中的任意一个,第三个匹配hello,helllo,hellllo中的任意一个，第四个匹配hello,helllo中的任意一个。

1. grep -n "\<[a-z]\{5\}\>" test

结果如下：

****

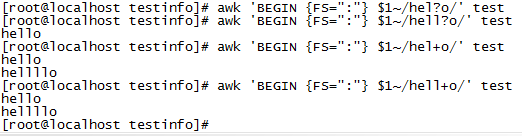
精确匹配只有5个小写英文字母的行。

1. awk 'BEGIN {FS=":"} $1~/hel?o/' test

awk 'BEGIN {FS=":"} $1~/hell?o/' test

awk 'BEGIN {FS=":"} $1~/hel+o/' test

awk 'BEGIN {FS=":"} $1~/hell+o/' test

****

以上可以看出”?”和”+”符号的作用，”?”匹配前面的字符0次或1次。”+”匹配前面的符号最少1次。

第一段脚本匹配:heo,helo，

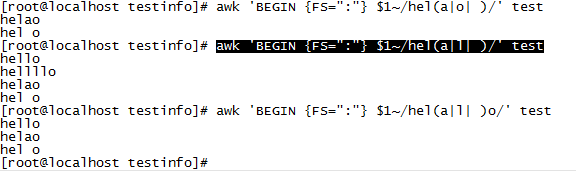
第二段脚本匹配:helo,hello

第三段脚本匹配:helo,hello,hello,……,hell……o

第四段脚本匹配: hello,hello,……,hell……o

1. awk 'BEGIN {FS=":"} $1~/hel(a|l| )/' test

结果如下：

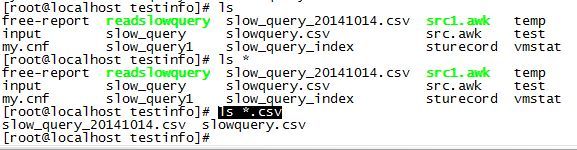
****”()”与”|”经常在一起使用，它们表示符合一组字符集中的任意一个，条件便成立。

(a|l| )等价于[al ]

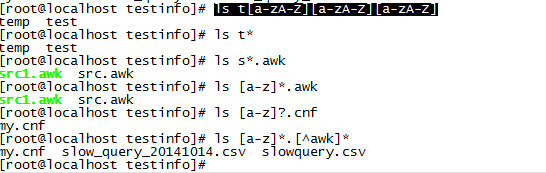
1. 通配符

ls \*.csv

结果如下：

****ls 等价于ls \*，显示当前目录下所有的文件和目录

ls \*.csv，显示当前目录（不包含子目录）下，所有.csv文件

****ls t[a-zA-Z][a-zA-Z][a-zA-Z] :显示当前目录下，文件名只由t开头，后三位由大小写英文字母组成的4个字符的文件和目录。

ls t\* : 显示当前目录下，文件名由t开头的所有文件和目录。

ls s\*.awk : 显示当前目录下，文件名由t开头，后缀为.awk的所有文件。

ls [a-z]\*.awk : 显示当前目录下，文件首字符是小写英文字母，后缀为.awk的所有文件。

ls [a-z]?.cnf : 显示当前目录下，文件首字符是小写英文字母，后跟任意一个满足文件或目录命名的字符，并且后缀为.awk的所有文件。

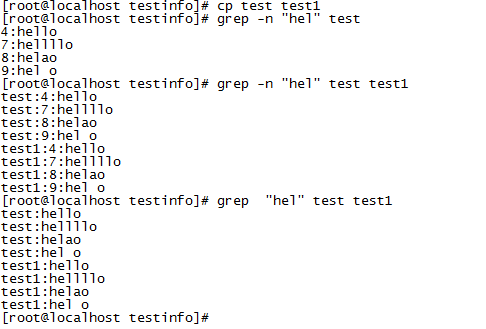
ls [a-z]\*.[^awk]\* : 显示当前目录下，文件首字符是小写英文字母，后缀不是.awk的所有文件。

# 四、grep 命令

## 1、grep的格式

grep [选项] [模式] [文件名(多个文件用空格隔开)]

示例如下：

****

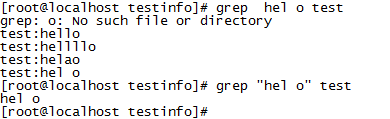
cp test test1 : 复制test文件。

grep –n “hel” test : 搜索test文件中匹配hel字符的行，打印出来并打印行号。

grep –n “hel” test test1 : 搜索test,test1文件中匹配hel字符的行，打印出来并打印行号。

grep“hel” test test1 : 搜索test,test1文件中匹配hel字符的行，打印出来。

grep hel o test与grep "hel o" test的区别

****

当需要匹配的模式中带有空格时，需要使用””把模式包含起来。

常用选项

-n 显示匹配的行，并显示行号

-c 统计匹配模式的总行数

-v 显示不匹配的行

-i 忽略大小写

-h 查询多文件时，不显示文件名，列出行

-l 查询多文件时，只显示文件名，不列出行

-s 表示不显示不存在或无匹配时的错误信息

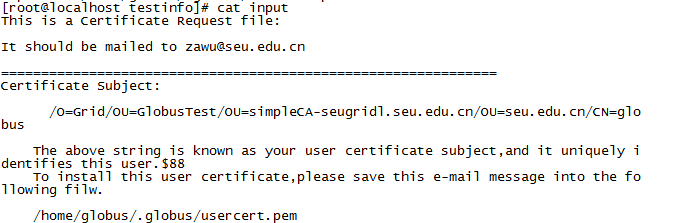
-r 对目录进行递归搜索

-x 匹配整词，如果一行有多个词，即使某一个词匹配也不满足匹配条件

-w 匹配整行，如果一行有多个词，某一个词匹配即满足匹配条件

## 2、grep与正则表达式

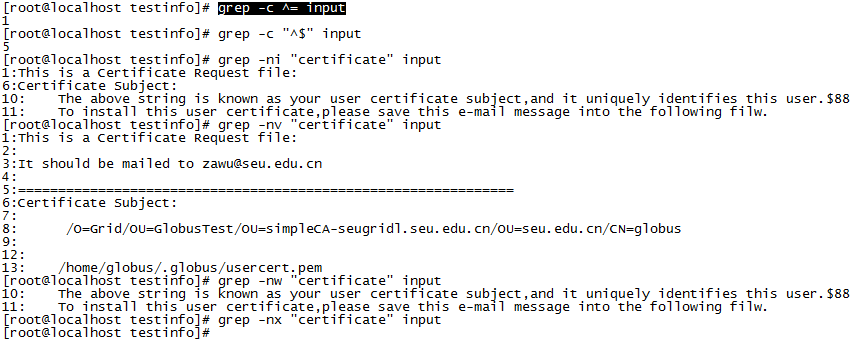
示例文件：

****

grep ^= input

****

匹配行首为”=”的行。

**** grep -c ^= input : 匹配行首为”=”的行，打印匹配的总行数

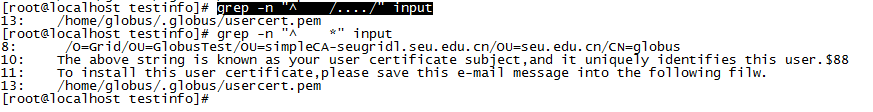
grep -c "^$" input : 匹配空行，打印匹配的总行数

grep -ni "certificate" input : 忽略大小写，打印匹配的行数和对应的行号。

grep -nv "certificate" input : 打印不匹配的行及其行号。

grep -nw "certificate" input : 匹配行中包含模式的行，打印该行及其行号。

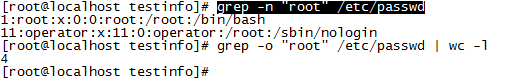
grep -nx "certificate" input : 整行单词匹配，打印匹配行及其行号

****grep -n "^ /..../" input : 匹配行首由4个空格组成，并且第5位为/，第6-9位为任意字符，第10位为/的行并打印行号。

grep -n "^ \*" input : 匹配行首由4个空格组成的行并打印行号。

grep -n "root" /etc/passwd ：获取文件中匹配root的行并打印行号

grep -o "root" /etc/passwd | wc –l ：获取文件中root出现的次数

****

# 五、sed命令

## 1、sed命令格式

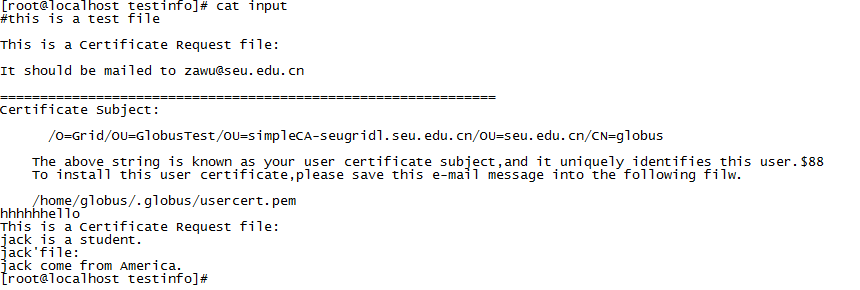
sed [命令选项] [命令] [文件]

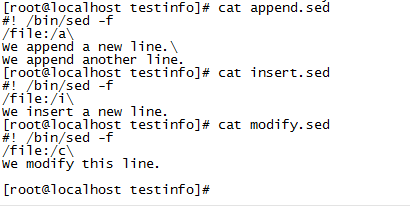
sed可以对管道文件进行处理，如下：

cat input | sed ‘命令’ ，sed文件仅仅对文件流进行编辑，源文件是不会被改动的。

|  |  |
| --- | --- |
| sed命令选项及其意义 | |
| -n | 不打印所有行，只输出满足条件的行 |
| -f | 表示正在调用的sed脚本 |
| -e | 表示下一个字符串解析为sed的编辑命令 |
| sed定位文本的方法 | |
| x | X为指定行号 |
| x,y | 指定从x到y的行号范围 |
| /pattern/ | 匹配包含pattern的行 |
| /pattern/pattern/ | 匹配包含两个pattern的行 |
| /pattern/,x | 从匹配第一个包含pattern的行到x行 |
| x,/pattern/ | 从x行到匹配第一个包含pattern的行 |
| x,y! | 查询不包含x到y的行 |
| sed编辑命令 | |
| = | 打印行号 |
| p | 打印匹配行 |
| a\ | 在定位行号后追加文本 |
| i\ | 在定位行号前插入文本 |
| c\ | 在定位行号替换该行的文本 |
| s | 替换匹配的字符串或字符 |
| d | 删除定位行 |
| r | 从另一个文件中读取文本 |
| w | 将文本写入一个文件 |
| y | 变换字符 |
| q | 第一个模式匹配后退出 |
| l | 显示与八进制ASCII码等价的控制字符 |
| {} | 在定位行执行的命令组 |
| n | 读取下一个输入行，用下一个命令处理新的行 |
| h | 将模式缓冲区的文本复制到保持缓冲区 |
| H | 将模式缓冲区的文本追加到保持缓冲区 |
| x | 互换模式缓冲区和保持缓冲区的内容 |
| g | 将保持缓冲区的文本复制到模式缓冲区 |
| G | 将保持缓冲区的文本追加到模式缓冲区 |

