sed和awk

用于文本过滤和转换的流编辑器sed(stream editor)

sed [OPTIONS] [SCRIPT] [INPUTFILE...]

从文件INPUTFILE或标准输入读取数据，对每行数据执行相应的流编辑命令后输出

- 缺省情况下，第一个参数为sed命令，后面为要编辑的文件名
- **如果采用选项-e script 或-f scriptfile，则后面的参数为要编辑的文件名**。-e/-f选项可出现多次，所有sed命令按照出现的先后顺序组合在一起，组成sed命令集
- 如果参数部分没有指定文件名，则从标准输入读
- 如果参数包括了一个或者多个文件，则从这些文件读
- 缺省情况下流编辑后的结果输出到STDOUT

-e SCRIPT	将SCRIPT中的命令加入到流编辑命令集
-f SCRIPTFILE	将文件SCRIPTFILE中的命令加入到流编辑命令集
没有-e -f选项时	第一个参数包含了流编辑命令
-E	采用扩展正则表达式，缺省为基本正则表达式
-n, --quiet	流编辑产生的结果缺省情况下会写到标准输出。-n表示不输出，除非使用流编辑命令(p)显式输出

p - print

```
$ seq 1 5 > 1.txt
$ sed '1p' 1.txt
1
1
2
3
4
5
$ sed -n '1p' 1.txt
1
$ sed -n '1p' <1.txt
1
$ sed -ne '1p' -e '2p' 1.txt
1
2
$ sed -ne '1p;2p' 1.txt
1
2
$ echo -e '3p\n4p' > tmp.sed
$ sed -n -e '1p' -f tmp.sed 1.txt
1
3
4
```

流编辑器sed：原地编辑

- sed命令有多个文件参数时，将这些文件看成一个大的数据流，也就是说行编号是连续的
- -s, --separate 将每个文件看成一个数据流，对于每个文件执行相应的sed命令
- 缺省情况下流编辑后的结果输出到STDOUT，采用-i选项进行原地编辑，-i选项也隐含着-s选项

-s, --separate	将每个文件看成一个数据流，对于每个文件执行相应的sed命令
-i[SUFFIX]	编辑的结果覆盖原文件。如果指定SUFFIX，且其中不包含*时，首先备份为FILE.SUFFIX，如果SUFFIX中包含*，则备份的文件名中*代替以原文件名

```
$ sed -n '2p' 1.txt 1.txt
2
$ sed -s -n '2p' 1.txt 1.txt
2
2
```

sed -i 's/hello/world/' file.txt	将file.txt中的hello替换成world
sed -i.bak 's/hello/world' file.txt	原地修改file.txt，原文件备份到file.txt.bak
sed -i'sed.*' 's/hello/world' file.txt	原地修改file.txt，原文件备份到sed.file.txt
sed -i'tmp/*' 's/hello/world/' file.txt	原地修改file.txt，原文件备份到tmp/file.txt

