老男孩教育
oldboyedu.com

跟老男孩学三剑客命令

你知道了，我做到了

26 期学员侯鹏飞　编著

Follow The Oldboy Study Linux Commands You Know But I Did

- 中国运维领域的专家、老男孩教育创始人拥有 10 多年的一线运维架构教学经验
- 本书不仅详细记录了每个命令的语法、参数以及功能描述同时配套了大量的实例和在生产环境下用过的案例
- 来老男孩教育不仅是你学到了更多的东西，同时也是你人生的新起点，你值得拥有

老男孩教育出版社 V1.3
Oldboyedu.com

目录

第一章：Awk 讲解

1.1.Awk 的讲解

awk 是一种很棒的语言，它适合文本处理和报表生成，其语法较为常见，借鉴了某些语言的一些精华，如 C 语言等。在 linux 系统日常处理工作中，发挥很重要的作用，掌握了 awk 将会使你的工作变的高大上。awk 是三剑客的老大，利剑出鞘，必会不同凡响。

1.2.Awk 实战讲解

1.2.1.awk 的原理

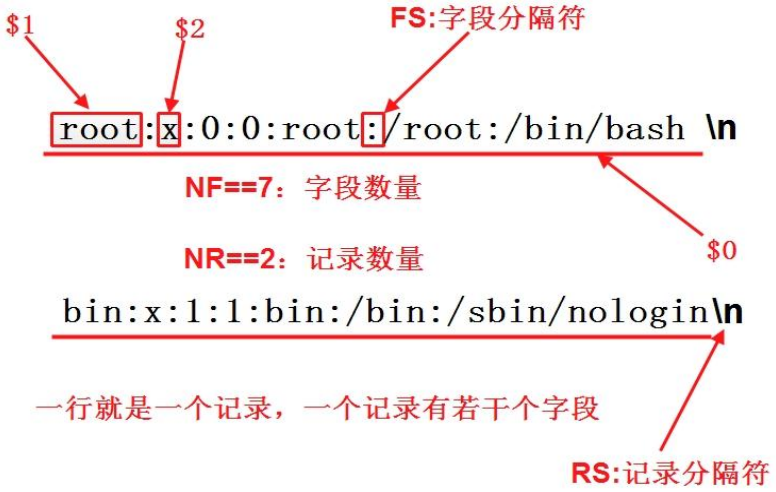
通过一个简短的命令，我们来了解其工作原理。

```
awk '{print $0}' /etc/passwd
echo hhh|awk '{print "hello,world"}'
awk '{ print "hiya" }' /etc/passwd
```

你将会见到/etc/passwd 文件的内容出现在眼前。现在，解释 awk 做了些什么。调用 awk 时，我们指定/etc/passwd 作为输入文件。执行 awk 时，它依次对/etc/passwd 中的每一行执行 print 命令。所有输出都发送到 stdout，所得到的结果与执行 cat /etc/passwd 完全相同。

现在，解释{ print }代码块。在 awk 中，花括号用于将几块代码组合到一起，这一点类似于 C 语言。在代码块中只有一条 print 命令。在 awk 中，如果只出现 print 命令，那么将打印当前行的全部内容。

再次说明，awk 对输入文件中的每一行都执行这个脚本。




```
$ awk -F":" '{ print $1 }' /etc/passwd
$ awk -F":" '{ print $1 $3 }' /etc/passwd
$ awk -F":" '{ print $1 " " $3 }' /etc/passwd
$ awk -F":" '{ print "username: " $1 "\t\tuid:" $3" }' /etc/passwd
```

1.2.2.BEGIN 和 END 模块

通常，对于每个输入行，awk 都会执行每个脚本代码块一次。然而，在许多编程情况中，可能需要在 awk 开始处理输入文件中的文本之前执行初始化代码。对于这种情况，awk 允许您定义一个 BEGIN 块。我们在前一个示例中使用了 BEGIN 块。因为 awk 在开始处理输入文件之前会执行 BEGIN 块，因此它是初始化 FS（字段分隔符）变量、打印页眉或初始化其它在程序中以后会引用的全局变量的极佳位置。

awk 还提供了另一个特殊块，叫作 END 块。awk 在处理了输入文件中的所有行之后执行这个块。通常，END 块用于执行最终计算或打印应该出现在输出流结尾的摘要信息。

1.2.3.运算符

表 1.2.3.awk 运算符表

运算符	描　述
赋值运算符	
= += -= *= /= %= ^= **=	赋值语句
逻辑运算符	
||	逻辑或
&&	逻辑与
正则运算符	
~　!~	匹配正则表达式和不匹配正则表达式
关系运算符	
< <= > >= != ==	关系运算符
算术运算符	
+　-	加，减
* / &	乘，除与求余
+ - !	一元加，减和逻辑非
^ ***	求幂
++ --	增加或减少，作为前缀或后缀
其他运算符	
$	字段引用
空格	字符串链接符
?:	三目运算符
In	数组中是否存在某键值

- awk 赋值运算符

  a+5;等价于：a=a+5;其他同类

```
[root@yum tmp]# awk 'BEGIN{a=5;a+=5;print a}'
10
```

- awk 逻辑运算符

```
[root@yum tmp]# awk 'BEGIN{a=1;b=2;print (a>2&&b>1,a=1||b>1)}'
0 1
```

- awk 正则运算符

```
[root@yum tmp]# awk 'BEGIN{a="100testaaa";if(a~/100/){print "ok"}}'
```

```
ok
[root@yum tmp]# echo|awk 'BEGIN{a="100testaaa"}a~/100/{print "ok"}'
ok
```

- 关系运算符

  如：> < 可以作为字符串比较，也可以用作数值比较，关键看操作数如果是字符串就会转换为字符串比较。两个都为数字 才转为数值比较。字符串比较：按照ascii码顺序比较。

```
[root@yum tmp]# awk 'BEGIN{a="11";if(a>=9){print "ok"}}'
[root@yum tmp]# awk 'BEGIN{a=11;if(a>=9){print "ok"}}'
ok
[root@yum tmp]# awk 'BEGIN{a;if(a>=b){print "ok"}}'
ok
```

