sed 和 awk

内容提要

1. 掌握正则表达式的使用
2. 学会使用 sed 对文本文件进行行编辑
3. 学会使用 awk 处理文本

sed

sed 简介

sed [http://www.proxyserve.net/index.php?q=aHR0cDovL3d3dy5tYW4uY3gvc2Vk] 是一个流编辑器（stream editor）。sed 是一个非交互式的行编辑器，它在命令行中输入编辑命令、指定被处理的输入文件，然后在屏幕上查看输出。输入文件可以是指定的文件名，也可以来自一个管道的输出。sed 不改变输入文件的内容，且总是将处理结果输出到标准输出，可以使用输出重定向将 sed 的输出保存到文件中。

与 vi 不同的是 sed 能够过滤来自管道的输入。在 sed 编辑器运行的时候不必人工干涉，所以 sed 常常被称作批编辑器 。此特性允许在脚本中使用编辑命令，极大的方便了重复性编辑任务。当对文件中大量的文本进行替换时， sed 将是一个有利的工具。

sed 以按顺序逐行的方式工作，过程为：

1. 从输入读取一行数据存入临时缓冲区，此缓冲区称为模式空间（pattern space）
2. 按指定的 sed 编辑命令处理缓冲区中的内容
3. 把模式空间的内容送往屏幕并将这行内容从模式空间中删除
4. 读取下面一行。重复上面的过程直到全部处理结束。

sed 命令的格式

sed 命令的格式如下：

```
格式1： sed [OPTION] [-e] command1 [[-e command2] ... [-e commandn]] [input-file]...
格式2： sed [OPTION] -f script-file [input-file]...
```

说明：

- 格式1：执行命令行上的sed编辑命令。可以指定多个编辑命令，每个编辑命令前都要使用 -e 参数，sed 将对这些编辑命令依次进行处理。若只有一个编辑命令时，-e 可以省略。
- 格式2：执行脚本文件中的sed编辑命令。当编辑命令很多时，可将所有的编辑命令存成sed脚本文件，然后在命令行上使用 -f 参数指定这个文件。
- 常用参数：
  - **-n**：sed 在将下一行读入pattern space之前，自动输出pattern space中的内容。此选项可以关闭自动输出，此时是否输出由编辑命令控制。
  - **-r**：使用扩展正则表达式进行模式匹配。
- input-file：sed 编辑的文件列表，若省略，sed 将从标准输入中读取输入，也可以从输入重定向或管道获得输入

sed 的编辑命令包括地址和操作两部分。地址用于指定sed要操作的行；操作指定要进行的处理。

- 通常使用单引号将整个操作命令括起来
- 若操作命令中包含shell变量替换，应该使用双引号将整个操作命令括起来

地址的表示方法列表如下：

分类	表示法	说明
0		省略地址部分，将对输入的每一行进行操作
1	n	表示第 n 行，特殊地： $ 表示最后一行
1	f~s	表示从 f 开始的，步长为 s 的所有行
1	/regexp/	表示与正则表达式匹配的行
2	m,n	表示从第 m 行到第 n 行， 特殊地： m,$ 表示从m行到最后一行
2	m,+n	表示第 m 行以及其后的 n 行
2	/regexp1/,/regexp2/	表示从匹配 regexp1 的行开始到匹配 regexp2 的行
2	/regexp/,n	表示从匹配 regexp 的行开始到第 n 行
2	n,/regexp/	表示从第n行开始到匹配 regexp 的行

另外，在地址部分还可以使用 ！表示反向选择，如 m,n! 表示除了m到n之外的所有行。

sed 支持 25 个操作，下面列出常用的几个，更多的操作的使用方法请参考 sed 手册。

操作	说明
p	打印
l	显示所有字符，包括控制字符(非打印字符)
d	删除
=	显示匹配行的行号
s/regexp/replacement/	将指定行中第一个匹配 regexp 的内容替换为 replacement
s/regexp/replacement/g	将指定行中所有匹配 regexp 的内容替换为 replacement （g 表示全局）
s/regexp/replacement/p	p 打印修改后的行
s/regexp/replacement/gp	p 打印修改后的行 （g 表示全局）
s/regexp/replacement/w fname	将替换后的行内容写到指定的文件 fname 中
s/regexp/replacement/gw fname	将替换后的行内容写到指定的文件 fname 中 （g 表示全局）
r fname	将另外一个文件 fname 中的内容附加到指定行
w fname	将当前模式空间（pattern space）的内容写入指定的文件 fname
n	将指定行的下面一行读入模式空间（pattern space）
q	读取到指定行之后退出 sed
a\	在指定行后面追加文本（主要用于 sed 脚本）
i\	在指定行前面追加文本（主要用于 sed 脚本）
c\	用新文本替换指定的行（主要用于 sed 脚本）