示例文本：

****示例脚本：append.sed ,insert.sed , modify.sed

****

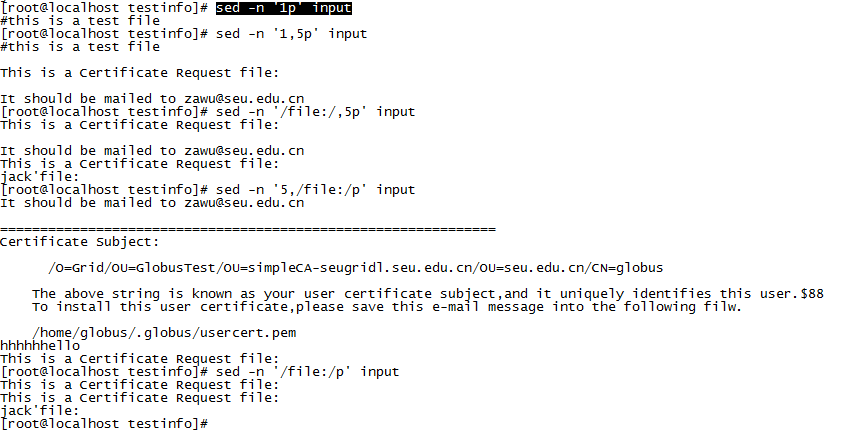
sed -n '1p' input ：打印第一行

sed -n '1,5p' input ：打印第1到第5行

sed -n '/file:/,5p' input ：打印从第一个匹配file:字符的行到第5行，并且打印第5行以后的匹配file:字符的行。

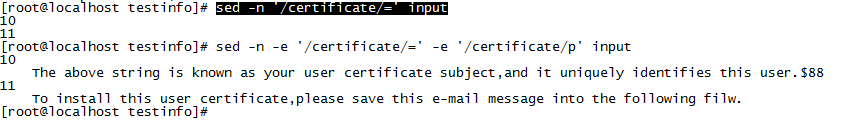
sed -n '5,/file:/p' input ：打印第5行到第一个匹配file:字符的行

sed -n '/file:/p' input : 打印匹配file:字符的所有行

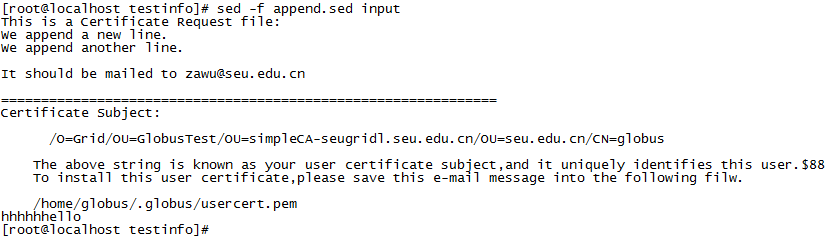
****

sed -n '/certificate/=' input ：打印匹配certificate字符的行号

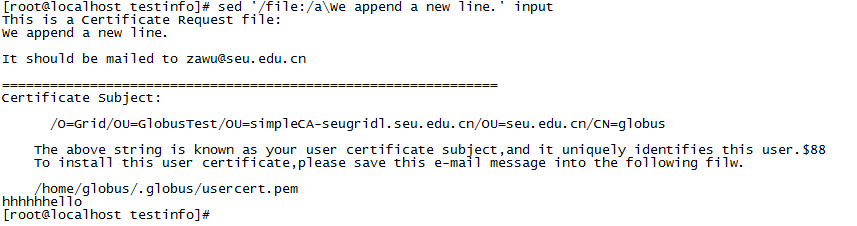
sed -n -e '/certificate/=' -e '/certificate/p' input ：打印匹配certificate字符的行号和行

****

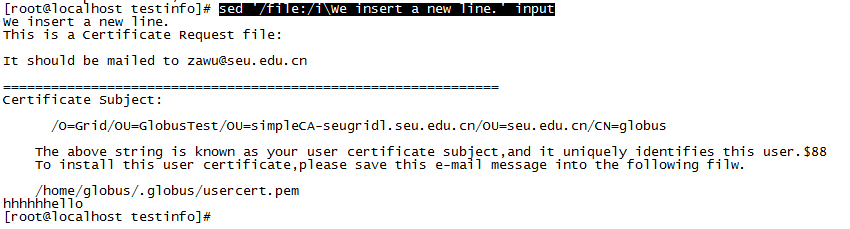
sed –f append.sed input ：执行append.sed脚本

****

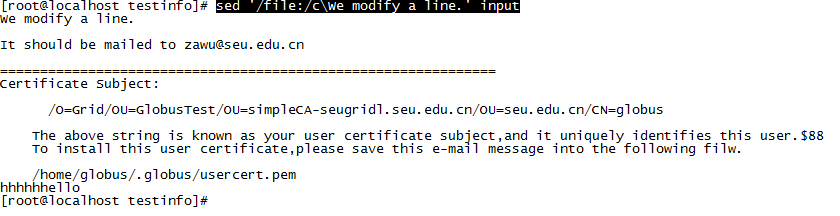
sed '/file:/a\We append a new line.' input ：在匹配file:字符的行后加入一行We append a new line

****

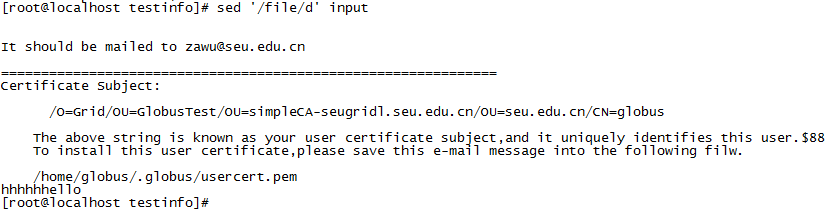
sed '/file:/i\We insert a new line.' input ：在匹配file:字符的行之前加入一行We insert a new line

****

sed '/file:/c\We modify a line.' input ：将匹配file:字符的行替换为We modify a line

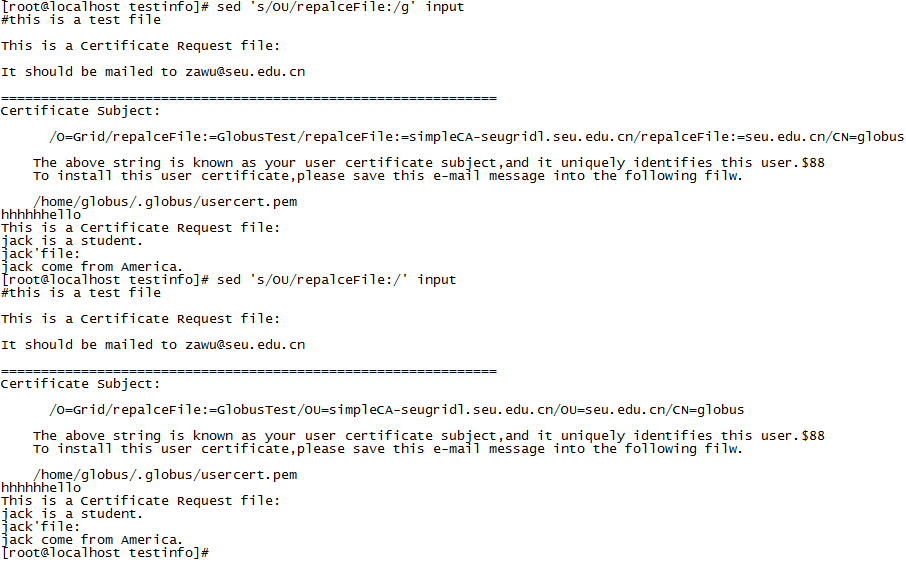
****

sed '/file:/d' input ：删除匹配file:字符的行

****

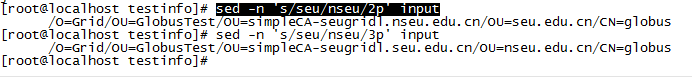
sed 's/OU/repalceFile:/g' input : 匹配OU的行，替换所有的OU为repalceFile:，g代表替换所有。

sed 's/OU/repalceFile:/' input : 匹配OU行，替换匹配行的第一个OU为repalceFile:

****

sed -n 's/seu/nseu/2p' input : 匹配含有seu的行，替换匹配行中前2次出现的seu为nseu，并打印替换后的匹配行

sed -n 's/seu/nseu/3p' input : 匹配含有seu的行，替换匹配行中前3次出现的seu为nseu，并打印替换后的匹配行。

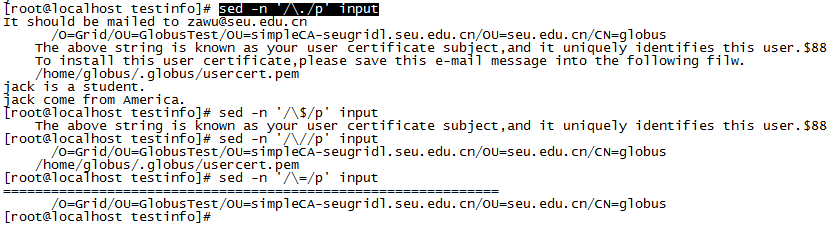
****

## 2、元字符匹配

sed -n '/\./p' input

sed -n '/\$/p' input

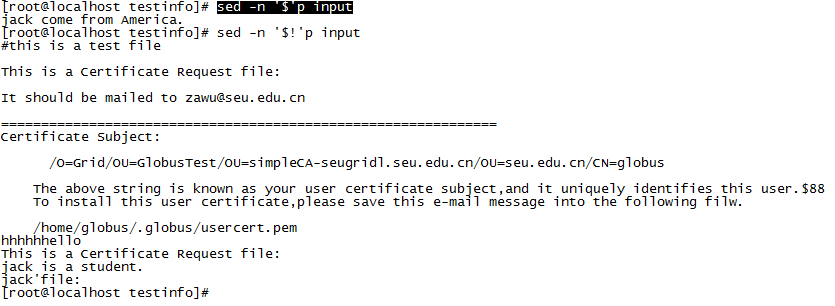
sed -n '/\=/p' input

****

## 3、正则表达式匹配

sed -n '$'p input : 匹配最后一行

sed -n '$!'p input : 匹配除开最后一行的所有行

****

sed -n 's/seu/nseu/w ouput' input ：查询匹配seu的行，替换匹配行中的第一个seu为nseu，并输出到ouput文件

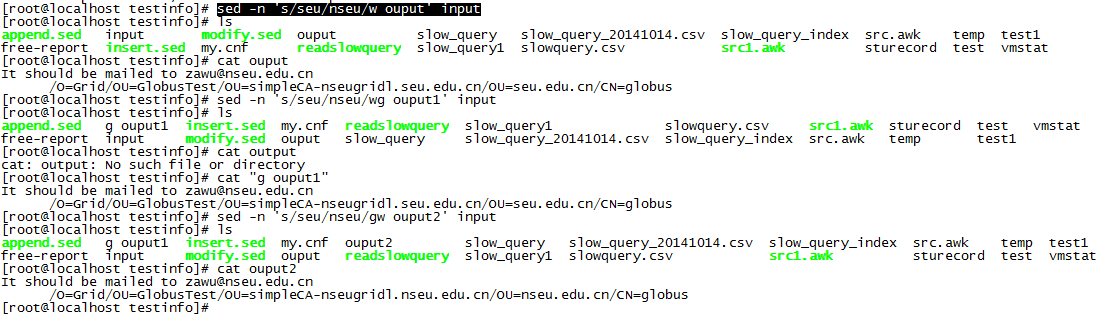
sed -n 's/seu/nseu/wg ouput1' input ：查询匹配seu的行，替换匹配行中的第一个seu为nseu，并输出到”g ouput”文件

sed -n 's/seu/nseu/gw ouput2' input ：查询匹配seu的行，替换匹配行中所有的seu为nseu，并输出到”g ouput”文件

注意：

命令参数w，g的顺序，如果g在w之前则先执行全部替换，再输出，否则是执行输出然后w之后所有的字符为文件名。

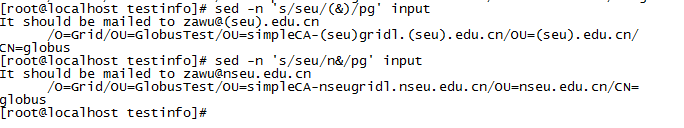
所以，替换的操作顺序是先替换后输出，参数位置错误将得不到正确结果。

****

sed -n 's/seu/(&)/pg' input : 匹配seu行，并替换所有seu为(seu)，打印输出

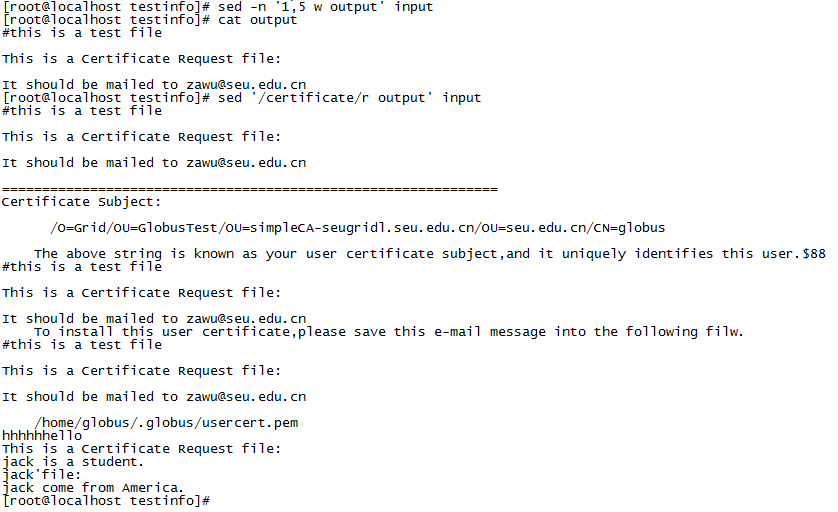
sed -n 's/seu/n&/pg' input : 匹配seu行，并替换所有seu为nseu，打印输出

&符号表示匹配字符串

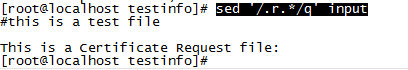
****

sed -n '1,5 w output' input : 查询1到5行写入output文件

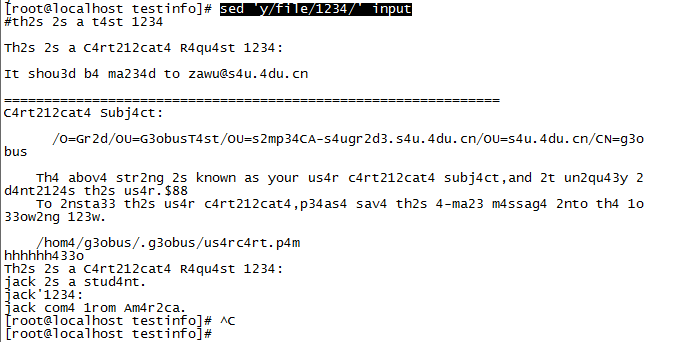
sed -n '/certificate/r output' input : 查询input文件中匹配certificate字符的行，并在每一个匹配行后追加output文件的内容

****

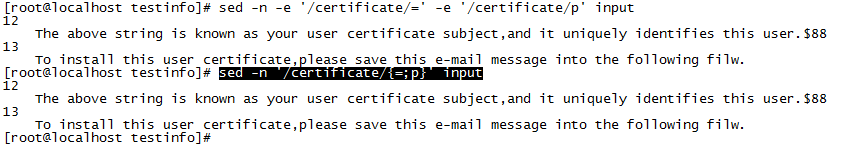
sed '/.r.\*/q' input : 查询第一匹配模式的行显示并退出。

****

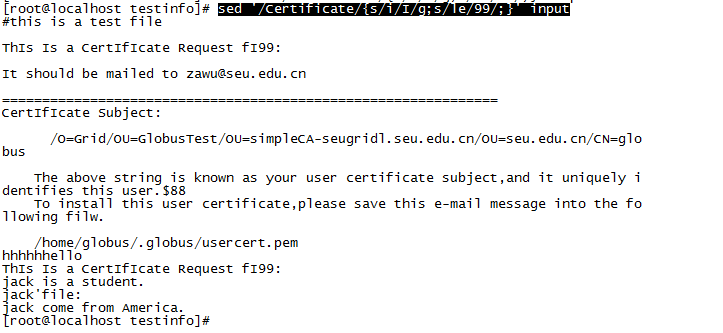
sed 'y/file/1234/' input : 在input文件中，把f字符替换为1，i字符替换为2，l字符替换为3，e字符替换为4

****

sed -n '/certificate/{=;p}' input : {}作为命令组。查询匹配certificate的行并打印行号和行。等同于：sed -n -e '/certificate/=' -e '/certificate/p' input

****

sed '/Certificate/{s/i/I/g;s/le/99/;}' input : 查询匹配Certificate的行，将i替换为I，le替换为99，并显示替换后的文本

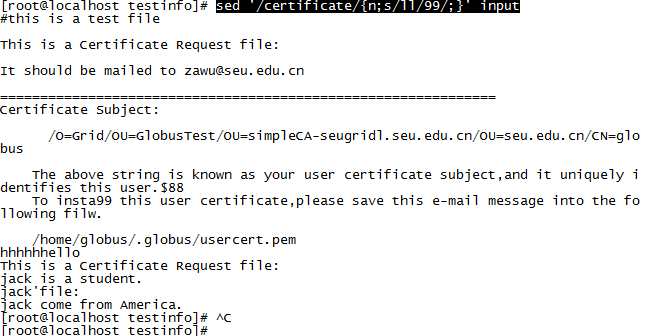
****

## 4、sed高级编辑命令

sed '/certificate/{n;s/ll/99/;}' input : 查询匹配certificate的行，将该行的下一行执行替换ll为99的操作。

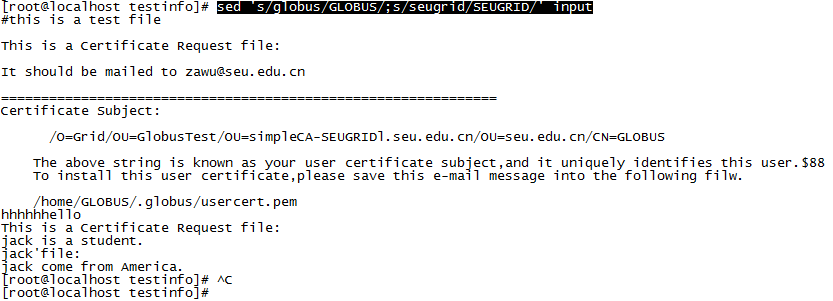
注意：

该示例中，sed的执行顺序：先匹配，然后找到下一行进行替换。

****

sed 's/globus/GLOBUS/;s/seugrid/SEUGRID/' input : 将globus替换为GLOBUS，seugrid替换为SEUGRID，并打印替换后的结果。

;可以分隔多个编辑命令

****

# 六、awk命令

## 1、awk命令格式

awk [域分隔符] [‘awk程序段’] [输入文件]

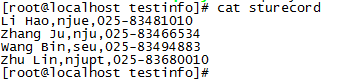
awk的执行格式

awk [–f] [awk脚本文件] [输入文件]

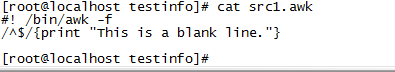
调用awk脚本格式

[./awk脚本文件] [输入文件]