- awk 算术运算符

  说明，所有用作算术运算符 进行操作，操作数自动转为数值，所有非数值都变为0。

```
[root@yum tmp]# awk 'BEGIN{a="b";print a++,++a}'
0 2
[root@yum tmp]# awk 'BEGIN{a="20b4";print a++,++a}'
20 22
```

- 其他运算符

  ?:三目运算符

```
[root@yum tmp]# awk 'BEGIN{a="b";print a=="b"?"ok":"err"}'
ok
[root@yum tmp]# awk 'BEGIN{a="b";print a=="c"?"ok":"err"}'
err
```

  in 运算符见后面数组

1.2.4.常用 awk 内置变量

表 1.2.4.awk 内置变量

变量名	属　性
$0	当前记录
$1~$n	当前记录的第 n 个字段
FS	输入字段分隔符 默认是空格
RS	输入记录分割符 默认为换行符
NF	当前记录中的字段个数，就是有多少列
NR	已经读出的记录数，就是行号，从 1 开始
OFS	输出字段分隔符 默认也是空格
ORS	输出的记录分隔符 默认为换行符

注：内置变量很多，参阅相关资料

- 字段分隔符 FS

  FS="\t" 一个或多个 Tab 分隔

```
[root@yum tmp]# awk 'BEGIN{FS="\t+"}{print $1,$2,$3}' tab.txt
ww CC IDD
[root@yum tmp]# cat tab.txt
```

```
ww          CC          IDD
[root@yum tmp]# awk 'BEGIN{FS="\t+"}{print $1,$2,$3}' tab.txt
ww CC IDD
[root@yum tmp]#
```

FS="[[:space:]+]" 一个或多个空白空格，默认的

```
[root@yum tmp]# cat space.txt
we are       studing awk now!
[root@yum tmp]# awk -F [[:space:]+] '{print $1,$2,$3,$4,$5}' space.txt
we are
[root@yum tmp]# awk -F [[:space:]+] '{print $1,$2}' space.txt
we are
```

FS="[" ":]+" 以一个或多个空格或：分隔

```
[root@yum tmp]# cat hello.txt
root:x:0:0:root: /root:/bin/bash
[root@yum tmp]# awk -F [" ":]+ '{print $1,$2,$3}' hello.txt
root x 0
```

- 字段数量 NF

```
[root@yum tmp]# cat hello.txt
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin:888
[root@yum tmp]# awk -F ":" 'NF==8{print $0}' hello.txt
bin:x:1:1:bin:/bin:/sbin/nologin:888
[root@yum tmp]#
```

- 记录数量 NR

```
[root@yum tmp]# ifconfig eth0| awk -F [" ":]+ 'NR==2{print $4}'
192.168.68.33
```

- RS 记录分隔符变量

  将 FS 设置成"\n"告诉 awk 每个字段都占据一行。通过将 RS 设置成""，还会告诉 awk 每个地址记录都由空白行分隔。

```
[root@yum tmp]# cat recode.txt
Jimmy the Weasel
100 Pleasant Drive
San Francisco, CA 12345
                                        #此处是空白行
Big Tony
200 Incognito Ave.
Suburbia, WA 67890
[root@yum tmp]# cat awk.txt
#!/bin/awk
BEGIN {
        FS="\n"
        RS=""
}
```

```
{
	print $1 ", " $2 ", " $3
}
[root@yum tmp]# awk -f awk.txt recode.txt
Jimmy the Weasel, 100 Pleasant Drive, San Francisco, CA 12345
Big Tony, 200 Incognito Ave., Suburbia, WA 67890
```

- OFS 输出字段分隔符

```
[root@yum tmp]# cat hello.txt
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin:888
[root@yum tmp]# awk 'BEGIN{FS=":"}{print $1","$2","$3}' hello.txt
root,x,0
bin,x,1
[root@yum tmp]# awk 'BEGIN{FS=":";OFS="#"}{print $1,$2,$3}' hello.txt
root#x#0
bin#x#1
```

- ORS 输出记录分隔符

```
[root@yum tmp]# cat recode.txt
Jimmy the Weasel
100 Pleasant Drive
San Francisco, CA 12345

Big Tony
200 Incognito Ave.
Suburbia, WA 67890

[root@yum tmp]# cat awk.txt
#!/bin/awk
BEGIN {
	FS="\n"
	RS=""
	ORS="\n\n"
}
{
	print $1 ", " $2 ", "$3
}

[root@yum tmp]#    awk -f awk.txt recode.txt
Jimmy the Weasel, 100 Pleasant Drive, San Francisco, CA 12345

Big Tony, 200 Incognito Ave., Suburbia, WA 67890

[root@yum tmp]#
```

1.2.5.awk 正则

表 1.2.5.awk 正则表达式

元字符	功能	示例	解释
^	行首定位符	/^root/	匹配所有以 root 开头的行
$	行尾定位符	/root$/	匹配所有以 root 结尾的行
.	匹配任意单个字符	/r..t/	匹配字母 r,然后两个任意字符，再以 l 结尾的行，比如 root,r33l 等
*	匹配 0 个或多个前导字符(包括回车)	/a*ool/	匹配 0 个或多个 a 之后紧跟着 ool 的行，比如 ool，aaaaool 等
+	匹配 1 个或多个前导字符	/a+b/	匹配 1 个或多个 a 加 b 的行，比如 ab,aab 等
?	匹配 0 个或 1 个前导字符	/a?b/	匹配 b 或 ab 的行
[]	匹配指定字符组内的任意一个字符	/^[abc]	匹配以字母 a 或 b 或 c 开头的行
[^]	匹配不在指定字符组内任意一个字符	/^[^abc]/	匹配不以字母 a 或 b 或 c 开头的行
()	子表达式组合	/(rool)+/	表示一个或多个 rool 组合，当有一些字符需要组合时，使用括号括起来
|	或者的意思	/(root)|B/	匹配 root 或者 B 的行
\	转义字符	/a\/\//	匹配 a//
~,!~	匹配，不匹配的条件语句	$1~/root/	匹配第一个字段包含字符 root 的所有记录
x{m} x{m,} X{m,n}	x 重复 m 次 x 重复至少 m 次 x 重复至少 m 次，但不超过 n 次 需要指定参数：-posix 或者--re-interval 没有该参数不能使用该模式	/(root){3}/ /(root){3,}/ /(root){5,6}/ cat rex.txt smierth,harry smierth,reru robin,tom	需要注意一点的是，root 加括号和不加括号的区别，x 可以表示字符串也可以只是一个字符，所以/root\{5\}/表示匹配 roo 再加上 5 个 t，及roottttt，/\(root\)\{2,\}/ 则表示匹配 rootrootrootroot 等 awk -posix '/er\{1,2\}/' rex.text smierth,harry smierth,reru