sed 使用举例

1、以 p 操作说明地址的使用方法

```
# 显示 myfile 文件的全部内容
$ sed -n p myfile
# 显示 myfile 文件中第 5 行的内容
$ sed -n 5p myfile
# 显示 myfile 文件中最后一行的内容
$ sed -n '$p' myfile
# 显示 myfile 文件从第 3 行开始步长为5的行的内容
$ sed -n 3~5p myfile
$ sed -n 3~5= myfile
# 显示 myfile 文件从第 3 行开始到第 10 行的内容
$ sed -n 3,10p myfile
# 显示 myfile 文件第 10 行及其后的 10 行内容
$ sed -n 3,+10p myfile
# 显示 myfile 文件从第 3 行开始到最后一行的内容
$ sed -n '3,$p' myfile
# 显示 myfile 文件中所有包含 LANG 的行
$ sed -n /LANG/p myfile
# 显示 myfile 文件中所有不包含 LANG 的行
$ sed -n '/LANG/!p' myfile
# 显示 myfile 文件从第 3 行开始到其后第一次出现 LANG 的行
$ sed -n 3,/LANG/p myfile
# 显示 myfile 文件从第一次出现 LANG 的行开始到最后一行的内容
$ sed -n '/LANG/,$p' myfile
# 显示 myfile 文件从第一次出现以 case 开始的行到第一次出现以 esac 开始的行
$ sed -n /^case/,/^esac/p myfile
```

以上 sed 命令中p 操作的地址使用也适用于其他操作。

2、替换命令使用举例

```
# 在每个输入行中，将第一个出现的 Windows 替换为 Linux
$ sed 's/Windows/Linux/'  myfile
# 在每个输入行中，将第一个出现的 Windows 替换为 Linux ，打印替换结果的行
$ sed -n 's/Windows/Linux/p'  myfile
# 在每个输入行中，将出现的每个 Windows 替换为 Linux
$ sed 's/Windows/Linux/g'  myfile
# 在每个输入行中，将出现的每个 Windows 替换为 Linux ，打印替换结果的行
$ sed -n 's/Windows/Linux/g'  myfile
# 在每个输入行中，将出现的每个 Unix 替换为 Unix/Linux（&表示匹配到的字符串）
$ sed -e 's/Unix/&\/Linux/g' myfile
# 将所有连续出现的c都压缩成单个的c
$ sed 's/cc*/c/g' myfile
# 删除行首的一个空格
$ sed 's/ //' myfile
# 删除每一行前导的连续“空白字符”（空格，制表符）
$ sed 's/^[ \t]*//' myfile
# 删除以句点结尾的行中末尾的句点
$ sed 's/\.$//g' myfile
# 删除每行的第一个字符
$ sed 's/.//' myfile
# 删除每行结尾的所有空格
$ sed 's/ *$//' myfile
# 在文件的每一行开始处插入两个空格
$ sed 's/^/  /' myfile
# 在每一行开头加上一个尖括号和空格（引用信息）
$ sed 's/^/> /' myfile
# 将每一行开头处的尖括号和空格删除（解除引用）
$ sed 's/^> //' myfile
# 删除路径前缀
$ sed 's/.*\///' myfile
```

```
$ ls -d /usr/share/man/man1 |sed 's/.*\///'
# 过滤掉所有标点符号 (.、,、?、!)
$ sed 's/\.//g' -e 's/\,//g' -e 's/\?//g' -e 's/\!//g' myfile
# 对于 GNU sed 可以使用如下的等效形式
$ sed 's/\.//g ; s/\,//g ; s/\?//g ; s/\!//g' myfile
# 不论什么字符，紧跟着s命令的都被认为是分隔符，
# 所以，“#”在这里是分隔符，代替了默认的“/”分隔符。
# 尤其适用于替换文件路径
$ sed 's#/some/path/old#/some/path/new#g' myfile
```