示例文本：

****

示例脚本：

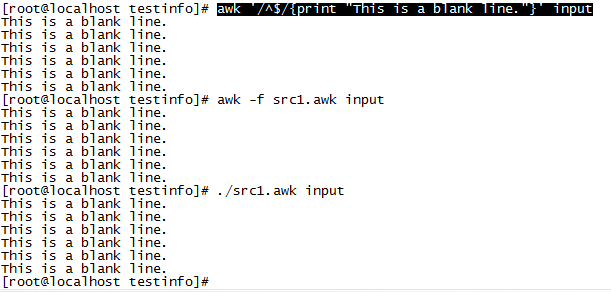
****

## 2、awk的匹配示例

awk '/^$/{print "This is a blank line."}' input : 查询文件中的空行，当该行为空行时，打印””包含的语句。^$是匹配空行，{}是操作。

awk -f src1.awk input : 调用src1.awk文件

./src1.awk input : 执行src1.awk文件

****

## 3、awk的逻辑运算

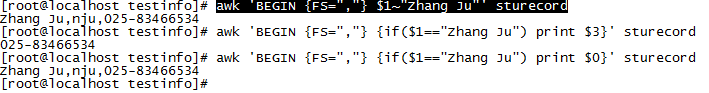
|  |  |
| --- | --- |
| awk关系运算符及其意义 | |
| 符号 | 意义 |
| > | 大于 |
| < | 小于 |
| >= | 大于等于 |
| <= | 小于等于 |
| == | 等于 |
| != | 不等于 |
| !~ | 不匹配正则表达式 |
| ~ | 匹配正则表达式 |
| awk布尔运算符及其意义 | |
| 符号 | 意义 |
| || | 逻辑或 |
| && | 逻辑与 |
| ! | 逻辑非 |

awk 'BEGIN {FS=","} $1~"Zhang Ju"' sturecord : 查询域分隔符为”,”，第一个域匹配”Zhang Ju”的行

awk 'BEGIN {FS=","} {if($1=="Zhang Ju") print $3}' sturecord : 查询域分隔符为”,”，第一个域匹配”Zhang Ju”的行的第三个域

awk 'BEGIN {FS=","} {if($1=="Zhang Ju") print $0}' sturecord : 查询域分隔符为”,”，第一个域匹配”Zhang Ju”的行

awk 'BEGIN {FS=","} $1~"Zhang Ju"' sturecord 与awk 'BEGIN {FS=","} {if($1=="Zhang Ju") print $0}' sturecord是等价的。

****

为示例文本每行最前面添加数字，并执行以下代码

awk 'BEGIN {FS=","} {if($1>1) print $0}' sturecord : 查询域分隔符为”,”，第一个域大于1的行

awk 'BEGIN {FS=","} {if($1<2) print $0}' sturecord : 查询域分隔符为”,”，第一个域小于2的行

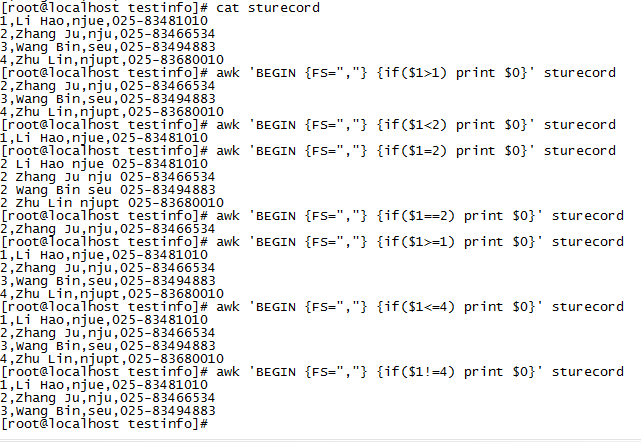
awk 'BEGIN {FS=","} {if($1=2) print $0}' sturecord : 查询域分隔符为”,”，将第一个域赋值为2，并输出结果。

awk 'BEGIN {FS=","} {if($1==2) print $0}' sturecord : 查询域分隔符为”,”，第一个域等于2的行

awk 'BEGIN {FS=","} {if($1>=1) print $0}' sturecord : 查询域分隔符为”,”，第一个域大于等于1的行

awk 'BEGIN {FS=","} {if($1<=4) print $0}' sturecord : 查询域分隔符为”,”，第一个域小于等于4的行

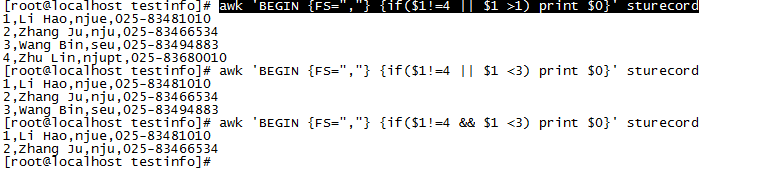
awk 'BEGIN {FS=","} {if($1!=4) print $0}' sturecord : 查询域分隔符为”,”，第一个域不等于4的行

****

awk 'BEGIN {FS=","} {if($1!=4 || $1 >1) print $0}' sturecord : 查询域分隔符为”,”，第1个域不等于4或者第1个域大于1的所有行

awk 'BEGIN {FS=","} {if($1!=4 || $1 <3) print $0}' sturecord : 查询域分隔符为”,”，第1个域不等于4或者第1个域小于3的所有行

awk 'BEGIN {FS=","} {if($1!=4 && $1 <3) print $0}' sturecord : 查询域分隔符为”,”，第1个域不等于4并且第一个域小于3的所有行

****

### awk逻辑运算进阶

为sturecord每一行加上学生分数

awk 'BEGIN {FS=","} {total=$5+$6+$7+$8+$9 avg=total/5} {print $2,avg}' sturecord : 打印每个学生的平均成绩，语法错误。

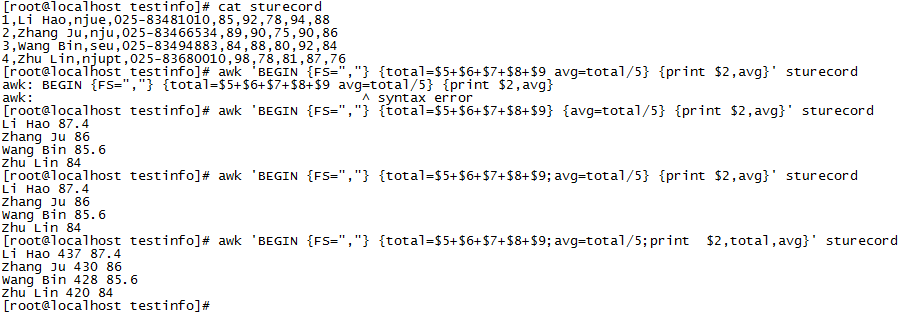
awk 'BEGIN {FS=","} {total=$5+$6+$7+$8+$9} {avg=total/5} {print $2,avg}' sturecord : 打印每个学生的平均成绩。

awk 'BEGIN {FS=","} {total=$5+$6+$7+$8+$9;avg=total/5} {print $2,avg}' sturecord : 打印每个学生的平均成绩。

awk 'BEGIN {FS=","} {total=$5+$6+$7+$8+$9;avg=total/5;print $2,total,avg}' sturecord : 打印每个学生的总分和平均成绩。

注意：

1. 前三行代码都是为了求学生的平均成绩，但第一行报语法错误，第二、三行执行成功得到正确结果。
2. 分析：在awk中定义变量，每第一个变量可以用{}包含起来，也可以在同一个{}中用;隔开多个变量。
3. 操作命令可以单独用{}包含，也可以在同一个{}中用;分隔。

****

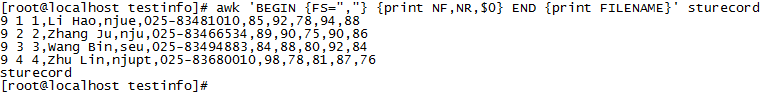
## 4、系统变量

BEGIN后的{FS=”,”}内容是指在执行操作之前执行重定义域分隔符为”,”，END后的{print FILENAME}指在执行操作之后执行打印文件名。{print NF,NR,$0}在执行时执行打印域数，行号和所有内容。

NF : 每行的域数

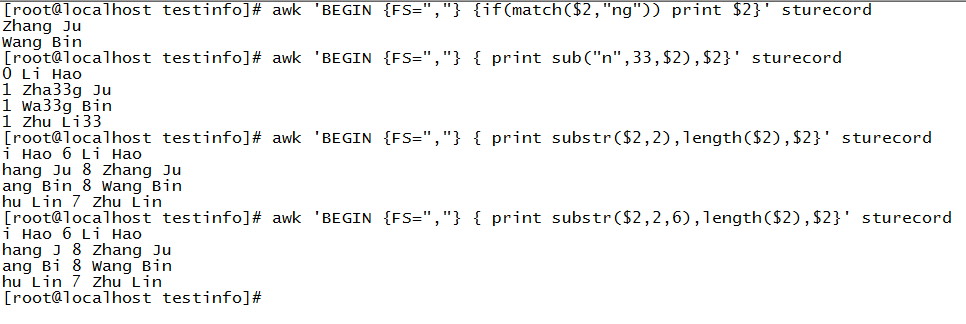
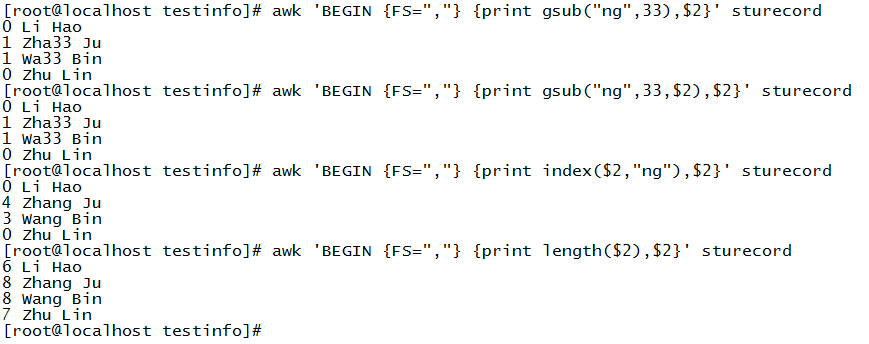
NR: 每行的行号

FILENAME：文件名

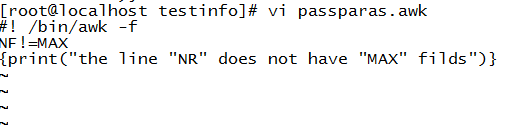
****

## 5、内置字符串函数

|  |  |
| --- | --- |
| 函数名 | 意义 |
| gsub(r,t) | 替换r字符为t字符 |
| gsub(r,s,t) | 在t中替换r字符为s字符 |
| index(t,s) | 返回s字符在t中的第一个位置 |
| length(t) | 返回t字符的长度 |
| match(t,s) | 判断在t中是否存在s字符，0返回不存在，1返回存在 |
| sub(r,s,t) | 在t中，替换第一个r为s |
| substr(t,index) | 截取t字符串中，从索引index开始到尾部的字符串 |
| substr(t,index,length) | 截取t字符串中，从索引index开始，长度为length的字符串 |

