**Shell脚本常用整理**

**1.Sed**

**1.1 定义以及常用参数**

Sed是一种流编辑器，它是文本处理中非常有用的工具。处理时把当前处理的行存储再临时缓冲区中，称为【模式空间】，接着用sed命令处理缓冲区的内容，处理完成后，把缓冲区的内容送往屏幕。接着处理下一行，这样不断重复直到文件末尾。

sed [options] ‘command’ file

Function：

-d 删除，删除选择的行

-s 替换指定字符。通常跟正则表达式。

-p 打印指定的行 通常和 sed -n 一起运行。

！表示后面的命令对所有没有被选中的行发生作用

= 打印当前行行号

# 把注释扩展到第一个换行符以前

c\ 用新文本替换定位文本

q 第一个模式匹配完成后退出

/g 通常和s一起用，表示file所有匹配的都替换

Options:

-n 读取下一行，用下一个命令处理新的行而不是第一个命令

-i 直接修改读取的文件内容，而不是输出到终端。 -i file 则function修改了文档。

~~-f 直接讲sed的动作写入一个文件内。 -f filename. 将动作命令写在文件里。~~

-e 直接在命令行模式上进行sed动作编辑，此为默认项。

Sed替换标记



sed元字符 (匹配都是放在/ / 之间。)

^ 匹配行开始，如：/^sed/匹配所有sed开头的行

$ 匹配行结束，如：/sed$/匹配所有以sed结尾的行。

. 匹配一个非换行符的任意字符，如：/s.d/匹配s后接任意一个字符，最后是d。

\* 匹配0个或多个字符，如：/\*sed/匹配所有模板是一个或多个空格后紧跟sed的行。

[] 匹配一个指定范围内的字符，如/[sS]ed/匹配sed和Sed。

[^] 匹配一个不在指定范围内的字符，如：/[^A-RT-Z]ed/匹配不包含A-R和T-Z的一个 字母开头，紧跟ed的行。

(..) 匹配子串，保存匹配的字符，如s/(love)able/\1rs，loveable被替换成lovers.

& 保存搜索字符用来替换其他字符，如s/love/&/

< 匹配单词的开始，如：/<love/匹配包含以love开头的行。

> 匹配单词的结束，如/love>/匹配包含以love结尾的单词的行。

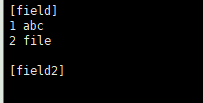
x{m} 重复字符x, m次，如：/0{5，}/ 匹配至少5个0的行。

X{m,} 重复字符x,至少m次，如：/0{5,}/ 匹配至少又5个0的行。

X{m,n} 重复字符x,至少m次，不多于n次如：/0{5，10}/

**1.2 常用模式举例**

1.txt 如下



1. 查找某行，输出行号和行内容



-n表示当前行，p表示打印当前行



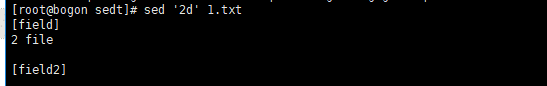
= 打印行号。

1. 打印2-3行。

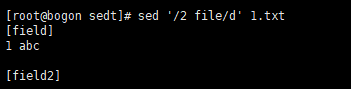


如果要遍历每一行，则不用指定行号。Sed ‘p’ 1.txt

1. 删除某一行

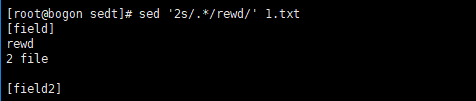


-d 删除第2行。



删除匹配行

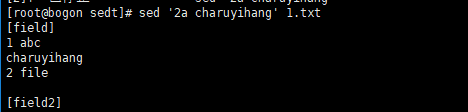
1. 替换某一行 （需要指定行号）



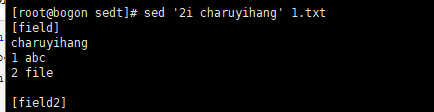
-s 替换部分，这里使用正则，第2行全部替换。行号s/源串/新串

Sed ‘2c xxx ’ file 也可以实现替换某一行

5）插入



在第二行后面插入一行。使用 -a参数。



在第2行前面插入一行。使用 -i参数。如果添加多行，每一行间需用\隔开。

1. 在单引号里使用参数 ‘ “xxx” ’ 单引号里包含双引号。



1. 替换某字符串（将/afc 替换成/D 其中\为转义字符）

sed -i 's/\/afc/\/D/' /etc/fstab

使用归纳：

Sed [option] ‘command’ file 。其中option指-n,-e ,-f，-r -i 这些影响形式的参数。而’’