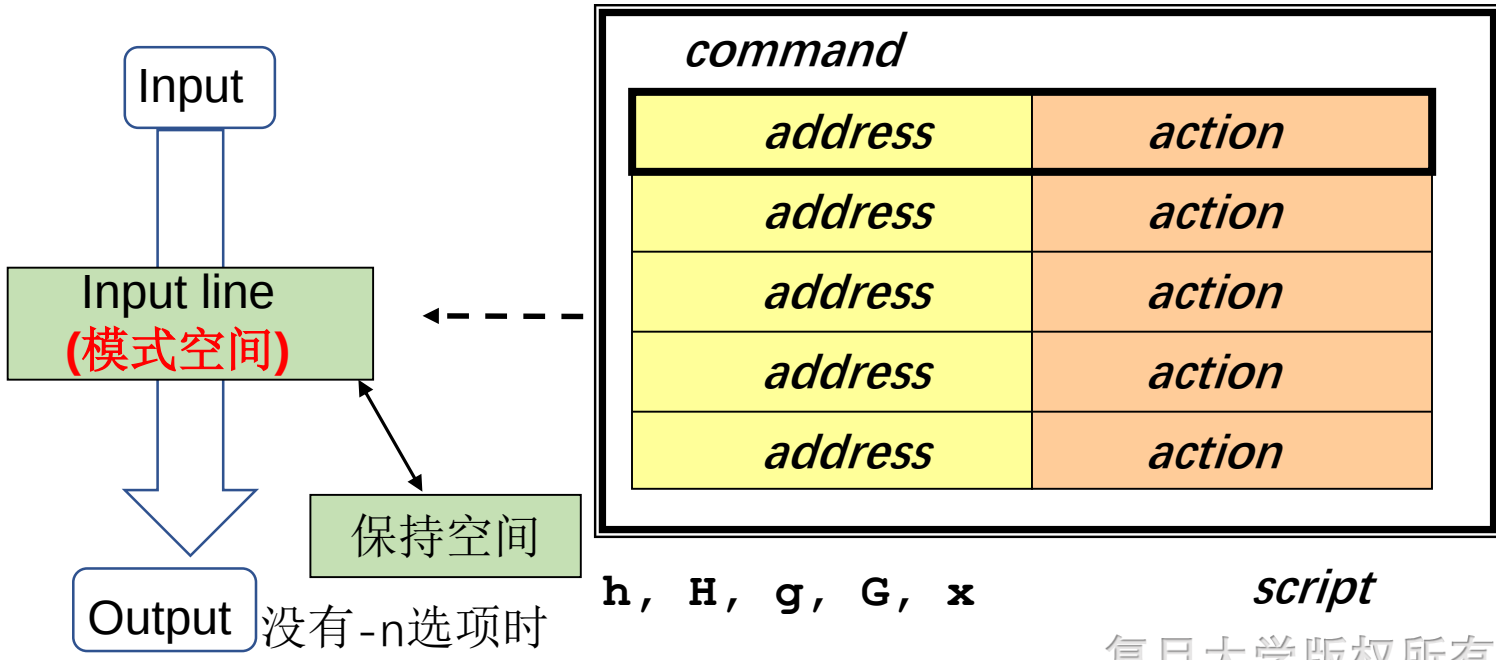
流编辑器sed：基本步骤

- 每个sed命令包括3个部分[addr]X[options]
  - addr为可选的地址部分，给出该命令可用于哪些行，不指定时表示所有行
  - X为单个字符，给出了哪个sed命令
  - options为该命令的可选参数
- sed命令之间用分号(;)或换行符(-f选项指出的脚本文件一般采用换行符分割）隔开

重复下面的步骤，直到没有更多的行或者sed命令明确退出：

- 从文件或STDIN取下一行，将最后的换行符移走后放入**模式空间(pattern space)**
- 接下来对于当前模式空间的行按照顺序检查sed命令：
  - 如果与当前行匹配，则执行该sed命令。该命令可能对模式空间的行的内容进行更新，后一个命令在前一个命令执行的基础上进一步执行
  - 如果与当前行不匹配，忽略该sed命令
- 在执行完最后一条命令之后，如果没有-n选项，则输出模式空间的行，在输出时附加换行符
- 除非特别指定(比如N/D命令），**从模式空间删除当前行**，接下来从文件或STDIN读下一行，**开启一个新的周期**，继续执行sed命令

保持(hold)空间为临时存储的空间，保存多行的内容。hHgGx命令用于操作模式空间和保持空间



流编辑器sed: 描述匹配的行的地址部分

- 如果一个命令前面没有指定地址，则表示对于所有行都执行该命令
- 地址可以指定某一行、某个范围内的行或者不在指定范围的行
  - 可以通过**行号或正则表达式**描述某一行
  - 范围是通过**逗号分割的两个行地址**来描述
  - 不在指定范围的行是通过在行地址或者范围之后**附加!**来描述

-E	采用扩展正则表达式， 缺省为基本正则表达式

地址	含义	例子
N	N为行号，表示第N行。有些情况下N还可以为0	3 表示第3行
$	最后一行	$ 表示最后一行
/REGEXP/	与正则表达式REGEXP匹配的行。 //表示重复上一个正则表达式。 对于REGEXP中的/需要进行转义，以避免认为是分隔符/	/PS1/ 表示包含PS1的行 '/\/bin/' 表示包含/bin的行
\%REGEXP%	用另一个字符作为分界符给出哪些属于REGEXP。%可是任一字符	'\$/bin$' 表示分隔符为$，匹配包含/bin的行
ADDR1, ADDR2	ADDR1行开始到ADDR2行(包含)为止，如果ADDR2超过了最大行数，或者无法找到与模式匹配的行，相当于到文件结尾。注意ADDR2为正则表达式时，表示从ADDR1给出的行的下一行开始查找	1,10表示1到10行 0,/PS/ 表示从第1行到包含PS的行(从第1行开始找)为止
ADDR1, +N	ADDR1行开始接下来的N行	10,+4 表示10到14行
ADDR!	ADDR可以是上面描述地址的方式，后面的!表示否定，即不在ADDR区域的行	1,10! 除了1到10行外 1! 除了1行外
FIRST~STEP	GNU扩展，FIRST和STEP都是正整数，从FIRST行开始，每隔STEP行	1~2 表示奇数行