✧ 正则应用

- 规则表达式

awk '/REG/{action} ' file,/REG/为正则表达式，可以将$0 中，满足条件的记录送入到：action 进行处理

```
[root@yum tmp]# awk '/root/{print $0}' passwd
root:x:0:0:root: /root:/bin/bash
operator:x:11:0:operator:/root:/sbin/nologin
[root@yum tmp]# awk  -F : '$5~/root/{print $0}' passwd
root:x:0:0:root: /root:/bin/bash
[root@yum tmp]# ifconfig eth0|awk 'BEGIN{FS="[[:space:]:]+"} NR==2{print $4}'   #取出 ip
192.168.68.33
```

```
[root@yum tmp]# ifconfig eth0|awk 'BEGIN{FS="([[:space:]]|:)+"} NR==2{print $4}' #取出 ip
192.168.68.33
```

- 布尔表达式

awk '布尔表达式{action}' file 仅当对前面的布尔表达式求值为真时，awk 才执行代码块。

```
[root@yum tmp]# awk -F: '$1=="root"{print $0}' passwd
root:x:0:0:root: /root:/bin/bash
[root@yum tmp]# awk -F: '($1=="root")&&($5=="root"){print $0}' passwd
root:x:0:0:root: /root:/bin/bash
```

1.3.awk 的 if、循环和数组

1.3.1 条件语句

- awk 提供了非常好的类似于 C 语言的 if 语句。

```
{
    if ( $1== "foo" ) {
        if ( $2== "foo" ) {
            print "uno"
        } else {
            print "one"
        }
    } elseif ($1== "bar" ) {
        print "two"
    } else {
        print "three"
    }
}
```

- 使用 if 语句还可以将代码：

```
! /matchme/ { print $1 $3 $4 }
```

转换成：

```
{
    if ( $0 !~ /matchme/ ) {
        print $1 $3 $4
    }
}
```

1.3.2.循环结构

我们已经看到了 awk 的 while 循环结构，它等同于相应的 C 语言 while 循环。awk 还有"do...while"循环，它在代码块结尾处对条件求值，而不像标准 while 循环那样在开始处求值。它类似于其它语言中的"repeat...until"循环。以下是一个示例：

do...while 示例

```
{
    count=1
```

```
    do {
        print "I get printed at least once no matter what"
    } while ( count !=1 )
}
```

与一般的 while 循环不同，由于在代码块之后对条件求值，"do…while"循环永远都至少执行一次。换句话说，当第一次遇到普通 while 循环时，如果条件为假，将永远不执行该循环。

✧ for 循环

awk 允许创建 for 循环，它就象 while 循环，也等同于 C 语言的 for 循环：

```
for ( initial  assignment;  comparison;  increment ) {
    code  block
}
```

以下是一个简短示例：

```
for ( x=1;x<=4;x++ ) {
    print "iteration",  x
}
```

此段代码将打印：

```
iteration1
iteration2
iteration3
iteration4
```

✧ break 和 continue

此外，如同 C 语言一样，awk 提供了 break 和 continue 语句。使用这些语句可以更好地控制 awk 的循环结构。以下是迫切需要 break 语句的代码片断：

```
while 死循环
while (1) {
    print "forever and ever…"
}
while 死循环 1 永远代表是真，这个 while 循环将永远运行下去。
```

以下是一个只执行十次的循环：

```
#break 语句示例
x=1
while(1) {
    print "iteration", x
    if ( x==10 ) {
        break
    }
    x++
}
```

这里，break 语句用于“逃出”最深层的循环。"break"使循环立即终止，并继续执行循环代码块后面的语句。

continue 语句补充了 break，其作用如下：

```
x=1
```

```
while (1) {
      if ( x==4 ) {
            x++
            continue
      }
      print "iteration", x
      if ( x>20 ) {
            break
      }
      x++
}
```

这段代码打印"iteration1"到"iteration21"，"iteration4"除外。如果迭代等于 4，则增加 x 并调用 continue 语句，该语句立即使 awk 开始执行下一个循环迭代，而不执行代码块的其余部分。如同 break 一样，continue 语句适合各种 awk 迭代循环。在 for 循环主体中使用时，continue 将使循环控制变量自动增加。以下是一个等价循环：

```
for ( x=1;x<=21;x++ ) {
      if ( x==4 ) {
            continue
      }
      print "iteration", x
}
```

在 while 循环中时，在调用 continue 之前没有必要增加 x，因为 for 循环会自动增加 x。

1.3.3.数组

- 数组

  AWK 中的数组都是关联数组,数字索引也会转变为字符串索引

```
{
cities[1]=”beijing”
cities[2]=”shanghai”
cities[“three”]=”guangzhou”
for( c in cities) {
print cities[c]
}
print cities[1]
print cities[“1”]
print cities[“three”]
}
```

for…in 输出，因为数组是关联数组，默认是无序的。所以通过 for…in 得到是无序的数组。如果需要得到有序数组，需要通过下标获得。

✧ 数组的典型应用

- 用 **awk** 中查看服务器连接状态并汇总

```
netstat -an|awk '/^tcp/{++s[$NF]}END{for(a in s)print a,s[a]}'
ESTABLISHED 1
LISTEN 20
```