单引号里面的称为动作。一般只对操作内容产生影响，如a(新增)，c(取代)，d(删除),i(插入)，p(打印)，s(替换。)。

Sed ‘行号a 添加的字符串1/添加的字符串2’ file

sed ‘/匹配行后/a\添加新行’ file 注意a关键字之后需要反斜杠 \

Sed ‘行号s /被替换的旧串/新串/’ file 如果不写行号，则匹配在匹配到旧串的地方替换。

Sed ‘行号p’ sed ‘/xxx/p’ 序号行或则指定行打印

Sed ‘行号d’ file sed ‘/xxx/d’ 删除指定行或则匹配行。

文件的行首添加一行

sed -i '1 i\sed command start' myfile

文件的行尾追加一行

 sed -i '$a \sed command end' myfile

for line in `cat num.csv`

do

sed -n '/^'$line'/p' 3.csv >> nt.csv

done

1. **Awk**
   1. **定义及常用参数**

Awk命令形式：

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

[-F|-f|-v] 大参数，-F 指定分隔符，-f调用脚本，-v定义变量

‘ ’ 单引号，代码引用块

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

/ / 匹配代码块，可以是正则表达式。

{ } 命令代码块，包含一条或多条命令。中间的{}对每行数据都执行。

; 多条命令使用分号。{ xxx; xxx}

END 结尾代码块。在遍历完文本所有行以后执行。

内置变量：

$0 表示整个当前行

$1 $1,$2...表示当前行的列

NF 字段数量变量

NR 每行的行号

FNR 与NR类似，不多多文件记录不递增，每个文件都是从1开始的

\t 制表符

\n 换行符

~ 匹配，与==相比不是精确比较

!~ 不匹配

== 等于，必须全部相等，精确比较

!= 不相等，精确比较

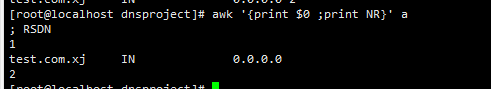
&& 逻辑与

|| 逻辑或

+ 匹配时表示1个或1个以上

* 1. **常用模式举例**

1. **遍历打印行和行号**



1. **打印匹配行，或者其中某一列**



1. 在单引号之间引用变量。 “ ’ ”xxx ” ’ ”



该句打印特定行的第4列。

1. 打印文本有多少行。END{}



即在文本所有行处理完以后打印行号。

使用归纳：

Awk 在处理文本的时候，注重与对文本进行行分析，分析某一列或者特定的字段大小什么的，在{ } 可以写很多if-else 或者while等操作，相比于sed的匹配更加灵活。

Sed 注重对文本进行增删改查，好比数据库操作。

mysql="mysql -uroot -p123456 -DDNSinfo"

#读取配置文件入库

#参数：1） 文件名称 2) 搜索一个数据段，并入库

searchOneField()

{

filename=$1

echo "$filename"

searchfield=$2 # ;RSDN 或者 ; Coremail Customer MX

echo "$searchfield"

startno=`awk '/'"$searchfield"'/ {print NR}' $filename`

echo "$startno"

startno=$(( $startno + 1 ))

strinfo=`awk 'NR=="'"$startno"'" {print}' $filename`

while((`expr length "$strinfo"`))

do

strinfo=`awk 'NR=="'"$startno"'" {print}' $filename`

#startno=$(( $startno + 1 ))

userDomain=`awk 'NR=="'"$startno"'" {print $1}' $filename `

userIP=`awk 'NR=="'"$startno"'" {print $4}' $filename `

line=$startno

echo "strno=$startno"

echo "line= ${line}"

sql='insert into userinfo(DnsName,DnsDomain,DnsIP,Line,Isexpired) values

("'${filename}'","'${userDomain}'","'${userIP}'",'${line}',"no") '

$mysql -e "${sql}"

startno=$(( $startno + 1 ))

strinfo=`awk 'NR=="'"$startno"'" {print}' $filename`

done

}

initUserInfo()

{

searchOneField "icoremail.net.view\_chinanet" "; RSDN"

searchOneField "icoremail.net.view\_chinanet" "; Coremail Customer MX"

}

#例2 从一个文件中匹配另一个文件的信息，并打印特定属性。

#查找对应号码的行提取第三列

gettheinfo()

{

num=`expr substr $1 1 7`

#echo $num

line=`awk '/'"$num"'/{print $0}' gd.csv`

#echo line $line

colon=`echo $line|awk -F"," '{print $3 }'`

echo $colon

}

#info=`gettheinfo 1899880139207883106`

echo $info

#export -f gettheinfo

#awk '{ system("gettheinfo"$1) }' res.csv

for line in `cat res.csv`

do

echo $line

cntext=`gettheinfo $line`

if [ "$cntext" != "" ]

then

echo $cntext,$line >>result3.csv

fi

done

例3 从数据库查询文件名，名拷到相应的目录

hostname="210.21.48.69"

port="3306"

username="root"

password="123456"