流编辑器sed：sed命令

每个sed命令除了可以是单字符命令外，还可以是组命令，即通过{ }将多个命令组合在一起，中间用分号或换行符分割，表示顺序执行这些命令。组命令还允许嵌套

```
seq 1 3 | sed -n '2{s/$/;/;p}'   第2行附加;然后输出
sed -n '/^#/{{/if/p}}' ~/.bashrc
等价于grep ^# ~/.bashrc | grep if
```

p - print命令表示打印当前模式空间(到标准输出)

- 一般和-n选项结合在一起使用
- 如果没有-n选项，缺省会输出模式空间，这样会导致模式空间输出两次

```
sed -n 3,5p 输出3到5行
sed -n '/^$/,$p' 从第一个空行到最后一行间的内容输出
```

= - 行号命令表示打印当前模式空间的行号(到标准输出)

```
sed -n $=   输出最后一行的行号，相当于wc -l
sed -n '/TODO/{=;p}'   输出包含TODO的行的行号和内容
```

s - substitute	**Q** - Quit
y - transform	**a** - append
d - delete	**i** - insert
p - print	**c** - change
= - line number	**r** - read
q - quit	**w** - write

d - delete命令删除当前模式空间，后面的命令不再执行，当然也不会输出当前模式空间，取下一行到模式空间重新开始新的周期

```
sed '/^\s*$/d' 删除所有空行(除空格外没有其他字符)
sed 1,3d   删除1到3行，即第4行以及之后内容输出，相当于 tail -n +4
sed '2,4!d'   删除除2-4行外的内容，仅保留2-4行
```

流编辑器sed：sed命令

s - substitute	Q - Quit
y - transform	a - append
d - delete	i - insert
p - print	c - change
= - line number	r - read
q - quit	w - write

q - quit命令退出sed，后面还可跟退出状态码，缺省为q0，不再取新的sed命令和行，当前模式空间仍可能输出。仅支持行地址，不支持范围地址

```
sed 3q 到第3行时退出，即只处理前面的1-3行，相当于 head -n 3
```

Q - quit命令退出sed，与q类似，只是不会输出当前的模式空间。仅支持行地址，不支持范围地址

```
sed 3Q 到第3行时退出，第3行不会输出，相当于 head -n 2
```

w - write命令将当前模式空间写入到文件中

```
sed '/export/w export.txt' ~/.bashrc
将所有包含export的行写入到文件export.txt中。同时.bashrc也会输出到标准输出
```

a - append命令 附加文本

- 可以是行地址，也可是范围地址，表示对于每一行，附加相应的文本，支持\t\n等标准字符转义
- a命令并不会更新模式空间，也不改变行号计数器。在当前模式空间的命令处理结束后（包括没有-n选项时输出模式空间)才执行相应的动作
- a命令可跨越多行，如果\后面为换行，表示下一行也是命令的一部分，直到行尾不是\为止

```
sed  '/export/a#export;' ~/.bashrc
在export的行下面附加一行#export;
sed '$a\
# end script \
# last line' ~/.bashrc
在最后一行后附加2行
sed  '$a # done
1p' /etc/hosts
```

流编辑器sed：sed命令

s - substitute	Q - Quit
y - transform	a - append
d - delete	i - insert
p - print	c - change
= - line number	r - read
q - quit	w - write

r - read命令在当前行之后插入某个文件的内容 r FILE
r命令与a命令类似，不更新模式空间，在当前模式空间的命令处理结束时输出FILE中的内容。支持单行与范围地址

第一行后面插入FILE
```
sed '1r FILE'
```

i - insert命令在当前行之前插入文本
i命令与a命令类似，只是a命令附加文本，而i命令插入文本。不更新模式空间，在当前模式空间的命令处理结束之前首先插入文本。

第一行前面插入2行
```
sed '1i #!/bin/bash \
# a simple shell script'
```
前面两行都插入#test
```
sed '1,2i#test' /etc/hosts
```

c - change命令删除地址或地址范围所对应的行，输出相应的多行文本
支持单行与范围地址。c命令不会输出当前模式空间的行，而是清除模式空间，取新一行到模式空间，开启一个新的周期，直到范围内的最后一行后才会输出替代的多行文本

```
sed  '/IPv6/,$ c\
# remove ipv6-related domain name '
</etc/hosts
```
删除IPv6到最后之间的行，代替以指定的注释行

y - transform命令格式为y/set1/set2/

- 除了缺省的/外，y命令后面也可以采取其他字符作为分隔符
- 要求set1和set2长度一样，在set1中出现的字符以set2中对应位置的字符替代
- 注意y命令并不支持tr中的范围，即a-z格式
- y命令用于在某些行需要转换时，如果所有行转换，建议采用tr命令

```
sed 'y/abcd/ABCD/'
```