****6、awk脚本传参

示例脚本：

****

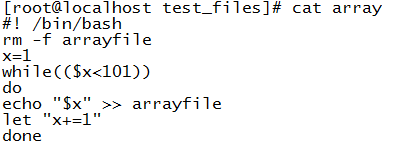
参数MAX与FS之间有空格隔开，MAX和FS的=号左右两边都不能有空格，否则语法错误

脚本含义：当输入参数不等于sturecord总行数，则逐行打印文本内容，并打印行号和输入参数。

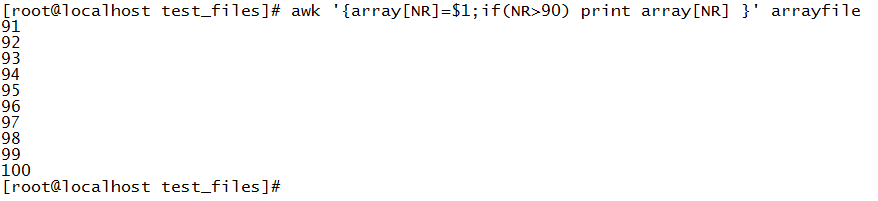
执行结果：

****7、条件语句、循环语句、数组

执行示例脚本可以生成一个1-100的arrayfile文件。

****

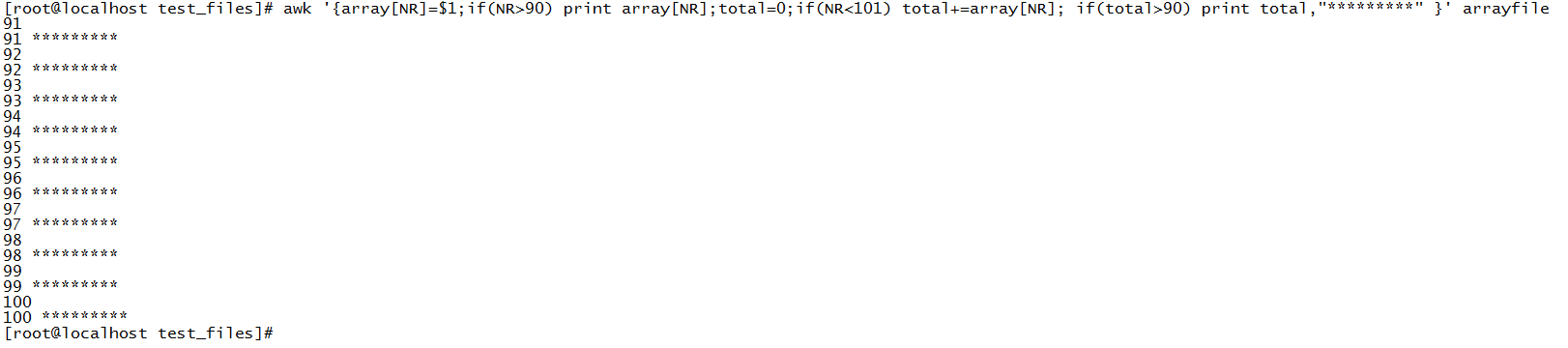
在{}中定义数组，使用if条件判断，当行号>90时，打印数组对应元素值

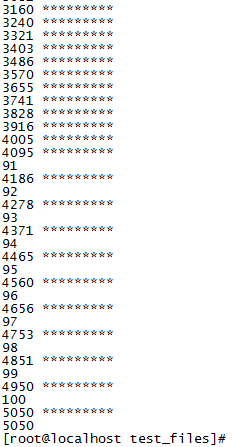
****

awk '{array[NR]=$1;if(NR>90) print array[NR];total=0;if(NR<101) total+=array[NR]; if(total>90) print total,"\*\*\*\*\*\*\*\*\*" }' arrayfile : 打印数组中索引>90的元素值，当total参数值大于90，打印total。

注意：

1、由于awk是逐行读取数据，可以把这个操作看做一个while循环，{}中的while相当于另一个循环，这样就相当于循环中嵌套循环，外循环中每次循环都把total重新赋值，这就导致结果并不是从1加到100的总数。

****awk 'BEGIN {total=0;}{array[NR]=$1;if(NR>90) print array[NR];if(NR<101) total+=array[NR]; if(total>90) print total,"\*\*\*\*\*\*\*\*\*" } END {print total}' arrayfile : 执行的结果与上面就有不同了，相当计算从1到100的总和，并且打印数组中索引>90的元素值，同时打印total大于90的值。

****

awk 'BEGIN {total=0;}{array[NR]=$1;if(NR<101) total+=array[NR] } END {print total}' arrayfile ：计算1-100的总和。相当于脚本：

total =0

i=1

while(($i<101))

do

let “array[i]=i”

let “total+=array[i]”

let “i+=1”

done

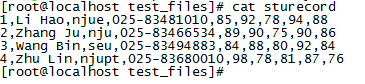
****

# 七、文件的排序、合并和分割

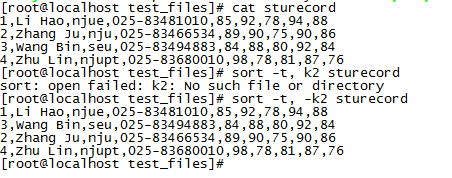
## 1、文件的排序

|  |  |
| --- | --- |
| sort命令选项及其意义 | |
| 选项 | 意义 |
| -t | 改变域分隔符 |
| -k | 选择按哪一个域排序 |
| -n | 升序排序 |
| -r | 降序排序 |
| -u | 去掉重复项 |
| -o | 将排序的结果输入到一个文件 |
| -m | 合并多个文件排序 |
| -c | 测试是否已经排序 |

示例文件

****

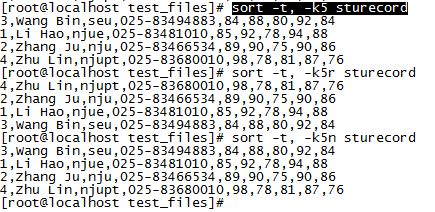
sort -t, -k2 sturecord ：sort为排序命令，-t是指定分隔符，-k是指定域，2是按第二个域来排序，sturecord是文件

****

sort -t, -k5 sturecord ：按第5个域排序，由于第5个域是数字，因此排序会按从小到大的顺序排序，而不是按字符的顺序排序。

sort -t, -k5r sturecord ：按第5个域排序，r是指按从大到小的顺序排序。

sort -t, -k5n sturecord ：按第5个域排序，n是指按从小到大的顺序排序。

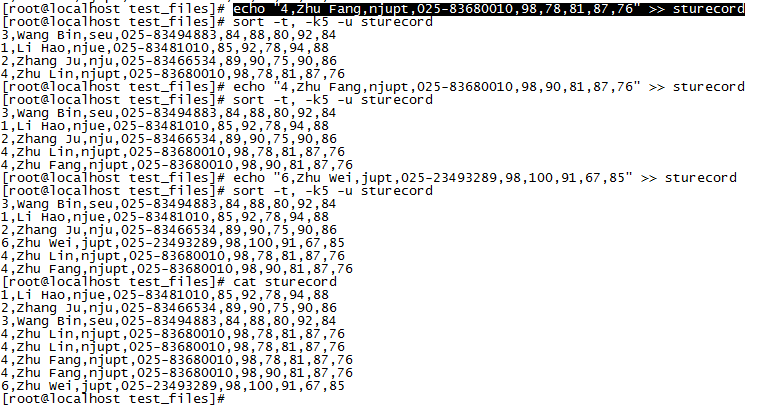
****

先复制第4条记录，写入到sturecord文件，然后执行

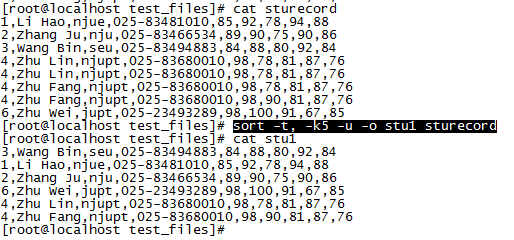
sort -t, -k5 sturecord

再执行

sort -t, -k5 -u sturecord ：按第5个域排序，并除掉重复项。为了查看去掉的重复项是哪些，继续写入与第5域相同值，但其他域不相同的行。从下方截图可以看出，这里的重复项是指每行完全相同的重复项。

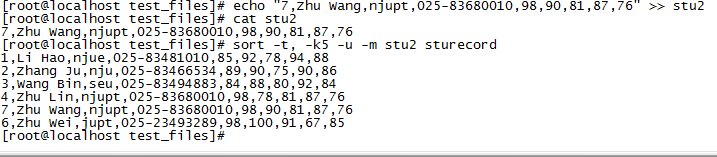
****

sort -t, -k5 -u -o stu1 sturecord ：将排序后的结果输入到stu1文件,-o输出排序后的结果到stu1文件。

****

新建一个stu2文件，写入一行记录

sort -t, -k5 -u -m stu2 sturecord ：合并stu2,sturecord文件并进行排序，-m是合并多个文件

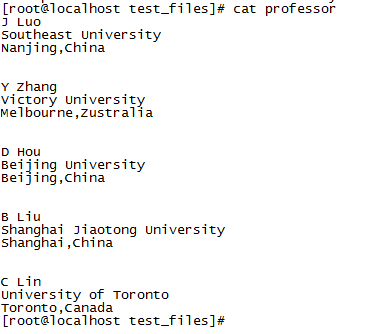
****

sort -t, -k3n -c sturecord ：输出第一个没有按第3域升序排序的行，如果文件已经排序，则不会输出任何结果



## 2、sort与awk的联合用法

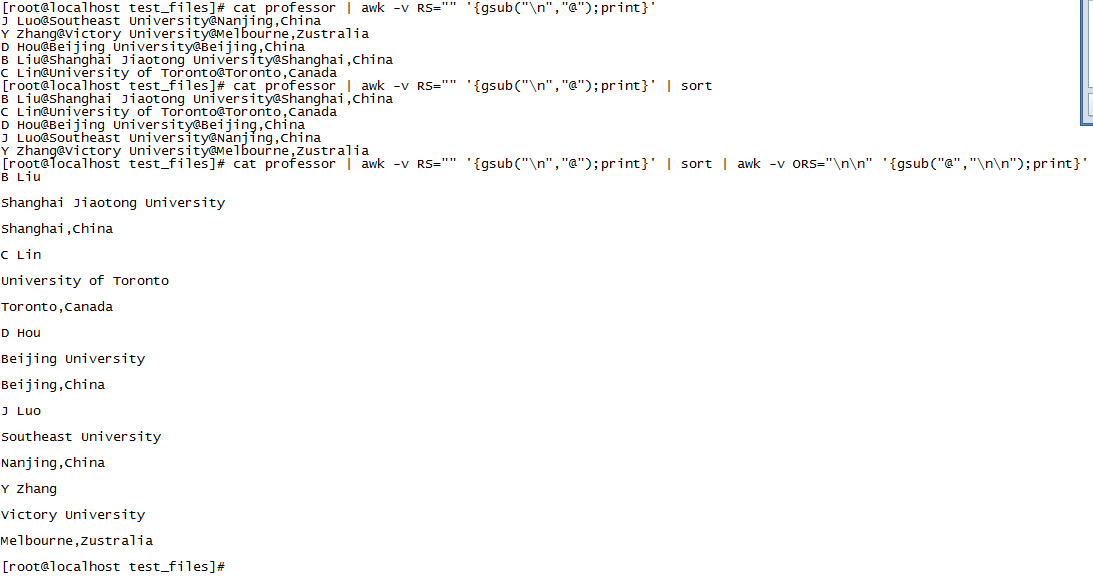
示例文件，每个文件块用2个空行分隔

****

cat professor | awk -v RS="" '{gsub("\n","@");print}' ：合并每个文件块的内容，输出为一行并打印，RS为记录分隔符，默认为空格键。