Insert()

{

Oneinfo=$1

phone=`echo $line|awk -F "," '{print$1}'`

name=`echo $line|awk -F "," '{print$2}'`

speech\_id=3

task\_id=3

select\_sql="insert into txacall.ai\_callout\_tbl(speech\_craft\_id ,task\_id, phone ,name)values (${speech\_id},${task\_id},${phone},${name}) "

mysql -h${hostname} -P${port} -u${username} -p${password} -e "${select\_sql}"

}

select\_sql="select recording\_file from txacall.call\_cdr\_tbl where outbound\_label != 'G' "

result=`mysql -h${hostname} -P${port} -u${username} -p${password} -e "${select\_sql}"`

echo $result |awk -F" \*" '{

for(i=1;i<=NF;i++)

{cmd="cp /home/record/"$i;cmd=cmd" ./yanqun" ;

print cmd;system(cmd)}}'

这里利用NF 做列遍历，使用system 调用cp指令。

1. **Grep**
   1. 简介和常用参数

grep 是一种强大的文本搜索工具，它能使用正则表达式搜索文本，并把匹配的行打印出来。\

Grep [-acinv] [--color=auto] ‘搜寻的字符串’ filename

-a 将binary 文件以 text文件的方式搜寻数据。

-c 计算找到 ‘搜寻字符串’的次数。

-i 忽略大小写的不同，

-n 顺便输出行号

-v 反向选择，即显示出没有 ‘搜寻字符串’内容的那一行。

--color=auto 可以将找到的关键词部分加上颜色的显示。

* 1. 常用举例

1. grep -n '^[^a-zA-Z]' regular\_express.txt

搜索 行首不是字母的行。

1. grep -n '^$' regular\_express.txt

找出空白行

1. grep -n 'o\{2\}' regular\_express.txt

找出有连续两个oo的行。 这里{}在shell里有特殊含义，所以需要转义。

1. grep -F '\*' /etc/profile

不使用正则，淡出查找 ‘\*’.

1. grep abcd file

查找含有abcd字符的行

5）爬网页的应用，使用wget下载网页内容，然后match匹配自己需要的curl,最后下载相应的资源。

getipfile()

{

url="$1"

echo "url:$url"

wget -O arintest "$url"

filetext=`sed -n -e '/href..de.\*latest"/p' "arintest"`

line=`echo "$filetext"|wc -l`

if [ $line -ne 1 ]

then

filetext=`echo "$filetext"|sed -n -e '/href..de.\*extended.\*latest/p'`

fi

temp=`expr match "${filetext}" '.\*href="\(de.\*latest"\)'`

echo "line=$line"

filename=`expr match "${temp}" '\(.\*latest\)'`

fullurl="$url""/$filename"

echo "fullurl=$fullurl"

wget -N "$fullurl"

rm -f arintest

return 0

}

**4.Wget**

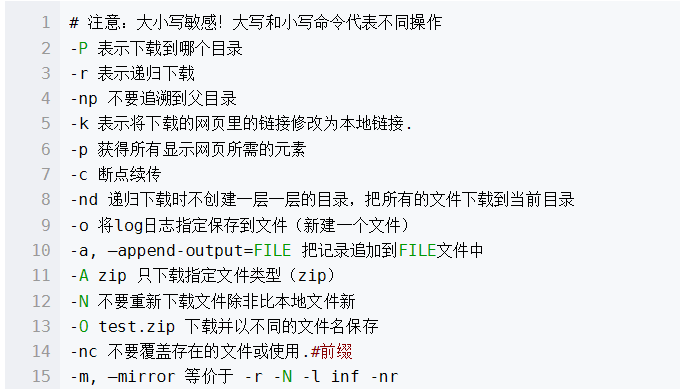
4.1 wget简介和常用参数

wget: 从网络上自动下载文件的工具。

wget [参数] URL地址

-o 使用新文件命名，但是重命名的文件如果是ts文件，则会导致文件内容有损坏而 无法打开。

-i 从指定文件获取要下载的url。如果有多个url，可以写在一个文件里，wget -i 会挨 个下载。



注意：

1. -r 参数会把网页里的url指向的内容全部下载下来，如果它引用了其他网址，会连带下载下来，慎用。
2. Wget只支持http协议，默认就下载保存为文件。是纯命令行为。而curl是libcurl支持的，支持多种协议，curl默认下载输出在控制台，需要加 -O file 保存。