流编辑器sed：sed命令

s - substitute命令是最常用的sed命令，格式为 s/pattern/replace/[flags]

```
$ echo '/home2 /home2/test' | sed 's%/home2%/home%g'
$ echo '/home2 /home/test' | sed -E 's%/home2?%/usr&%g'
```

- pattern: 正则表达式描述的模式
  - s命令的分隔符缺省为/
  - 如果pattern里面有/，则需要在之前加上\进行转义
  - 也可以使用其他任意字符作为分隔符，如s%pattern%replace%flags
- replace：将模式空间中pattern匹配的内容替换为replace，其中有如下字符时：
  - \ 转义字符，比如\&表示字符&，而\\表示反斜杠\，\n表示换行符, \t表示制表符
  - &表示pattern所匹配的内容
  - \n，n为数字(1-9)，表示pattern中的第n个子模式所匹配的内容
- flags为可选，其可能的取值包括：
  - i表示匹配时大小写无关
  - g表示模式空间中的所有与pattern匹配的内容都替换，缺省仅替换第一次出现的模式
  - 数字n: 表示对于第n次出现的模式进行替换
  - p表示输出替换后的模式空间
  - w file表示将替换后的模式空间保存到文件file中

sed的正则表达式以及替代字符串都支持标准字符转义序列

```
$ echo -e '1.\tLinux\t90\n2\tPython\t85' |sed 's/\t/\n/g'
#将每行的所有制表符替换为换行符
```

流编辑器sed：sed程序

- 最前面插入：Linux Distributions Report + 一个空行
- 所有的英文字母都变成大写
- 将日期格式12/07/2006变为2006-12-07，日期格式：月和日为1到2位数字，年份为4位数字

```
$ cat distros.txt
SUSE    10.2    12/07/2006
Fedora  10      11/25/2008
$ cat distros.sed
#!/bin/sed -Ef
# sed script to produce Linux distributions report $
1 i\
Linux Distributions Report\

# s/\([0-9]\{2\}\)\/\([0-9]\{2\}\)\/\([0-9]\{4\}\)$/\3-\1-\2/
s%([0-9]{1,2})/([0-9]{1,2})/([0-9]{4})%\3-\1-\2%
y/abcdefghijklmnopqrstuvwxyz/ABCDEFGHIJKLMNOPQRSTUVWXYZ/
$ sed -Ef distros.sed distros.txt
Linux Distributions Report

SUSE    10.2    2006-12-07
FEDORA  10      2008-11-25
```

- sed支持注释，#之后的内容为注释
- sed程序允许执行，并添加相应的shebang行，可以直接执行

awk概述

- awk命令名来源于发明者Aho, Weinberger, Kernighan的首字母
- 一种数据驱动的程序设计语言，支持各种数据处理和计算任务，常用于数据提取和报告
- 把待处理的数据看成一条一条的记录(Record)，记录分隔符RS(Record Seperator)一般为换行符，即每条记录对应着一行。$0可以访问当前记录
- 每条记录进一步基于字段分隔符FS(Field Seperator)分割成多个字段，$1,$2...可以访问当前记录的各个字段
- 类似于sed，pattern-action架构
  - 一直重复，每次基于RS读取一条记录，进行相应的处理，直到没有更多记录为止
  - 当前记录逐个按顺序与pattern匹配，如果匹配，则执行相应的action
  - 一条记录可能会有多个pattern匹配，相应的action都会执行
  - BEGIN是可选的，在读取记录之前执行
  - END是可选的，在所有记录读取完毕之后执行
- AWK程序语言采用类C的设计，但是awk提供了记录和字段的自动分割、内存管理机制，支持字符串和数值类型，不需要声明变量的类型，且支持两种类型间的自动转换
- 只需要几行代码就能够完成简单的数据处理任务

```
BEGIN {action}
pattern {action}
pattern {action}
  .
  .
pattern { action}
END {action}
```

awk使用

```
BEGIN {action}
pattern {action}
pattern {action}
.
pattern { action}
END {action}
```

awk [-F value] [-v var=value] 'program text' [file ...]

awk [-F value] [-v var=value] [-f program-file] [file ...]