统计 **web** 日志访问流量，要求输出访问次数，请求页面或图片，每个请求的总大小，总访问流量的大小汇总

```
awk '{a[$7]+=$10;++b[$7];total+=$10}END{for(x in a)print b[x],x,a[x]|"sort -rn -k1";print "total size is :"total}' /app/log/access_log
total size is :172230
21 /icons/poweredby.png 83076
14 / 70546
8 /icons/apache_pb.gif 18608
```

a[$7]+=$10 表示以第 7 列为下标的数组（$10 列为$7 列的大小），把他们大小累加得到$7 每次访问的大小，后面的 for 循环有个取巧的地方，a 和 b 数组的下标相同，所以一条 for 语句足矣

1.4 常用字符串函数

1.4.1awk 常用函数表

表 1.4.awk 常用字符串函数

函数	说明
gsub(Ere, Repl, [In])	除了正则表达式所有具体值被替代这点，它和 sub 函数完全一样地执行，。
sub(Ere, Repl, [In])	用 Repl 参数指定的字符串替换 In 参数指定的字符串中的由 Ere 参数指定的扩展正则表达式的第一个具体值。sub 函数返回替换的数量。出现在 Repl 参数指定的字符串中的 &（和符号）由 In 参数指定的与 Ere 参数的指定的扩展正则表达式匹配的字符串替换。如果未指定 In 参数，缺省值是整个记录（$0 记录变量）。
index(String1, String2)	在由 String1 参数指定的字符串（其中有出现 String2 指定的参数）中，返回位置，从 1 开始编号。如果 String2 参数不在 String1 参数中出现，则返回 0（零）。
length [(String)]	返回 String 参数指定的字符串的长度（字符形式）。如果未给出 String 参数，则返回整个记录的长度（$0 记录变量）。
blength [(String)]	返回 String 参数指定的字符串的长度（以字节为单位）。如果未给出 String 参数，则返回整个记录的长度（$0 记录变量）。
substr(String, M, [N])	返回具有 N 参数指定的字符数量子串。子串从 String 参数指定的字符串取得，其字符以 M 参数指定的位置开始。M 参数指定为将 String 参数中的第一个字符作为编号 1。如果未指定 N 参数，则子串的长度将是 M 参数指定的位置到 String 参数的末尾 的长度。
match(String, Ere)	在 String 参数指定的字符串（Ere 参数指定的扩展正则表达式出现在其中）中返回位置（字符形式），从 1 开始编号，或如果 Ere 参数不出现，则返回 0（零）。RSTART 特殊变量设置为返回值。RLENGTH 特殊变量设置为匹配的字符串的长度，或如果未找到任何匹配，则设置为 -1（负一）。
split(String, A, [Ere])	将 String 参数指定的参数分割为数组元素 A[1], A[2], . . ., A[n]，并返回 n 变量的值。此分隔可以通过 Ere 参数指定的扩展正则表达式进行，或用当前字段分隔符（FS 特殊变量）来进行（如果没有给出 Ere 参数）。除非上下文指明特定的元素还应具有一个数字值，否则 A 数

	组中的元素用字符串值来创建。
tolower(String)	返回 String 参数指定的字符串，字符串中每个大写字符将更改为小写。大写和小写的映射由当前语言环境的 LC_CTYPE 范畴定义。
toupper(String)	返回 String 参数指定的字符串，字符串中每个小写字符将更改为大写。大写和小写的映射由当前语言环境的 LC_CTYPE 范畴定义。
sprintf(Format, Expr, Expr, . . .)	根据 Format 参数指定的 printf 子例程格式字符串来格式化 Expr 参数指定的表达式并返回最后生成的字符串。

1.4.2 字符串函数的应用

替换

```
awk 'BEGIN{info="this is a test2010test!";gsub(/[0-9]+/,"!",info);print info}' this is a test!test!
```

在 info 中查找满足正则表达式，/[0-9]+/ 用”!”替换，并且替换后的值，赋值给 info 未给 info 值，默认是$0

查找

```
awk 'BEGIN{info="this is a test2010test!";print index(info,"test")?"ok":"no found";}'
ok #未找到，返回 0
```

匹配查找

```
awk 'BEGIN{info="this is a test2010test!";print match(info,/[0-9]+/)?"ok":"no found";}'
ok #如果查找到数字则匹配成功返回 ok，否则失败，返回未找到
```

截取

```
awk 'BEGIN{info="this is a test2010test!";print substr(info,4,10);}'
s is a tes #从第 4 个 字符开始，截取 10 个长度字符串
```

分割

```
awk 'BEGIN{info="this is a test";split(info,tA," ");print length(tA);for(k in tA){print k,tA[k];}}' 4
4 test 1 this 2 is 3 a
#分割 info,动态创建数组 tA,awk for …in 循环，是一个无序的循环。 并不是从数组下标 1…n 开始
```