多个sed编辑命令是顺序执行的，例如下面的命令

```
$ sed -e 's/Unix/UNIX/g' -e 's/UNIX System/UNIX Operating System/g' myfile
```

首先将 Unix 替换为 UNIX，然后将 UNIX System 替换为 UNIX Operating System

下面的命令将不会得到预想的结果

```
$ sed -e 's/Unix/UNIX/g' -e 's/Unix System/UNIX Operating System/g' myfile
```

因为Unix在缓冲区中已经被替换成了UNIX，所以再也找不到 Unix System 了。

之所以没有使用下面的命令

```
$ sed -e 's/Unix System/UNIX Operating System/g' myfile
```

而使用了两个替换命令，是为了将 UNIX System 也替换为 UNIX Operating System。在支持扩展正则表达式的 sed 中也可以使用如下的命令

```
$ sed -r 's/(Unix|UNIX) System/UNIX Operating System/g' myfile
```

替换的速度优化：可以考虑在替换命令（“s/…/…/”）前面加上地址表达式来提高速度。举例来说：

```
sed 's/foo/bar/g' filename         # 标准替换命令
sed '/foo/ s/foo/bar/g' filename   # 速度更快
sed '/foo/ s//bar/g' filename      # 简写形式
```

若只替换第一次匹配 foo 的行，可以使用 q 短路后续行的执行。举例来说：

```
sed '/foo/{s/foo/bar/;q}'  filename
```

3、其他命令使用举例

```
# 删除所有空白行
$ sed '/^$/d' myfile
$ sed '/./!d' myfile
# 删除文件顶部的所有空行
$ sed '/./,$!d' myfile
# 从输入的开头一直删除到第1个空行(第一个空行也删除掉)
$ sed '1,/^$/d'  myfile
# 删除所有偶数行，与 sed -n '1~2p' myfile 等效
$ sed 'n;d' myfile
# 删除掉所有包含"GUI"的行
$ sed '/GUI/d' myfile
# 将所有"GUI"都删除掉，并保持剩余部分的完整性
$ sed 's/GUI//g' myfile
# 在每一行后面增加一空行
$ sed G myfile
# 在匹配“regex”的行之后插入一空行
$ sed '/regex/G' myfile
# 将 myfile 中从case开始的行到esac结束的行写到文件 case-block
$ sed '/^case/,/^esac/w case-block' myfile
# 在 myfile 末尾（$）追加新行
# 反斜线 \ 是必需的，它表示将插入一个回车符。在任何要输入回车的地方您必须使用反斜线。
$ sed '$a\
> newline1\
> newline2\
> newline3' myfile
# 在匹配“regex”的行之后追加新行
$ sed '/regex/a\
> newline1\
> newline2\
> newline3' myfile
# i\ 和 c\ 操作的格式与 上面的 a\ 操作的格式相同
```

awk

awk 简介

awk [http://www.proxyserve.net/index.php?q=aHR0cDovL21hbi5jeC9nYXdr] 是一种用于处理文本的编程语言工具。它使用类似于C的语法，并在很多方面类似于 shell 编程语言。awk 名称是由它三个最初设计者的姓氏的第一个字母而命名的：Alfred Aho、Peter Weinberger 和 Brian Kernighan。gawk 是 GNU 版本 awk，gawk 最初在1986年完成，之后不断地被改进、更新。gawk 的当前版本是 3.1.5。gawk 包含 awk 的所有功能。Linux 下的 awk 是 gawk 的符号链接。

与sed和grep很相似，awk 是一种模式扫描和处理语言。但其功能却大大强于sed和grep。awk尤其适合处理结构化的文本，如纯文本的表格等。awk提供了极其强大的功能：它几乎可以完成grep和sed所能完成的全部工作。同时，awk还支持流程控制、数学运算、进程控制语句甚至于内置的变量和函数。它具备了一个完整的语言所应具有的几乎所有精美特性。

与 sed 一样，awk 不会修改输入文件的内容，可以使用输出重定向将 awk 的输出保存到文件中。

awk 命令的格式

awk 命令的格式如下：

```
格式1： awk [OPTION] 'program-statements' [input-file]...
格式2： awk [OPTION] -f program-file [input-file]...
```