- -F value: 设置字段分隔符FS，缺省为空格类字符
- -v var=value 给变量var赋值为value，可以有多个-v选项
- -f program-file awk程序来自于文件，而不是通过第一个参数传递，可以有多个-f选项
- 如果没有文件参数，则处理的数据来自于标准输入。处理的结果输出到标准输出
- awk程序中的pattern和action部分可以省略，当然不能全部省略
  - pattern缺省表示所有记录都匹配
  - action缺省相当于 { print }，即输出当前记录
  - action部分一定要通过括号括起来
  - 不同的pattern {action}之间可没有有分隔符，但建议使用换行符分割
  - 如果action部分比较简单，pattern {action}可在同一行
  - action部分是awk语句，awk语句之间以分号或换行符分割，最后一个awk语句可没有分隔符
  - awk支持注释，#之后的部分表示注释

```
$ cat demo.awk
#!/usr/bin/awk -f
# a simple awk script
{print}
```

```
$ awk '{print}' /etc/hosts
$ awk '{print}
NR==2 {print "line2",$0}' < /etc/hosts
```

```
$ awk '{print}NR==2{print "line2",$0}' /etc/hosts
$ echo '{print}' > tmp.awk
$ awk -f tmp.awk /etc/hosts
```

awk变量和运算符

- awk支持数值和字符串两种基本类型
- 数值类型的常量可以是整数、小数表示法或科学计数法的浮点数
- 字符串常量使用双引号定义
- 变量不需要声明类型，甚至都可以访问一个未赋值的变量，相当于0或""
- 支持赋值语句，也支持复合赋值语句
- 根据所进行的运算，会自动将数值转换为字符串，或者字符串转换为数值

- 支持算术运算符：+ - * / % ^， ^为幂运算
- 字符串拼接运算符：空格
- 支持关系运算符：> < >= <= == !=
- 支持逻辑运算符： &&(逻辑and) ||(逻辑or) !(逻辑not)，也支持?:三元运算
- 没有专门的boolean类型，运算的结果为真时，用1表示，为假时用0表示
- 非0数值表示真，0表示假

```
$ cat a.awk
BEGIN {
    a1 = a2 = 0
    a1 += 4.5
    a2 = 4.50 + "7"   # 11.5
    s1 = "awk" 7 0.1  # "awk70.1"
    s2 = "test" b     # "test"
    print a1, a2, s1, s2
    print 4 * 5, 2 ^ 4, 9 / 2, 8 / 2, 9 % 2
    # 20 16 4.5 4 1
    print (4 > 5), 4 < 5, (4 >= 5), 4 <= 5, 4 == 5, 4 != 5
    print (5 > 4) && s2 == "test" || (0 > 1)   # 1
}
$ awk -f a.awk
```

awk：模式

- 正则表达式匹配运算符 ~和!~：
  - text ~ regexp　text中是否有regexp匹配的内容，1 or 0
  - text !~ regexp　与~运算符相反，是否没有匹配的模式
- awk内置变量
  - NR: 当前记录数
  - NF: 当前记录的字段数
  - $0表示当前记录，　$1,$2...：　当前记录的第1/2个字段
  - $(NF)当前记录的最后一个字段　$(NF-1) 当前记录的倒数第2个字段
- pattern可以是
  - BEGIN或END
  - 表达式，一般是比较关系+逻辑运算符结合在一起
  - 扩展正则表达式：/regexp/，　等价于 $0 ~ "regexp"
  - 范围：expr1, expr2
    - 首先寻找符合expr1的记录
    - 然后检查后续的记录，直到其满足expr2，如果无法找到，则到文件结尾为止
    - 指定范围的每条记录，执行相应的动作

```
$ cat marks.txt
Dept    Name    Marks
ECE     Raj     53
ECE     Joel    72
EEE     Moi     68
CSE     Surya   81
EEE     Tia     59
ECE     Om      92
CSE     Amy     67
$ awk '$1 == "ECE" && $(NF) > 70' marks.txt
ECE     Joel    72
ECE     Om      92
$ awk '/CSE|ECE/' marks.txt
```

```
$ awk 'NR >= 2 && NR <= 4 {print NR
":" $0 }' marks.txt
2:ECE     Raj     53
3:ECE     Joel    72
4:EEE     Moi     68
$ awk '/CSE/, /ECE/' marks.txt
CSE     Surya   81
EEE     Tia     59
ECE     Om      92
CSE     Amy     67
```