SED 手册

中央研究院计算中心 ASPAC 计划(刘刚 2008年5月7号 整理 ganghust@gmail.com）
aspac@phi.sinica.edu.tw 技术报告: 96005 1996年12月1日 Version:1.0

1.Introduction

Sed(Stream EDitor)为 UNIX 系统上提供将编辑工作自动化的编辑器 ，使用者无需直接编辑数据。使用者可利用 **sed** 所提供 20 多种不同的函数参数 ，组合(批注 [1])它们完成不同的编辑动作。此外 ，由于 **sed** 都以行为单位编辑文件 ，故其亦是行编辑器(line editor)。
sed 是一个非交互式上下文(context)编辑器，它被设计在下列三种情况下发挥作用:

1) 编辑那些对舒适的交互式编辑而言太大的文件。
2) 在编辑命令太复杂而难于在交互模式下键入的时候编辑任何大小的文件。
3) 要在对输入的一趟扫描中有效的进行多个‘全局’(global)编辑函数。

因为每次只把输入的某些行驻留在内存中，并且不使用临时文件，所以可编辑的文件的有效大小，只受限于输入和输出要同时共存于次级存储的要求。

可以单独的建立复杂的编辑脚本并作为给 sed 的命令文件。对于复杂的编辑，这节省了可观的键入和随之而来的错误。从命令文件运行 sed 高效于作者所知道的任何交互式编辑器，甚至包括能用预先写好的脚本驱动的编辑器。

相较于交互式编辑器而言，根本性的损失是缺乏相对地址(由于操作是每次一行的)，和缺乏对命令如期运行的立即验证。

sed 是 UNIX 编辑器 ed 的直系后代。由于在交互式和非交互式操作之间的差异，在 ed 和 sed 之间已经有了可观的变化；甚至 ed 的惯常用户都会经常感到惊讶(并可能气愤)，如果他们没有阅读本文档的章节 2 和 3，就草率的使用 sed 的话。在两个编辑器之间最显著的家族性共同之处，在于他们所识别的模式(‘正则表达式’)的种类；匹配模式的代码可以从 ed 的代码几乎原封不动的复制过来，在章节 2 中对正则表达式的描述就是从 UNIX Programmer’s Manual[1] 几乎原封不动的复制过来的。(代码和描述都是 Dennis M. Ritchie 写的)。
一般 **sed** 最常用在编辑那些需要不断重复某些编辑动作的档上 ，例如将文件中的某个字符串替换成另一个字符串等等。这些相较于一般 UNIX 编辑器(交谈式的, 如 vi、emacs)用手动的方式修改檔 ，**sed** 用起来较省力。下面几节将分别介绍:
何时使用 **sed**

何处获得 **sed**
sed 能做那些编辑动作
sed 如何工作

1.1 何时使用 sed

在修改檔时，如果不断地重复某些编辑动作，则可用 **sed** 自动一次执行这些编辑动作。例如要使 received 文件内 1000 封电子信件内的发信人属名 "Tom" 改成 "John"，此时只要在命令列上执行一简单的 **sed** 命令就可把檔内所有的 "Tom" 字符串替换成 "John"。

再者，当檔需要许多不同编辑动作时，则 **sed** 一次可执行那些不同的编辑动作。例如 **sed** 能一次执行完将檔中所有空白行删除、替换字符串、并将使用者输入的文字添加在文件的第六行等等不同的编辑动作。

1.2 何处获得 sed

一般的 UNIX 系统，本身即附有 **sed**。不同的 UNIX 系统所附的 **sed** 版本亦不尽相同。若读者所使用的 UNIX 系统上未附有 **sed**，则可透过 anonymous ftp 到下列地方去取得：

phi.sinica.edu.tw:/pub/GNU/gnu
gete.sinica.edu.tw:/unix/gnu
ftp.edu.tw:/UNIX/gnu
ftp.csie.nctu.edu.tw:/pub/Unix/GNU
ftp.fcu.edu.tw: /pub3/UNIX/gnu
axp350.ncu.edu.tw:/Packages/gnu
leica.ccu.edu.tw :/pub2/gnu
mail.ncku.edu.tw :/pub/unix/gnu
bbs.ccit.edu.tw :/pub1/UNIX/gnu
prep.ai.mit.edu.tw:/pub/gnu

1.3 sed 能做那些编辑动作

sed 可删除(delete)、改变(change)、添加(append)、插入(insert)、合并、交换文件中的数据行，或读入其它文件的数据到文件中，也可替换(substuite)它们其中的字符串、或转换(tranfer)其中的字母等等。例如将檔中的连续空白行删成一行、"local" 字符串替换成 "remote" 、"t" 字母转换成 "T"、将第 10 行数据与第 11 数据合并等。

1.4 sed 如何工作

如同其它 UNIX 命令，**sed** 由标准输入读入编辑文件并由标准输出送出结果。下图表示 **sed** 将资料行 "Unix" 替换成 "UNIX"，

在图中，上方 standard input 为标准输入，是读取数据之处；standard output 为标准输出，是送出结果之处；中间

sed 方块的下面两个虚线方块表示 **sed** 的工作流程。其中，左边虚线方块表示 **sed** 将标准输入数据置入 pattern space，右边虚线方块表示 **sed** 将 pattern space 中编辑完毕后的数据送到标准输出。
在虚线方块中，两个实线方块分别表示 pattern space 与 **sed** script。其中，pattern space 为一缓冲区，它是 **sed** 工作场所；而 **sed** script 则表示一组执行的编辑指令。
在图中，左边虚线方块 "Unix" 由标准输入置入 pattern space；接着，在右边虚线方块中，**sed** 执行 **sed** script 中的编辑指令 s/Unix/UNIX/ (批注 [2])，结果 "Unix" 被替换成 "UNIX"，之后，"UNIX" 由 pattern space 送到标准输出。
总合上述所言，当 **sed** 由标准输入读入一行数据并放入 pattern space 时，**sed** 依照 **sed** script 的编辑指令逐一对 pattern space 内的数据执行编辑，之后，再由 pattern space 内的结果送到标准输出，接着再将下一行数据读入。如此重复执行上述动作，直至读完所有数据行为止。