说明：

- 格式1：执行命令行上的awk程序语句。若在一行上书写多个awk程序语句时，各个语句使用分号（;）间隔。
- 格式2：执行脚本文件中的awk程序语句。当awk程序语句很多时，可将所有的awk程序语句存成脚本文件，然后在命令行上使用 -f 参数指定这个文件。
- 常用参数：
  - **-F fs** ：在awk中，缺省的字段分隔符一般是空格符或TAB。在- F后面跟着你想用的分隔符即可改变字符分隔符。
  - **-v var=val** ：对变量 var 赋初值为 val，变量既可以是 awk 的内置变量也可以是自定义变量。
- input-file：awk 处理的文件列表，若省略，awk 将从标准输入中读取输入，也可以从输入重定向或管道获得输入。

awk 中每一个语句（statements）都由两部分组成：模式（pattern）和相应的动作（actions）。只要模式匹配，awk 就会执行相应的动作。动作部分由一个或多个命令、函数、表达式组成，之间由换行符或分号隔开，并位于大括号内。

- pattern 和 {actions} 可以省略，但不能同时省略；
- pattern 省略时表示对所有的输入行执行指定的 {actions}；
- {actions} 省略时表示打印匹配行，即 { print } 。
- 模式（pattern）部分可以是：
  - **/regular expression/** ：使用扩展的正则表达式。
  - **relational expression** ：使用关系表达式，可以使用与 C 语言类似的关系运算符。
  - **pattern1, pattern2** ：范围模式，匹配行的范围。表示从匹配pattern1的行到匹配pattern2的行。
  - **BEGIN** ：指定在第一条输入记录被处理之前要执行的动作，通常可在此设置全局变量。
  - **END** ：指定在最后一条输入记录被读取之后要执行的动作，通常可在此输出统计数据。
- 动作（actions）部分可以是：
  - 变量或数组赋值
  - 输入/输出语句
  - 内置函数和自定义函数
  - 流程控制语句

awk 命令的一般形式为：

```
awk 'BEGIN {actions}
     pattern1 {actions}
      ......
     patternN {actions}
     END {actions}'  input-file
```

其中 BEGIN {actions} 和 END {actions} 是可选的。awk 的执行过程如下：

1. 如果存在 BEGIN ，awk 首先执行它指定的 actions。
2. awk 从输入中读取一行，称为一条输入记录。
3. awk 将读入的记录分割成数个字段，并将第一个字段放入变量 $1 中，第二个放入变量 $2 中，以此类推；$0 表示整条记录；字段分隔符可以通过选项 -F 指定，否则使用缺省的分隔符。
4. 把当前输入记录依次与每一个语句中 pattern 比较：如果相匹配，就执行对应的 actions；如果不匹配，就跳过对应的 actions，直到完成所有的语句。
5. 当一条输入记录处理完毕后，awk 读取输入的下一行，重复上面的处理过程，直到所有输入全部处理完毕。
6. 如果输入是文件列表，awk 将按顺序处理列表中的每个文件。
7. awk 处理完所有的输入后，若存在 END，执行相应的 actions。

awk 常用的内置变量

变量	说明
NF	当前记录中的字段数。
NR	当前记录数。
FS	字段分隔符(默认是任何空格)。
RS	记录分隔符(默认是一个换行符)。
OFS	输出字段分隔符(默认值是一个空格)。
ORS	输出记录分隔符(默认值是一个换行符)。
IGNORECASE	如果为真，则进行忽略大小写的匹配。