**5.Curl**

5.1 简介和常用参数

在linux中curl是一个利用url规则在命令行下工作的文件传输工具，它支持文件的上传和下载，是综合传输工具。

curl [参数] url



**6.字符串拼接，裁剪，分割**

**Match**

**1) expr match “$string”, ‘\($substring\)’**

取合适的子串 小括号里的部分会保留, \( xxx\) ,其中xxx为匹配串。结果返回substring



小括号里的部分会保留，整个单引号里的为匹配串。\( 前面的内容参与匹配，但是不构成结果。

1. 字符串拼接，放在一起即可。



**3.1)将含空格的字符变成数组。**

1. **[] , (), (()), [[]]**

4.1 单小括号 ()

1. 命令组。括号中的命令将会新开一个子shell顺序执行，所以括号中的变量不能被脚本余下的部分使用。括号中多个命令用分号隔开，最后一个命令可以没有分号，各命令和括号之间不必有空格。

(cmd1;cmd2;cmd3)

1. 命令替换。等同于 ` cmd` , shell扫描一遍命令行，发现$(cmd)结构，便将$(cmd)中的cmd执行一次，得到其标准输出，再将此输出放到原来的命令。有些shell不支持，如tcsh.
2. 用于初始化数组。如 array=(a b c d)

4.2 双小括号 (())

1. 证书扩展。
2. 只要括号中的运算符，表达式符合c语言运算规则，都可用再$((exp))中，甚至是三目运算符。作不同进位运算时，输出结果都自动转换成10进制。

如：



1. 常用于算术运算比较，双括号中的变量可以不使用$ 符号前缀。括号内支持多个表达式用逗号分开。只要括号中的表达式符合c语言运算规则，

比如 for((i=0;i<5;i++)) 不适用双括号则：for i in `seq 0 4`

If(($i<5)) if [ $i -lt 5 ]

4.3 单中括号 []

1. bash的内部命令， [ 和test等同

if test $1>4 if [ $1 -gt 4 ]

1. test 和[] 中可用的比较运算符只有 == 和 != ,两者都是用于字符串比较的，不可用于整数比较，整数比较只能使用-eq -gt 这种。无论是字符串还是整数比较都不支持大于号小于号，不过字符串比较可以使用转义。如 “ab” 和 “ac” [ ab \< ac ] 结果为真。
2. 正则表达式中表示范围。
3. Array数组结构中，下标引用。

4.4 双中括号 [[exp]]

1. 支持字符串的模式匹配，使用 =~操作时甚至支持shell的正则表达式。字符串比较可以把右边的作为一个模式，而不仅仅作为一个字符串。 如：[[ hello == hell?]] 结果为真。

[[]]中匹配字符串或者通配符，不需要引号。

1. 使用[[]] 做条件判断，不使用[] ，可以避免很多错误。里面多用&&，||，<，>。

如： if[[ $a !=1 &&$a != 2]] ===> if [ $a!=1 ] &&[ $a != 2 ]

**7.数字操作**

**8.一般脚本常用操作**

2）$? 可以取到上一条命令的返回值。

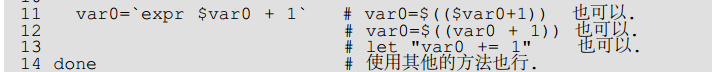
1. $# 是传给脚本的参数个数
2. $0 是脚本本身的名字
3. $1 是传递给该shell脚本的第一个参数
4. $2 是传递给该shell脚本的第二个参数
5. $@ 是传给脚本的所有参数的列表
6. $\* 是以一个单字符串显示所有向脚本传递的参数，与位置变量不同，参数可超过9个
7. $$ 是脚本运行的当前进程ID号
8. $? 是显示最后命令的退出状态，0表示没有错误，其他表示有错误
9. 执行数字加1的方法。

Var0=`expr $Var0 +1`

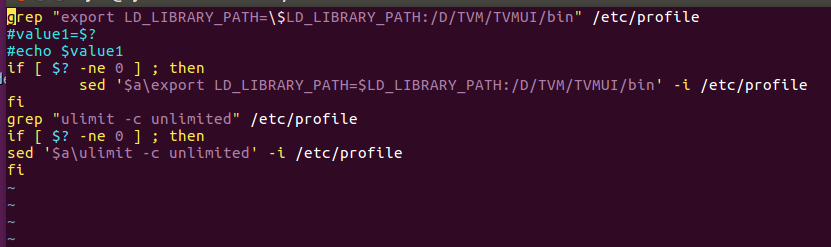
Var0=$(($Var0+1))

Var0=$((Var0+1))

Let “Var0+=1”



例1：在文件末尾添加一行，且二次执行不重复添加



1. 常用示例
   1. 比较