cat professor | awk -v RS="" '{gsub("\n","@");print}' | sort：合并每个文件块的内容，输出为一行，并进行排序打印

cat professor | awk -v RS="" '{gsub("\n","@");print}' | sort | awk -v ORS="\n\n" '{gsub("@","\n\n");print}' ：合并每个文件块的内容，输出为一行，并进行排序，在把@符号替换为2个空行后输出打印。ORS为输出记录分隔符，默认为换行符

****

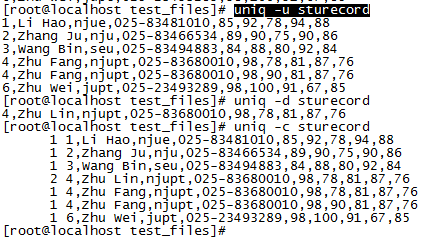
## 3、uniq命令

uniq类似与sort –u ，但是只对连续的重复项有作用。

uniq -u sturecord ：只显示没有重复的项

uniq -d sturecord ：只显示重复项

uniq -c sturecord ：显示重复次数

****

grep -v "==========" input | grep -v "^$" | sed -e 's/\./\n/g' -e 's/\//\n/g' -e 's/\=/\n/g' -e 's/\:/\n/g' -e 's/\,/\n/g' -e 's/\@/\n/g' -e 's/ /\n/g' -e 's/\#//g' -e 's/\$//g' -e 's/'"'"'/\n/g' | grep -v "^$" | sort | uniq -c | sort

统计input文件中，每个单词出现的频率，并按出现次数升序排序。

grep -v "==========" input : 去掉======号行

grep –v “^$” ：去掉空行

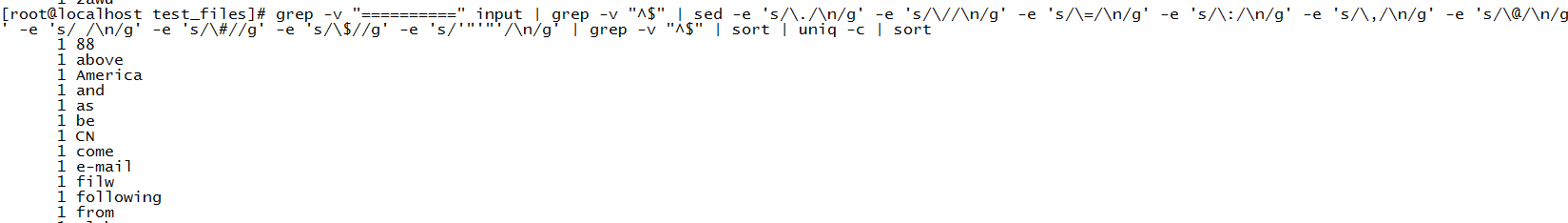
sed -e 's/\./\n/g' -e 's/\//\n/g' -e 's/\=/\n/g' -e 's/\:/\n/g' -e 's/\,/\n/g' -e 's/\@/\n/g' -e 's/ /\n/g' -e 's/\#//g' -e 's/\$//g' -e 's/'"'"'/\n/g' ：将.用换行符替换，将/用换行符替换，将=用换行符替换，将:用换行符替换，将,用换行符替换，将@用换行符替换，将空格用换行符替换，将#用空字符替换，将$用空字符替换，将’用换行符替换，实际上，这一系列的替换是将每个单词单独提取出来作为一行。

grep -v "^$" ：替换空行

sort ：排序

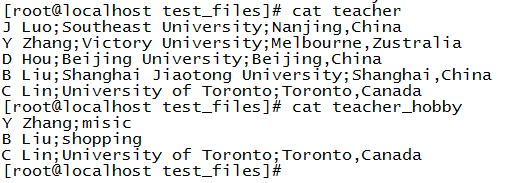
uniq –c ：记录每个单词的出现次数

sort ：按出现次数排序

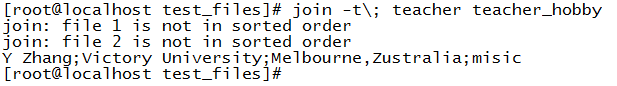
****

## 4、join命令

示例文本：

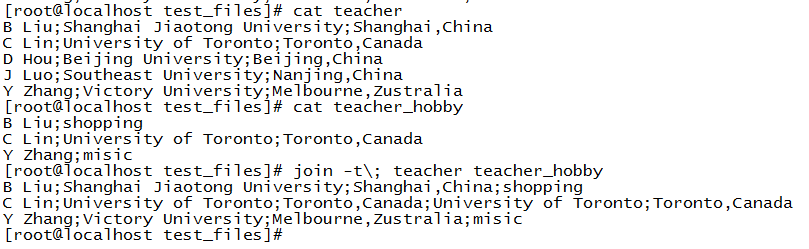
****

join只能对已排序的文件进行处理

****

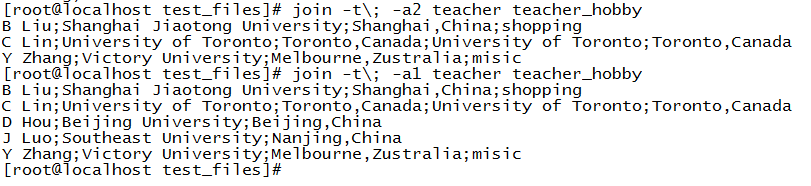
重新处理teacher，teacher\_hobby文件，进行排序。

join -t\; teacher teacher\_hobby ：查询两个文件中域相同的行，显示顺序为：相同域，第一个文件的不同域，第二个文件的不同域

****

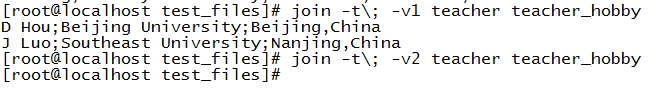
join -t\; -a2 teacher teacher\_hobby ：显示第2个文件的内容，包括第1个文件的相同项的不同部分。

join -t\; -a1 teacher teacher\_hobby ：显示第1个文件的内容，包括第2个文件的相同项的不同部分。

****

join -t\; -v1 teacher teacher\_hobby ：显示第1个文件的不同项

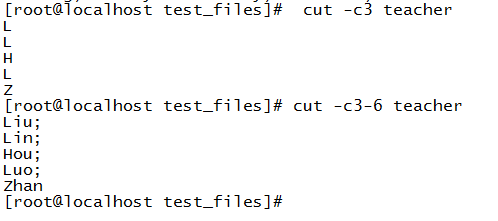
join -t\; -v2 teacher teacher\_hobby ：显示第2个文件的不同项