awk 使用举例

有关 awk 编程的详细内容参见 Gawk: Effective AWK Programming [http://www.proxyserve.net/index.php?q=aHR0cDovL3d3dy5nbnUub3JnL3NvZnR3YXJlL2dhd2svbWFudWFsLw%3D%3D]。下面给出一些使用 awk 的简单例子。

```
# 使用awk打印字符串
$ awk 'BEGIN { print "hello" }'
hello

# 使用awk进行浮点运算
$ awk 'BEGIN { print 1.05e+2/10.5+2.0**3-3.14 }'
14.86

# 显示要处理的输入文件
$ cat test.txt
F115!16201!1174113017250745 10.86.96.41 211.140.16.1 200703180718
F125!16202!1174113327151715 10.86.96.42 211.140.16.2 200703180728
F235!16203!1174113737250745 10.86.96.43 211.140.16.3 200703180738
F245!16204!1174113847250745 10.86.96.44 211.140.16.4 200703180748
F355!16205!1174115827252725 10.86.96.45 211.140.16.5 200703180758

# 显示输入文件的内容
$ awk {print} test.txt
F115!16201!1174113017250745 10.86.96.41 211.140.16.1 200703180718
F125!16202!1174113327151715 10.86.96.42 211.140.16.2 200703180728
F235!16203!1174113737250745 10.86.96.43 211.140.16.3 200703180738
F245!16204!1174113847250745 10.86.96.44 211.140.16.4 200703180748
F355!16205!1174115827252725 10.86.96.45 211.140.16.5 200703180758

# 使用正则表达式匹配行，{actions} 省略时表示 { print }
$ awk '/F[12].*/' test.txt
F115!16201!1174113017250745 10.86.96.41 211.140.16.1 200703180718
F125!16202!1174113327151715 10.86.96.42 211.140.16.2 200703180728
F235!16203!1174113737250745 10.86.96.43 211.140.16.3 200703180738
F245!16204!1174113847250745 10.86.96.44 211.140.16.4 200703180748

# 使用正则表达式匹配行，并打印匹配的第1和第3列（域、字段）
$ awk '/^F[12].*/ {print $1,$3}' test.txt
F115!16201!1174113017250745 211.140.16.1
F125!16202!1174113327151715 211.140.16.2
F235!16203!1174113737250745 211.140.16.3
F245!16204!1174113847250745 211.140.16.4

# 更改字段分隔符为!，执行上面的操作
$ awk -F\! '/^F[12].*/ {print $1,$3}' test.txt
F115 1174113017250745 10.86.96.41 211.140.16.1 200703180718
F125 1174113327151715 10.86.96.42 211.140.16.2 200703180728
F235 1174113737250745 10.86.96.43 211.140.16.3 200703180738
F245 1174113847250745 10.86.96.44 211.140.16.4 200703180748

# 使用空格或!做为字段分隔符（正则表达式 [ !]）
$ awk -F '[ !]' '{print $1,$2,$3,$4,$5,$6}' test.txt
F115 16201 1174113017250745 10.86.96.41 211.140.16.1 200703180718
F125 16202 1174113327151715 10.86.96.42 211.140.16.2 200703180728
F235 16203 1174113737250745 10.86.96.43 211.140.16.3 200703180738
F245 16204 1174113847250745 10.86.96.44 211.140.16.4 200703180748
F355 16205 1174115827252725 10.86.96.45 211.140.16.5 200703180758

# 使用 awk 内置的取子串函数提取输入文件中的手机号
$ awk -F '[ !]' '{print substr($3,6)}' test.txt
13017250745
13327151715
13737250745
13847250745
15827252725

# 使用关系表达式书写模式，打印所有奇数行
$ awk 'NR % 2 == 1'  test.txt
F115!16201!1174113017250745 10.86.96.41 211.140.16.1 200703180718
F235!16203!1174113737250745 10.86.96.43 211.140.16.3 200703180738
F355!16205!1174115827252725 10.86.96.45 211.140.16.5 200703180758

# 使用关系表达式书写模式，打印所有奇数行的第1和第3列（域、字段）
$ awk 'NR % 2 == 1 {print $1,$3}'  test.txt
F115!16201!1174113017250745 211.140.16.1
F235!16203!1174113737250745 211.140.16.3
F355!16205!1174115827252725 211.140.16.5

# 打印输入文件的行数，类似于 wc -l test.txt
$ awk 'END { print NR }' test.txt
5

# 为每一笔记录前添加行号，类似于 cat -n test.txt
$ awk '{print NR,$0}' test.txt
1 F115!16201!1174113017250745 10.86.96.41 211.140.16.1 200703180718
2 F125!16202!1174113327151715 10.86.96.42 211.140.16.2 200703180728
3 F235!16203!1174113737250745 10.86.96.43 211.140.16.3 200703180738
4 F245!16204!1174113847250745 10.86.96.44 211.140.16.4 200703180748
5 F355!16205!1174115827252725 10.86.96.45 211.140.16.5 200703180758

# 为每一笔记录前添加行号，使用制表符作为行号和记录的间隔符
$ awk '{print NR "\t" $0}' test.txt
1       F115!16201!1174113017250745 10.86.96.41 211.140.16.1 200703180718
2       F125!16202!1174113327151715 10.86.96.42 211.140.16.2 200703180728
3       F235!16203!1174113737250745 10.86.96.43 211.140.16.3 200703180738
```

```
4       F245!16204!1174113847250745 10.86.96.44 211.140.16.4 200703180748
5       F355!16205!1174115827252725 10.86.96.45 211.140.16.5 200703180758
```