第二章：Sed 讲解

2.1.Sed 命令

2.1.1.Sed 简介

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

2.1.2.sed 工作过程

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

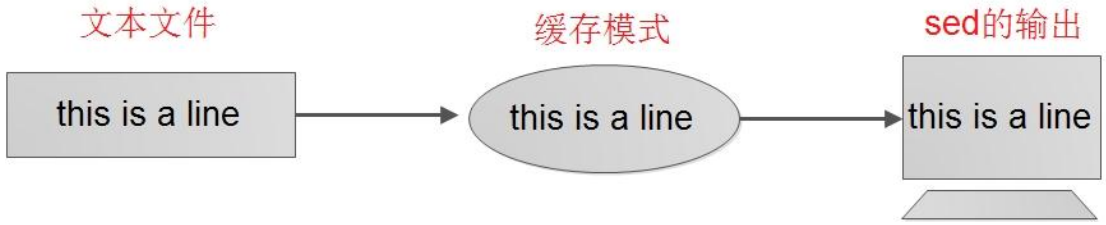


图 1

从上图可以看出 sed 不是破坏性的，它不会修改正在编辑的文件。

2.1.3.Sed 命令格式

sed 命令行格式为： sed [选项] ‘command’ 输入文本

2.1.3.1.Sed 定位

Sed 命令在没有给定的位置时，默认会处理所有行；

Sed 支持一下几种地址类型：

1、first~step

这两个单词的意思：first 指起始匹配行，step 指步长，例如：sed -n 2~5p 含义：从第二行开始匹配，隔 5 行匹配一次，即 2,7,12……。

2、$

这个$符表示匹配最后一行。

3、/REGEXP/

这个是表示匹配正则那一行，通过//之间的正则来匹配。

4、\cREGEXPc

这个是表示匹配正则那一行，通过\c 和 c 之间的正则来匹配,c 可以是任一字符

5、addr1，add2

定址 addr1，add2 决定用于对哪些行进行编辑。地址的形式可以是数字、正则表达式或二者的结合。如果没有指定地址，sed 将处理输入文件中的所有行。

如果定址是一个数字，则这个数字代表行号，如果是逗号分隔的两个行号，那么需要处理的定址就是两行之间的范围（包括两行在内）。范围可以是数字，正则或二者组合。

6、addr1，+N

从 addr1 这行到往下 N 行匹配，总共匹配 N+1 行

7、addr1，~N

Will match addr1 and the lines following addr1 until the next line whose input line number is a multiple of N.【没有看懂是什么意思】

2.1.3.2.Sed 的正则表达式

表 1：sed 的正则表达式元字符

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

2.1.3.3.Sed 的常用选项

表 2.sed 的常用选项

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

2.1.3.4.Sed 操作命令

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

表 3.sed 命令

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

表 4.替换标志

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

2.1.4.报错信息和退出信息

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

2.2.Sed 范例

2.2.1.Sed 测试实例

下面这组范例展示了如何使用 sed，包括如何使用它的选项、命令和正则表达式。记住 sed 不是破坏性的，它不会修改正在编辑的文件，除非重定向它的输出结果。

测试实例

下面给出测试文件 ceshi.txt 作为输入文件。

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

2.2.2.打印：p 命令

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

案例 1

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

说明：

默认情况下，sed 把所有输入行都打印在标准输出上。如果在某一行匹配到 north，sed

就把该行另外打印一遍。

案例 2：

```
[root@yum test]# sed -n '/north/p' sed.txt
northwest       NW      Charles Main        3.0     .98     3       34
northeast       NE      AM Main Jr.         5.1     .94     3       13
north           NO      Margot Weber        4.5     .89     5       9
```

说明：

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

2.2.3.删除：d 命令

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

案例 3：

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

说明：

删除第 3 行。默认情况下，其余的行都被打印到屏幕上。

案例 4：

```
[root@yum test]# sed '3,$d' sed.txt
northwest       NW      Charles Main        3.0     .98     3       34
western         WE      Sharon Gray         5.3     .97     5       23
```

说明：

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

案例 5：

```
[root@yum test]# sed '/north/d' sed.txt
western         WE      Sharon Gray         5.3     .97     5       23
southwest       SW      Lewis Dalsass       2.7     .8      2       18
southern        SO      Suan Chin           5.1     .95     4       15
southeast       SE      Patricia Hemenway   4.0     .7      4       17
eastern         EA      TB Savage           4.4     .84     5       20
```

```
central         CT      Ann Stephens        5.7     .94     5       13
```

说明：所有包含模式 north 的行都被动删除，其余行被打印。

2.2.4.替换：s 命令

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

案例 6：

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

说明：

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

案例 7：

```
[root@yum test]# sed -n 's/^west/north/p' sed.txt
northern        WE      Sharon Gray         5.3     .97     5       23
```

说明：

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

案例 8：

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

说明：

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

案例 9：

```
[root@yum test]# sed -n 's/Hemenway/Jones/gp' sed.txt
southeast       SE      Patricia Jones  4.0     .7      4       17
```

说明：

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

案例 10：

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

说明：

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

案例 11：

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

说明：

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

2.2.5.指定行的范围：逗号

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

案例 12：

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

说明：

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

案例 13：

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

说明：

打印从第 5 行开始第一个以 northeast 开头的行之间的所有行。

案例 14：

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

说明：

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

2.2.6.多重编辑：e 命令

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

案例 15：

```
[root@yum test]# sed -e '1,3d' -e 's/Hemenway/Jones/' sed.txt
southern        SO      Suan Chin               5.1     .95     4       15
southeast       SE      Patricia Jones          4.0     .7      4       17
eastern         EA      TB Savage               4.4     .84     5       20
northeast       NE      AM Main Jr.             5.1     .94     3       13
```

```
north           NO      Margot Weber          4.5     .89     5       9
central         CT      Ann Stephens          5.7     .94     5       13
```

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

2.2.7.追加：a 命令

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

案例 16：

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

说明：

命令 a 用于追加。字符串 Hello，World！被加在以 north 开头，north 后面跟一个空格的各行之后。如果要追加的内容超过一行，则除最后一行外，其他各行都必须以反斜杠结尾。

2.2.8.插入：i 命令

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

案例 17：

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

说明：

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

2.2.9.修改：c 命令

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

案例 18：

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

说明：

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

2.2.10.获取下一行：n 命令

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

案例 19：

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

说明：

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

2.2.11.转换：y，命令

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

案例 20：

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

说明：

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

2.2.12.退出：q 命令

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

案例 21：

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

说明：

打印完第 5 行之后，q 让 sed 程序退出。

案例 22：

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

说明：

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

2.3.生产环境案例

2.3.1.使用 sed 命令取出 IP 地址

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

案例 23：

```
[root@yum test]# cat sed2.txt
   tody is nice day
```

```
    you can walk out on the street
    it will be import to you
[root@yum test]# sed 's/^[ ]*//' sed2.txt
tody is nice day
you can walk out on the street
it will be import to you
[root@yum test]#
或者
[root@yum test]# sed 's/^[[:space:]]*//' sed2.txt
tody is nice day
you can walk out on the street
it will be import to you
[root@yum test]#
```