****

## 5、cut命令

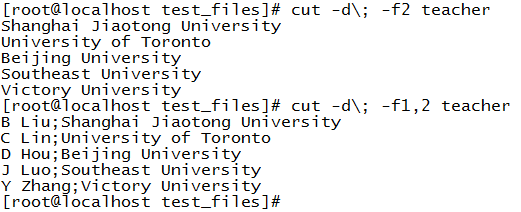
cut -c3 teacher ：截取第3个字符

cut -c3-6 teacher ：截取从第3个到第6个字符

****

cut -d\; -f2 teacher ：以;为分隔符，截取第2个域的字符

cut -d\; -f1,2 teacher ：以;为分隔符，截取从第1个域到第2个域的字符

****

# 6、paste命令

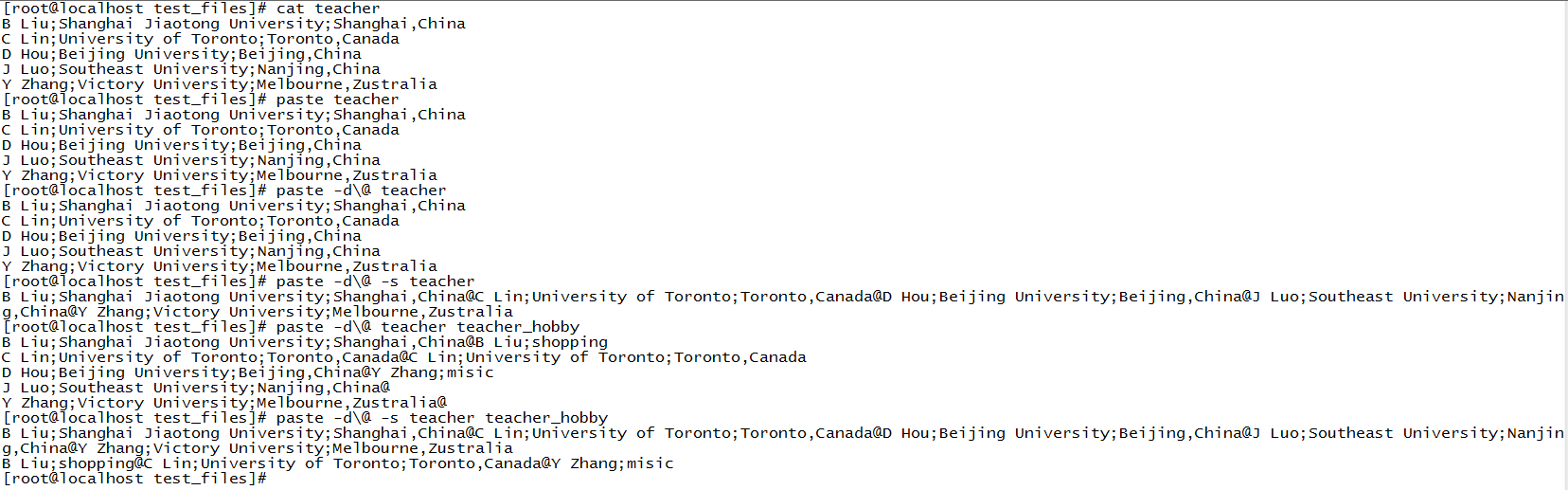
paste teacher ：作用等同于cat

paste -d\@ teacher ：使用@作为分隔符

paste -d\@ -s teacher ：使用@作为分隔符，把所有内容连成一行

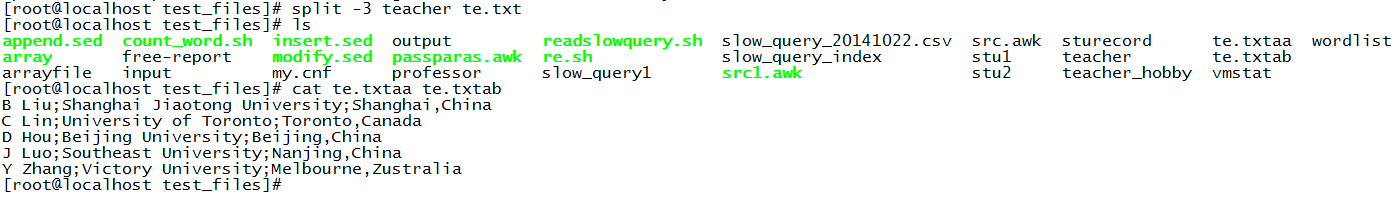
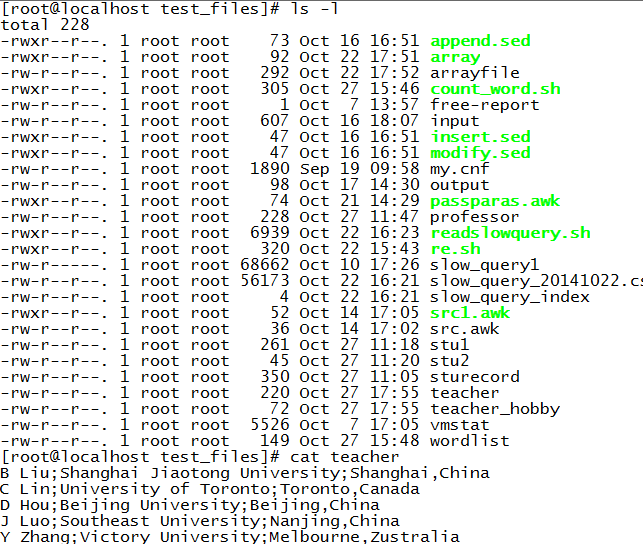
paste -d\@ teacher teacher\_hobby ：把两个文件相同行号的内容连成一行，并使用@作为分隔符分隔两个文件的所属。

paste -d\@ -s teacher teacher\_hobby ：先把第1个文件的内容连成一行，然后把第2个文件的内容连成一行，最后两行连成一行，使用@分隔。

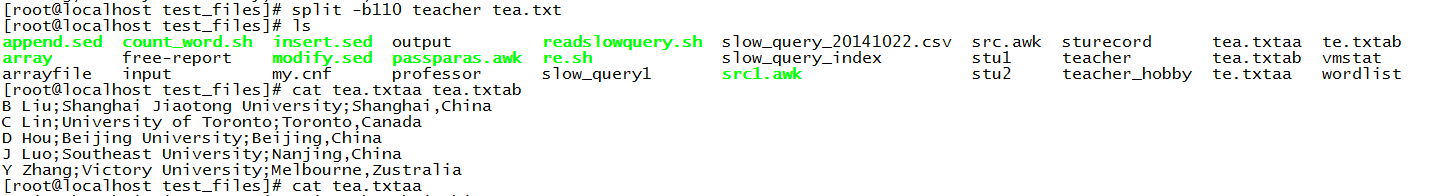
****

## 7、split命令

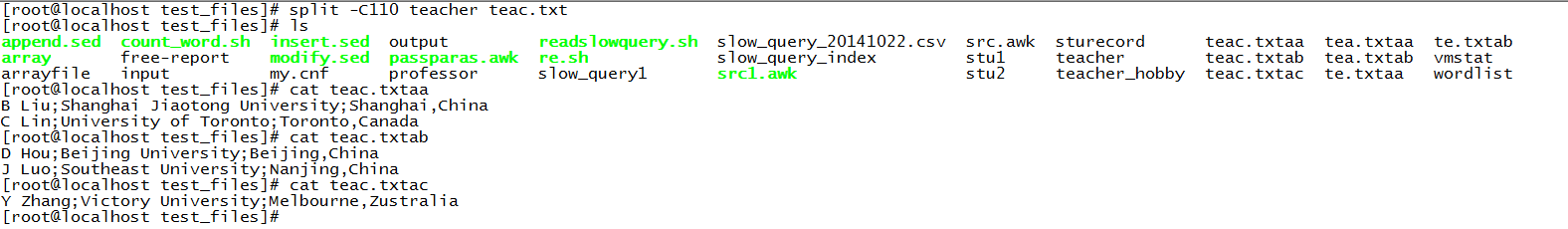
split -3 teacher te.txt ：按每个文件3行数据，切割teacher文件。

****

split -b110 teacher tea.txt ：按每个文件110个字符切割teacher文件。

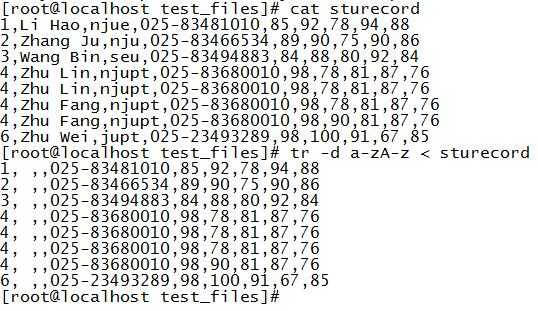
****

split -C110 teacher teac.txt ：按每个文件110个字符切割teacher文件，并保持文件内容的完整性，它并没截断第3行来严格满足110个字符。

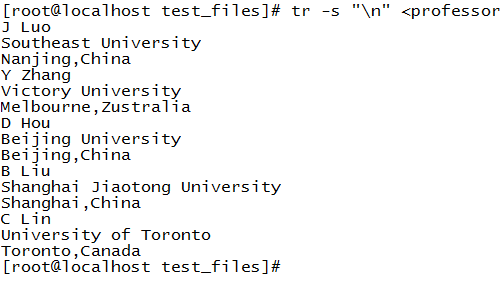
****

## 8、tr命令

tr -d a-zA-z < sturecord ：删除文件中所有大小写字母

****

tr -s "\n" <professor ：将连续的换行符替换为一个换行符，类似于去掉空行。-s将连续的相同字符去重。

****

使用前面示例中的input文件

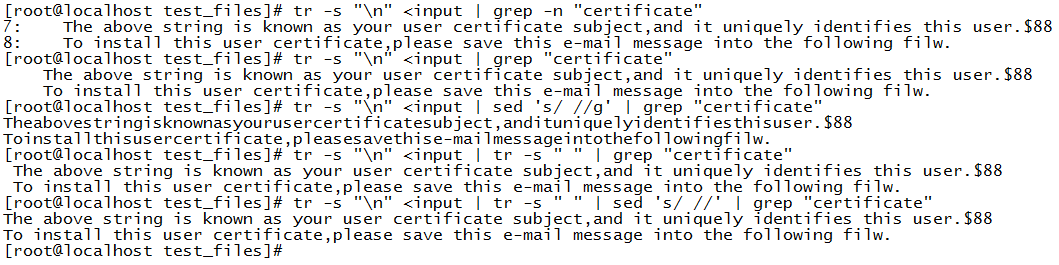
tr -s "\n" <input | grep -n "certificate" ：去掉空行，查询匹配certificate的项，并显示行号

tr -s "\n" <input | grep "certificate" ：去掉空行，查询匹配certificate的项

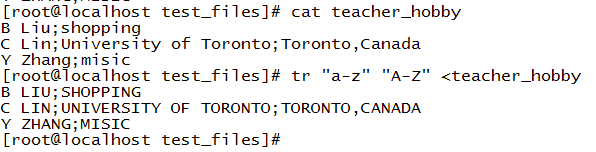
tr -s "\n" <input | sed 's/ //g' | grep "certificate" ：去掉空行，去掉空格，查询匹配certificate的项

tr -s "\n" <input | tr -s " " | grep "certificate" ：去掉空行，将连续的空格用一个空格替换，单个空格保持不变，查询匹配certificate的项

tr -s "\n" <input | tr -s " " | sed 's/ //' | grep "certificate" ：去掉空行，将连续的空格用一个空格替换，单个空格保持不变，并去掉行首的空格，查询匹配certificate的项

****

tr "a-z" "A-Z" <teacher\_hobby ：将小写字母替换为大写字母

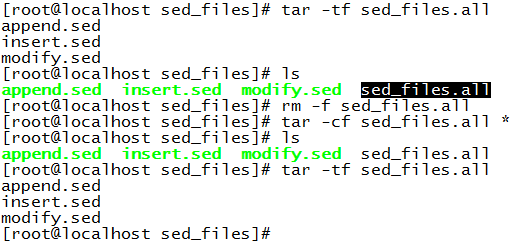
****

## 9、tar命令

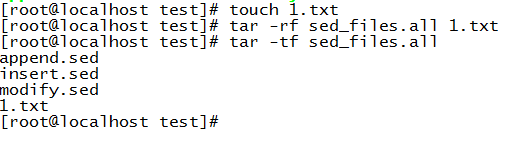
|  |  |
| --- | --- |
| tar命令选项及其意义 | |
| 选项 | 意义 |
| -c | 创建新的包 |
| -r | 为包添加新文件 |
| -t | 列出包的内容 |
| -u | 更新包的文件，若包中没有该文件，则添加进去 |
| -f | 使用压缩文件或设备，该选项通常是必须 |
| -x | 解压缩文件 |
| -v | 详细报告tar处理的信息 |
| -z | 使用gzip压缩和解压缩文件，若加上该选项创建压缩包，解压缩时必须使用该选项 |

tar -cf sed\_files.all \* ：将当前目录下的所有文件压缩到sed\_files.all

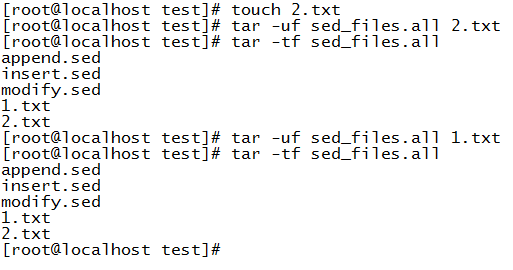
tar -tf sed\_files.all ：查看压缩文件中的文件名

****

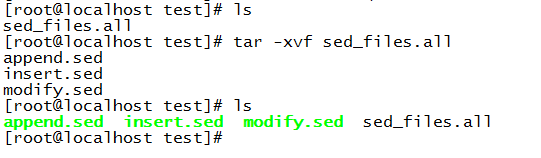
tar -rf sed\_files.all 1.txt ：向压缩包里面添加新文件