下面再给出一些 awk 和其他命令结合使用的例子：

```
# 提取文件 test.txt 中的手机号
$ cat test.txt | awk -F\! '{print $3}' | awk '{print $1}'|cut -c6-16
13017250745
13327151715
13737250745
13847250745
15827252725

# 以文件修改顺序生成当前目录下带有时间的文件名
$ ls -alt * --time-style='+%F_%H:%M'| awk '{print $7"--"$6}'
file5--2007-12-27_12:00
file4--2007-12-26_12:00
file3--2007-12-25_12:00
file2--2007-12-24_12:00
file1--2007-12-23_12:00

# 计算当前目录中所有12月份创建的文件的字节数
$ ls -l | awk '$6 == "Dec" { sum += $5 } ; END { print sum }'
79878

$ who
root     tty1         2007-12-14 20:33
osmond   pts/0        2007-12-14 16:26 (192.168.0.77)

# 显示当前所有的登录用户和其使用的终端
$ who | awk '{print $1"\t"$2}'
root    tty1
osmond  pts/0

$ df -hPT -x tmpfs
Filesystem    Type    Size  Used Avail Use% Mounted on
/dev/mapper/VolGroup00-LogVolRoot ext3  3.9G  1.1G  2.7G  28% /
/dev/mapper/VolGroup00-LogVolHome ext3  2.9G  106M  2.6G   4% /home
/dev/sda1     ext3     99M   12M   83M  13% /boot

# 使用 awk 筛选字段并格式化输出
$ df -hPT -x tmpfs| awk '{print "| " $1 " | " $2 " | " $3 " | " $7 " |"}'
| Filesystem | Type | Size | Mounted |
| /dev/mapper/VolGroup00-LogVolRoot | ext3 | 3.9G | / |
| /dev/mapper/VolGroup00-LogVolHome | ext3 | 2.9G | /home |
| /dev/sda1 | ext3 | 99M | /boot |

$ cat /proc/meminfo | grep MemTotal
MemTotal:       515476 kB

$ cat /proc/meminfo | grep MemTotal | awk -F\: '{print $2}'
       515476 kB

$ cat /proc/meminfo | grep MemTotal | awk -F\: '{print $2}' | awk '{print $1 " " $2}'
515476 kB

$ cat /proc/cpuinfo | grep 'model name'
model name      : Intel(R) Core(TM)2 Quad CPU    Q6600  @ 2.40GHz
model name      : Intel(R) Core(TM)2 Quad CPU    Q6600  @ 2.40GHz

$ cat /proc/cpuinfo | grep 'model name' | awk -F\: '{print $2}'
 Intel(R) Core(TM)2 Quad CPU    Q6600  @ 2.40GHz
 Intel(R) Core(TM)2 Quad CPU    Q6600  @ 2.40GHz

$ cat /proc/cpuinfo | grep 'model name' | awk -F\: '{print $2}'|uniq|sed -e 's/ //'
Intel(R) Core(TM)2 Quad CPU    Q6600  @ 2.40GHz

# 显示 ifconfig -a 的输出中以单词开头的行
# ifconfig -a |grep '^\w'
eth0      Link encap:Ethernet  HWaddr 00:0C:29:B3:75:80
lo        Link encap:Local Loopback
sit0      Link encap:IPv6-in-IPv4

# 显示除了 lo 之外的所有网络接口
# ifconfig -a |grep '^\w'|awk '!/lo/{print $1}'
eth0
sit0

# ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 00:0C:29:B3:75:80
          inet addr:192.168.0.101  Bcast:192.168.0.255  Mask:255.255.255.0
          inet6 addr: fe80::20c:29ff:feb3:7580/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:44783 errors:0 dropped:0 overruns:0 frame:0
          TX packets:53687 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:3876638 (3.6 MiB)  TX bytes:16922382 (16.1 MiB)
          Interrupt:169 Base address:0x2000

# 匹配 inet 的行，以分号为字段间隔符打印第2个字段
# ifconfig eth0 |awk -F\: '/inet / {print $2}'
192.168.0.101  Bcast

# ifconfig eth0 |awk -F\: '/inet / {print $2}'|awk '{print $1}'
192.168.0.101
```

```
# ifconfig eth0 | grep 'inet '
          inet addr:192.168.0.101  Bcast:192.168.0.255  Mask:255.255.255.0

# 使用一个或多个空格 或 : 做为字段分隔符（正则表达式 '  +|:'）
# ifconfig eth0 | grep 'inet '| awk -F ' +|:' '{print $4}'
192.168.0.101
# 使用一个或多个空格 或 一个或多个: 做为字段分隔符（正则表达式 '[ :]+'）
# ifconfig eth0 | grep 'inet '| awk -F '[ :]+' '{print $4}'
192.168.0.101

# 删除所有名为 foo 的进程
# kill `ps ax|grep 'foo'|grep -v 'grep'|awk '{print $1}'`

# 查看Apache的并发请求数及其TCP连接状态
# FROM :  http://blog.s135.com/read.php/269.htm
$ netstat -n | awk '/^tcp/ {++S[$NF]} END {for(a in S) print a, S[a]}'
```