删除文本中空行和空格组成的行及#号注释的行

案例 24：

```
[root@yum tmp]# grep -viE "^#|^$" ssh_config
Host *
    GSSAPIAuthentication yes
    ForwardX11Trusted yes
    SendEnv LANG LC_CTYPE LC_NUMERIC LC_TIME LC_COLLATE LC_MONETARY LC_MESSAGES
    SendEnv LC_PAPER LC_NAME LC_ADDRESS LC_TELEPHONE LC_MEASUREMENT
    SendEnv LC_IDENTIFICATION LC_ALL LANGUAGE
    SendEnv XMODIFIERS
用 sed 方法
[root@yum tmp]# grep -viE "^#|^$" ssh_config
Host *
    GSSAPIAuthentication yes
    ForwardX11Trusted yes
    SendEnv LANG LC_CTYPE LC_NUMERIC LC_TIME LC_COLLATE LC_MONETARY LC_MESSAGES
    SendEnv LC_PAPER LC_NAME LC_ADDRESS LC_TELEPHONE LC_MEASUREMENT
    SendEnv LC_IDENTIFICATION LC_ALL LANGUAGE
    SendEnv XMODIFIERS
```

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

案例 25：

```
[root@yum test]# cat sed2.txt
1tody is nice day
2you can walk out on the street
13it will be import to you
[root@yum test]# sed 's/^[0-9][0-9]*//g' sed2.txt
tody is nice day
you can walk out on the street
it will be import to you
```

2.4.总结

本章内容总结了 sed 命令的用法，前面部分是 sed 命令的语法，后面部分则主要以实际案例来说明 sed 的用法，最后面一点介绍了 sed 命令在生产实践中的运用。所谓学为练，练为战，希望大家能够将 sed 命令勤加练习，必将会在工作中有所用途，尤其是频繁的分析日志文件，Awk+Sed 是比较好的组合。最后希望本文对大家有所帮助，真正达到熟练的程度这就靠大家在工作中归纳总结了。

第三章：Grep 讲解

3.1.Grep 命令

3.1.1.Grep 命令的介绍

grep 这个命令是一个全局查找正则表达式并且打印结果行的命令。它的输入是一个文件或者是一个标准输入(stdin)，或者是一个“-”连字符(),????待查阅后解释。它的输出一般是打印在里屏幕上。grep 家族里还有 egrep 和 fgrep 这两个命令，这里我们就不做介绍了。

3.1.2.Grep 是如何工作的

grep 命令在一个或多个文件中查找某个字符模式。如果这个模式中包含空格，就必须用引号把它括起来。grep 命令中，模式可以是一个被引号括括起来的字符串，也可以是单个词，位于模式之后所有的单词都被视为文件名。grep 将输出发送到屏幕，它不会对输入文件进行任何修改或变化，下面我们以一个命令来说明。

命令格式

grep [选项] 模式 [文件....]

案例 1：

```
grep Tom /etc/passwd
```

说明

grep 将在文件中查找/etc/passwd 中查找模式 Tom。如果查找成功，文件中相应行会显示在屏幕上，如果没有找到指定的模式，就不会有任何输出，如果指定的文件不是一个合法的文件，屏幕上就会显示报错信息。如果发现了要查找的模式，grep 就返回退出状态 0，表示成功，如果没找到，返回的退出状态为 1，而找不到指定文件时，退出状态将是 2。

grep 的程序输入可以来自标准输入或管道，而不仅仅是文件。如果忘了指定文件，grep 会以为你要它从标准输入(即键盘)获取输入，于是停下来等你健入一些字符。如果输入来自管道，就会有另一条命令的输出通过管道变成 grep 命令的输入，如果匹配到要查找的模式，grep 会把输出打印在屏幕上。

案例 2：

```
ps -ef | grep root
```

ps 命令的输出被送到 grep，然后所有包含 root 的行都被打印在屏幕上。

3.2.正则表达式元字符和选项

3.2.1.正则表达式元字符

元字符也是一种字符，但他表达的含义不同于字符本身的字面含义。例如，^和$就是元字符。grep 支持很多正则表达式元字符，以便用户更精确的定义要查找模式。

表 1.2.1.正则表达式元字符

元字符	功　能	示　例	示例的匹配对象
^	行首定位符	/^love/	匹配所有以 love 开头的行
$	行尾定位符	/love$/	匹配所有以 love 结尾的行
.	匹配除换行外的单个字符	/l..e/	匹配包含字符 l、后跟两个任意字符、再跟字母 e 的行

*	匹配零个或多个前导字符	/*love/	匹配在零个或多个空格紧跟着模式 love 的行
[]	匹配指定字符组内任一字符	/[LI]ove/	匹配包含 love 和 Love 的行
[^]	匹配不在指定字符组内任一字符	/[^A-KM-Z]ove/	匹配包含 ove，但 ove 之前的那个字符不在 A 至 K 或 M 至 Z 间的行
\(..\)	保存已匹配的字符		
&	保存查找串以便在替换串中引用	s/love/**&**/	符号&代表查找串。字符串 love 将替换前后各加了两个**的引用，即 love 变成**love**
\<	词首定位符	/\<love/	匹配包含以 love 开头的单词的行
\>	词尾定位符	/love\>/	匹配包含以 love 结尾的单词的行
x\{m\}	连续 m 个 x	/o\{5\}/	分别匹配出现连续 5 个字母 o、至少 5 个连续的 o、或 5~10 个连续的 o 的行
x\{m,\}	至少 m 个 x	/o\{5,\}/	
x\{m,n\}	至少 m 个 x，但不超过 n 个 x	/o\{5,10\}/	