****

tar -uf sed\_files.all 2.txt ：更新压缩包里面的文件，如果新文件在压缩包中不存在，则添加进压缩包。

****

tar -xvf sed\_files.all ：解压缩到当前路径下，并显示解压缩详情

****

# 八、变量

## 1、本地变量以及本地变量替换和赋值

values1=23 ：为变量values1赋值

echo $values1 ：输出变量values1的值

echo “$values1” ：输出变量values1的值等同于echo $values1，等同于echo ${values1}

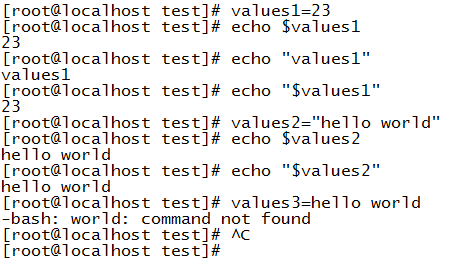
echo “values1” ：输出变量values1的变量名

valuse2="hello world" ：为变量valuse2赋值

values3=hello world ：为变量values3赋值，但是赋值失败。

注意：

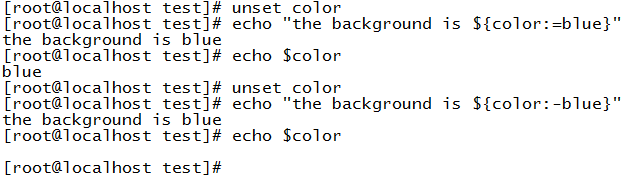
1. 变量名可以由字母，数字，下划线组成，但是数字不能作为变量的首个字符。
2. 为变量赋值时，如果值中间有空格，需要用””包含起来。
3. 本地变量的作用域只在当前脚本中，类似于C#方法中的变量，环境变量则类似于方法外的变量，多个方法都可以调用。

****

unset color ：清除变量值

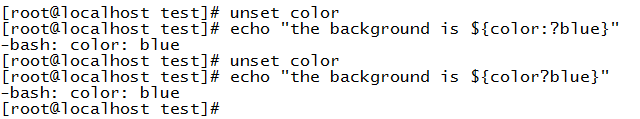
echo "the background is ${color:=blue}" ：”:=”当变量没有值的时候，给变量赋值

echo "the background is ${color:-blue}" ：”:-”当变量没有值的时候，给变量赋值，但值不会保存

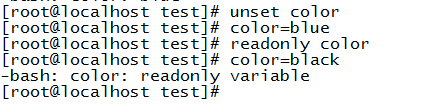
****

echo "the background is ${color:?blue}" ：判断变量是否赋值

echo "the background is ${color?blue}" ：判断变量是否赋值

****

readonly color ：设置变量只读

****

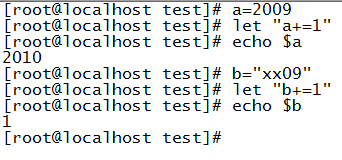
## 2、变量无类型

以下的例子可以看出，先为变量a赋值，该值为一个整数，然后a加1，打印a的值为2010。

为变量b赋值为一个字符串，然后b加1，打印b的值为1。

注意：

1. shell脚本中，为变量赋值，变量是没有类型的，它的类型跟它赋值之后的操作相关。
2. 当一个变量赋值为字符串，但却使用算术运算，那么这个变量在运算之前会初始化，初始化的值为0。
3. 变量可以不定义，而直接赋值使用。

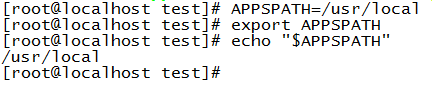
****

## 3、环境变量

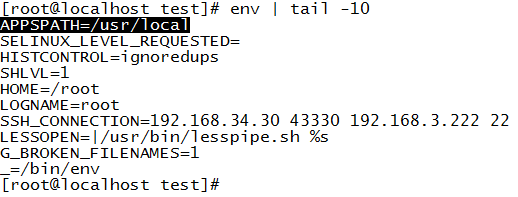
环境变量也是无类型的，它可以作用于所有登录进程所产生的子进程。

APPSPATH=/usr/local ：变量赋值

export APPSPATH ：声明变量为环境变量

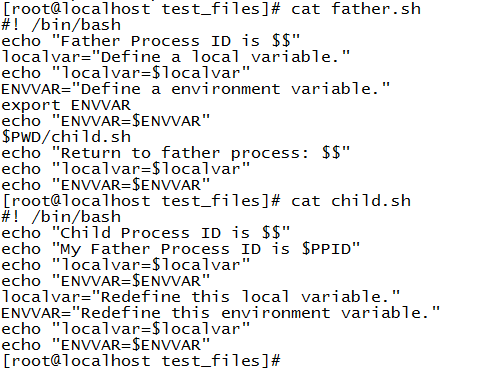
****

env ：显示所有环境变量，如下图，上一步所定义的环境变量APPSPATH，清除环境变量与清除临时变量一样。

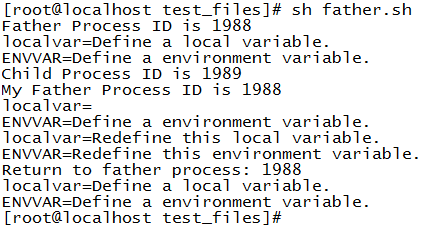
****

## 4、shell脚本嵌套

示例脚本：

****

执行结果：

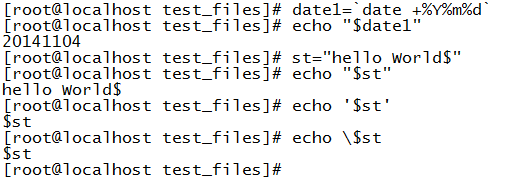
****

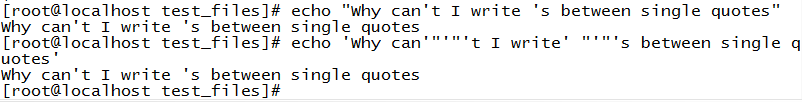
1. 本地变量的作用域只存在于当前脚本。
2. 环境变量的作用域存在于所有连接进程中。
3. $$ ：当前进程ID
4. $PPID ：创建当前进程的父进程ID

## 5、引用

|  |  |  |
| --- | --- | --- |
| 引用符号、名称及其意义 | | |
| 符号 | 名称 | 意义 |
| “” | 双引号 | 引用除$，``,\之外的所有字符 |
| ‘’ | 单引号 | 引用所有字符 |
| `` | 反引号 | shell将反引号之间的内容解释为系统命令 |
| \ | 反斜杠 | 转义符，屏蔽下一个字符的特殊含义 |

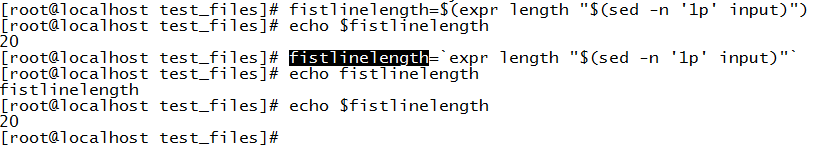
示例如下：

****

****

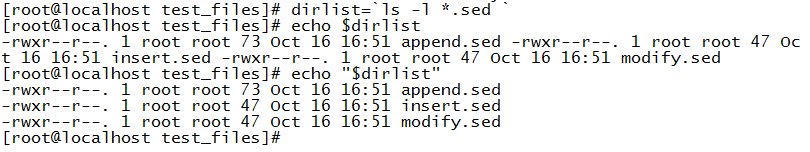
## 6、命令替换

在第5节中，示例使用``作为命令替换，另一种命令替换则是$(命令)。$(命令)可以嵌套$(命令)

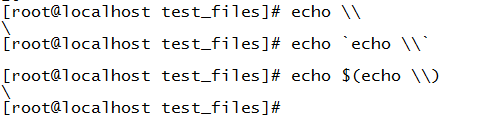
****

下面的示例中，可以得出两点：

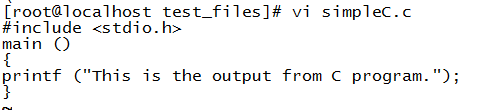
1. 字符分割，””可以保留分隔符
2. 变量无类型，可以把shell命令执行的结果赋值给变量。

****

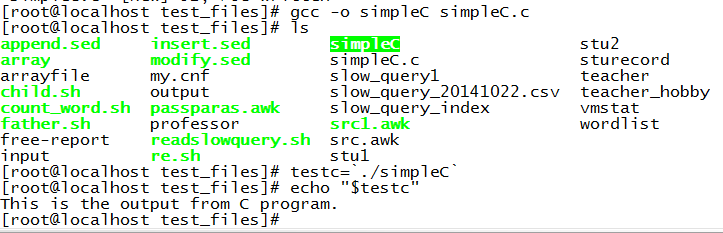
1. ``,\,$()的使用，\\转义\为普通字符，``处理\\为空字符，$()处理\\为普通字符\。

****

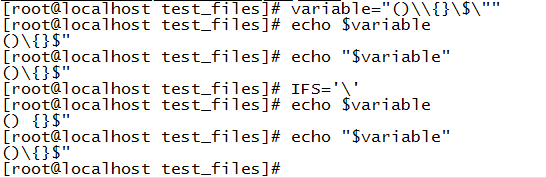
示例为一段C语言的脚本，打印一句话：This is the output from C program.

****

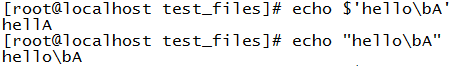
编译脚本，并将脚本执行结果赋值给变量testc，然后打印testc的值。

****

转义符的使用

****

$’’的用法

****