参考

- http://www.student.northpark.edu/pemente/sed/index.htm [http://www.proxyserve.net/index.php?q=aHR0cDovL3d3dy5zdHVkZW50Lm5vcnRocGFyay5lZHUvcGVtZW50ZS9zZWQvaW5kZXguaHRt]
- USEFUL ONE-LINE SCRIPTS FOR SED (Unix stream editor) [http://www.proxyserve.net/index.php?q=aHR0cDovL3d3dy5wZW1lbnQub3JnL3NlZC9zZWQxbGluZS50eHQ%3D] -- 中译本
- http://www.ringkee.com/note/opensource/sed.htm [http://www.proxyserve.net/index.php?q=aHR0cDovL3d3dy5yaW5na2VlLmNvbS9ub3RlL29wZW5zb3VyY2Uvc2VkLmh0bQ%3D%3D]
- http://www.ringkee.com/note/opensource/awk.htm [http://www.proxyserve.net/index.php?q=aHR0cDovL3d3dy5yaW5na2VlLmNvbS9ub3RlL29wZW5zb3VyY2UvYXdrLmh0bQ%3D%3D]
- http://www.cbi.pku.edu.cn/chinese/documents/csdoc/awkdoc/awk/nawk_toc.html [http://www.proxyserve.net/index.php?q=aHR0cDovL3d3dy5jYmkucGt1LmVkdS5jbi9jaGluZXNlL2RvY3VtZW50cy9jc2RvYy9hd2tkb2MvYXdrL25hd2tfdG9jLmh0bWw%3D
- http://www.cbi.pku.edu.cn/chinese/documents/csdoc/awkdoc/gawk/gawk_toc.html [http://www.proxyserve.net/index.php?q=aHR0cDovL3d3dy5jYmkucGt1LmVkdS5jbi9jaGluZXNlL2RvY3VtZW50cy9jc2RvYy9hd2tkb2MvZ2F3ay9nYXdrX3RvY5odG1s]
- http://phi.sinica.edu.tw/aspac/reports/96/96005/index.html [http://www.proxyserve.net/index.php?q=aHR0cDovL3BoaS5zaW5pY2EuZWR1LnR3L2FzcGFjL3JlcG9ydHMvOTYvOTYwMDUvaW5kZXguaHRtbA%3D%3D]
- http://plan9.bell-labs.com/cm/cs/awkbook/ [http://www.proxyserve.net/index.php?q=aHR0cDovL3BsYW45LmJlbGwtbGFicy5jb20vY20vY3MvYXdrYm9vay8%3D]
- http://www.student.northpark.edu/pemente/awk.htm [http://www.proxyserve.net/index.php?q=aHR0cDovL3d3dy5zdHVkZW50Lm5vcnRocGFyay5lZHUvcGVtZW50ZS9hd2suaHRt]
- http://www.gnu.org/software/gawk/manual/ [http://www.proxyserve.net/index.php?q=aHR0cDovL3d3dy5nbnUub3JnL3NvZnR3YXJlL2dhd2svbWFudWFsLw%3D%3D]
- sed/awk与unix命令等价代码 [http://www.proxyserve.net/index.php?q=aHR0cDovL3d3dy5saW51eHNpci5vcmcvYmJzL3Nob3d0aHJlYWQucGhwP3RocmVhZGlkPTEwOTgxMA%3D%3D]
- 显示源文件
- 登录