3.2.2.grep 选项

grep 选线用于调整执行查找或显示结果的方式。例如通过选线来关闭大小写敏感、要求显示行号，或者只显示报错信息等。

表 1.2.2.grep 选项

选　项	功　　能
-E	如果加这个选项，那么后面的匹配模式就是扩展的正则表达式，也就是 grep -E = egrep
-i	比较字符时忽略大小写区别
-w	把表达式作为词来查找，相当于正则中的"\<...\>"(...表示你自定义的规则)
-x	被匹配到的内容，正好是整个行，相当于正则"^...$"
-v	取反，也就是输出我们定义模式相反的内容
-c	count.统计，统计匹配结果的行数，主要不是匹配结果的次数，是行数。
-m	只匹配规定的行数，之后的内容就不在匹配了
-n	在输出的结果里显示行号，这里要清楚的是这里所谓的行号是该行内容在原文件中的行号，而不是在输出结果中行号
-o	只显示匹配内容，grep 默认是显示满足匹配条件的一行，加上这个参数就只显示匹配结果，比如我们要匹配一个 ip 地址，就只需要结果，而不需要该行的内容。

-R	递归匹配。如果要在一个目录中多个文件或目录匹配内容，则需要这个参数
-B	输出满足条件行的前几行，比如 grep -B 3 "aa" file 表示在 file 中输出有 aa 的行，同时还要输出 aa 的前 3 行
-A	这个与-B 类似，输出满足条件行的后几行
-C	这个相当于同时用-B -A，也就是前后都输出

3.3.使用正则表达式 grep 实例

3.3.1.grep 的测试实例

以下所有的例子都是用的 grep.txt 文件。

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

案例 1：

```
[root@yum test]# grep NW grep.txt
northwest       NW      Charles Main        3.0     .98     3       34
```

说明：

打印文件 grep.txt 文件包含正则表达式 NW 的行

案例 2：

```
[root@yum test]# grep ^n grep.txt
northwest       NW      Charles Main        3.0     .98     3       34
northeast       NE      AM Main Jr.         5.1     .94     3       13
north           NO      Margot Weber        4.5     .89     5       9
```

说明：

打印以字母 n 开头的行，(^)行首定位符

案例 3：

```
[root@yum test]# grep "4$" grep.txt
northwest       NW      Charles Main        3.0     .98     3       34
```

说明：

打印所有以数字 4 结尾的行。($)行尾定位符

案例 4：

```
[root@yum test]# grep 'TB Savage' grep.txt
eastern         EA      TB Savage       4.4     .84     5       20
```

说明：

打印所有包含 TB Savage 的行。如果不用引号(这个例子中，使用单引号或双引号都可以)，TB 和 Savage 之间的空格将导致 grep 会在 Savage 和 grep.txt 查找 TB。所以，如果字符串之间有空格，必须要用引号引起来。

案例 5：

```
[root@yum test]# grep '5\..' grep.txt
western         WE      Sharon Gray     5.3     .97     5       23
southern        SO      Suan Chin       5.1     .95     4       15
northeast       NE      AM Main Jr.     5.1     .94     3       13
central         CT      Ann Stephens    5.7     .94     5       13
```

说明：

打印所有包含数字 5，后面跟一个.号 再跟一个任意字符的行。 (.)号代表单个字符，被(\)转义后，只代表本身一个.号。

案例 6：

```
[root@yum test]# grep   '^[we]' grep.txt
western         WE      Sharon Gray     5.3     .97     5       23
eastern         EA      TB Savage       4.4     .84     5       20
```

说明：

打印所有字母 w 和 e 开头的行。[]表示任意一个字符都可以匹配。

案例 7：

```
[root@yum test]# grep '[^0-9]' grep.txt
northwest       NW      Charles Main    3.0     .98     3       34
western         WE      Sharon Gray     5.3     .97     5       23
southwest       SW      Lewis Dalsass   2.7     .8      2       18
southern        SO      Suan Chin       5.1     .95     4       15
southeast       SE      Patricia Hemenway       4.0     .7      4       17
eastern         EA      TB Savage       4.4     .84     5       20
northeast       NE      AM Main Jr.     5.1     .94     3       13
north           NO      Margot Weber    4.5     .89     5       9
central         CT      Ann Stephens    5.7     .94     5       13
```

说明：

打印包含非数字字符的行。由于至少每一行有一个非数字字符，因此说有行都被打印。

案例 8：

```
[root@yum test]# grep '[A-Z][A-Z] [A-Z]' grep.txt
eastern         EA      TB Savage       4.4     .84     5       20
northeast       NE      AM Main Jr.     5.1     .94     3       13
```

说明：

打印了包含两个大写字符、后跟一个空格和一个大写字符的行，例如 TB Savage 和 MB Main。

案例 9：

```
[root@yum test]# grep 'ss* ' grep.txt
northwest       NW      Charles Main            3.0     .98     3       34
southwest       SW      Lewis Dalsass           2.7     .8      2       18
central         CT      Ann Stephens            5.7     .94     5       13
```

说明：

打印包含一个 s、后跟 0 个或多个连着的 s 和一个空格的文本行。

案例 10：

```
[root@yum test]# grep '[a-z]\{9\}' grep.txt
northwest       NW      Charles Main            3.0     .98     3       34
southwest       SW      Lewis Dalsass           2.7     .8      2       18
southeast       SE      Patricia Hemenway       4.0     .7      4       17
northeast       NE      AM Main Jr.             5.1     .94     3       13
```

说明：

打印所有出现至少 9 个小写字母连在一起的行，例如，northwest，southwest，southeast，northeast。

案例 11：

```
[root@yum test]# grep '\(3\)\.[0-9].*\1     *\1' grep.txt
northwest       NW      Charles Main            3.0     .98     3       34
```

说明：

如果某一行包含一个 3 后面跟一个句点和一个数字，再任意多个字符(.*),然后跟一个 3 个或任意多个制表符，再接一个 3，则打印该行。

案例 12：

```
[root@yum test]# grep '\<north\>' grep.txt
north           NO      Margot Weber            4.5     .89     5       9
```

说明：

打印所有包含单词 north 的行。“\<”是词首定位符“\>”是词尾定位符。

案例 13：

```
[root@yum test]# grep '\<[a-z].*n\>' grep.txt
northwest       NW      Charles Main            3.0     .98     3       34
western         WE      Sharon Gray             5.3     .97     5       23
southern        SO      Suan Chin               5.1     .95     4       15
eastern         EA      TB Savage               4.4     .84     5       20
northeast       NE      AM Main Jr.             5.1     .94     3       13
central         CT      Ann Stephens            5.7     .94     5       13
```

说明：

打印所有包含以小写字母开头，以 n 结尾，中间由任意多个字符组成的单词的行。注意符号.*,他代表任意字符，包括空格。

3.3.2.grep 选项测试案例

测试文件同上面那个

案例 14：

```
[root@yum test]# grep -n 'north' grep.txt
1:northwest          NW     Charles Main          3.0     .98     3       34
7:northeast          NE     AM Main Jr.           5.1     .94     3       13
8:north              NO     Margot Weber          4.5     .89     5       9
```

说明：

选项-n 在找到指定模式的行前面加上其行号再一并输出。

案例 15：

```
[root@yum test]# grep -i 'pat' grep.txt
southeast          SE     Patricia Hemenway     4.0     .7      4       17
```

说明：

选项-i 关闭大小写敏感性。表达式 pat 包含任意大小写的组合都符合。

案例 16：

```
[root@yum test]# grep -v 'Suan Chin' grep.txt
northwest          NW     Charles Main          3.0     .98     3       34
western            WE     Sharon Gray           5.3     .97     5       23
southwest          SW     Lewis Dalsass         2.7     .8      2       18
southeast          SE     Patricia Hemenway     4.0     .7      4       17
eastern            EA     TB Savage             4.4     .84     5       20
northeast          NE     AM Main Jr.           5.1     .94     3       13
north              NO     Margot Weber          4.5     .89     5       9
central            CT     Ann Stephens          5.7     .94     5       13
```

说明：

这个实例中，选项-v 打印所有不含模式 Suan Chin 的行。选项-v 可用来删除输入文件汇中特定的条目。如果真要删除这些条目，就要把 grep 的输出重定向到一个临时文件中，然后把临时文件的名字改成原文件的名字。

注意不能从原文件重定向到原文件，这样会破坏原文件的。

案例 17：

```
[root@yum test]# grep -l 'SE*'    sed.txt grep.txt
sed.txt
grep.txt
```

说明：

选项-l 使 grep 只输出包含模式的文件名，而不输出文本行。

案例 18：

```
[root@yum test]# grep -c 'west' grep.txt
3
```

说明：

选项-c 让 grep 打印出含有模式的行的数目。这个数字并不代表模式的出现次数。例如，即使 west 在某行中出现 3 次，这行也只计一次。

案例 19：

```
[root@yum test]# grep -w 'north' grep.txt
north           NO      Margot Weber            4.5     .89     5       9
```

说明：

选项-w 只查找作为一个词，而不是词的一部分出现的模式。这条命令只打印包含词 north 的行，而不打印那些 northwest、northwest 等中出现的行。

3.3.3.grep 与管道

grep 的输入不一定都是文件，它也常常从管道读取输入。

案例 20：

```
[root@yum test]# ls
grep.txt    sed1.txt    sed2.txt    sed.txt
[root@yum test]# ls | grep "grep"
grep.txt
[root@yum test]# ls | grep "^gr"
grep.txt
[root@yum test]#
```

说明：

ls 的命令的输出通过管道传给 grep。输出结果字母 gr 开头的所有行都被打印出来了，也就是说，被选中的目录被打印出来了。

3.3.4.egrep 扩展

egrep 在 grep 的基础上增加了更多的元字符。但是 egrep 不允许使用\(\),\{\}.

表 1.3.4.egrep 使用的正则表达式元字符

元字符	功　能	示　例	示例的匹配对象
^	行首定位符	/^love/	匹配所有以 love 开头的行
$	行尾定位符	/love$/	匹配所有以 love 结尾的行
.	匹配除换行外的单个字符	/l..e/	匹配包含字符 l、后跟两个任意字符、再跟字母 e 的行
*	匹配零个或多个前导字符	/*love/	匹配在零个或多个空格紧跟着模式 love 的行
[]	匹配指定字符组内任一字符	/[Ll]ove/	匹配包含 love 和 Love 的行
[^]	匹配不在指定字符组内任一字符	/[^A-KM-Z]ove/	匹配包含 ove，但 ove 之前的那个字符不在 A 至 K 或 M 至 Z

			间的行
egrep 新增的元字符			
+	匹配一个或多个加号前面的字符	'[a-z]+ove'	匹配一个或多个小写字母后跟 ove 的字符串。move love approve
？	匹配 0 个或一个前导字符	'lo?ve'	匹配 l 后跟一个或 0 个字母 o 以及 ve 的字符串。love lve
a|b	匹配 a 或 b	'love|hate'	匹配 love 和 hate 这两个表达式之一
()	字符组	'love(able|ly)(ov+)'	匹配 loveable 或 lovely 匹配 ov 的一次或多次出现

案例 21：

```
[root@yum test]# grep "west|north" grep.txt
[root@yum test]# agrep "west|north" grep.txt
-bash: agrep: command not found
[root@yum test]# egrep "west|north" grep.txt
northwest       NW      Charles Main      3.0     .98     3       34
western         WE      Sharon Gray       5.3     .97     5       23
southwest       SW      Lewis Dalsass     2.7     .8      2       18
northeast       NE      AM Main Jr.       5.1     .94     3       13
north           NO      Margot Weber      4.5     .89     5       9
```

说明：

grep 不支持“|”这个，egrep 支持“|”，egrep 查到了，包含 west 或者 north 的行。