2 使用 sed

Sed 命令列可分成编辑指令与文件文件部份。其中，编辑指令负责控制所有的编辑工作；檔檔表示所处理的档案。**sed** 的编辑指令均由位址(address)与函数(function)两部份组成，其中，在执行时，**sed** 利用它的地址参数来决定编辑的对象;而用它的函数参数(批注[3])编辑。
此外，**sed** 编辑指令，除了可在命令列上执行，也可在档案内执行。其中差别只是在命令列上执行时，其前必须加上选项 -e；而在档案(批注[4])内时，则只需在其文件名前加上选项 -f。另外，**sed** 执行编辑指令是依照它们在命令列上或檔内的次序。
下面各节，将介绍执行命令列上的编辑指令、**sed** 编辑指令、执行档案内的编辑指令、执行多个档案的编辑、及执行 **sed** 输出控制。

2.1.执行命令列上的编辑指令

当编辑指令(参照[section 2.2])在命令列上执行时，其前必须加上选项 -e。其命令格式如下：
sed -e '编辑指令1' -e '编辑指令2' ... 文件檔
其中，所有编辑指令都紧接在选项 -e 之后，并置于两个 "'" 特殊字符间。另外，命令上编辑指令的执行是由左而右。
一般编辑指令不多时，使用者通常直接在命令上执行它们。例如，删除 yel.dat 内 1 至 10 行数据，并将其余文字中的 "yellow" 字符串改成 "black" 字符串。此时，可将编辑指令直接在命令上执行，其命令如下：
sed -e '1,10d' -e 's/yellow/black/g' yel.dat
在命令中，编辑指令 '1,10d'(批注[5])执行删除 1 至 10 行数据；编辑指令 's/yellow/black/g'(批注[6])，"yellow" 字符串替换(substuite)成 "black" 字符串。

2.2 sed 的编辑指令

sed 编辑指令的格式如下：
[address1[,address2]]function[argument]
其中，地址参数 address1、address2 为行数或 regular expression 字符串，表示所执行编辑的数据行；函数参数 funct

ion[argument] 为 **sed** 的内定函数，表示执行的编辑动作。
下面两小节，将仔细介绍地址参数的表示法与有哪些函数参数供选择。
2.2.1 地址(address)参数的表示法
实际上，地址参数表示法只是将要编辑的数据行，用它们的行数或其中的字符串来代替表示它们。下面举几个例子说明(指令都以函数参数 d(参照[section4.2]) 为例) :
删除文件内第 10 行数据，则指令为 10d。
删除含有 "man" 字符串的数据行时，则指令为 /man/d。
删除档内第 10 行到第 200 行数据, 则指令为 10,200d。
删除档内第 10 行到含 "man" 字符串的数据行，则指令为 10,/man/d。
接下来，以地址参数的内容与其个数两点，完整说明指令中位址参数的表示法(同样也以函数参数 d 为例)。
地址参数的内容:
地址为十进制数：此数字表示行数。当指令执行时，将对符合此行数的数据执行函数参数指示的编辑动作。例如，删除数据文件中的第 15 行数据，则指令为 15d(参照[section4.2])。其余类推 ,如删除数据文件中的第 m 行数据，则指令为 md 。
地址为 regular expression(参照[附录 A]):
当数据行中有符合 regular expression 所表示的字符串时，则执行函数参数指示的编辑动作。另外，在 regular expression 前后必须加上 "/"。例如指令为 /t.*t/d , 表示删除所有含两 "t" 字母的数据行。其中 ,"." 表示任意字符; "*" 表示其前字符可重复任意次，它们结合 ".*" 表示两 "t" 字母间的任意字符串。
地址参数的个数：在指令中，当没有地址参数时，表示全部数据行执行函数参数所指示的编辑动作; 当只有一地址参数时，表示只有符合地址的数据行才编辑 ; 当有两个地址参数，如 address1,address2 时，表示对数据区执行编辑，address1 代表起始数据行 , address2 代表结束资料行。对于上述内容，以下面例子做具体说明。
例如指令为
d
其表示删除文件内所有数据行。
例如指令为
5d
其表示删除文件内第五行资料。
例如指令为
1,/apple/d
其表示删除资料区，由檔内第一行至内有 "apple" 字符串的数据行。
例如指令为
/apple/,/orange/d
其表示删除资料区，由檔内含有 "apple" 字符串至含有 "orange" 字符串的数据行
2.2.2 有那些函数(function)参数
下页表仲介绍所有 **sed** 的函数参数(参照[chapter 4])的功能。
函数参数 功能
: label 建立 script file 内指令互相参考的位置。
建立批注
{ } 集合有相同位址参数的指令。
! 不执行函数参数。
= 印出资料行数(line number)。

a\ 添加使用者输入的数据。
b label 将执行的指令跳至由 : 建立的参考位置。
c\ 以使用者输入的数据取代数据。
d 删除数据。
D 删除 pattern space 内第一个 newline 字母 \ 前的数据。
g 拷贝数据从 hold space。
G 添加资料从 hold space 至 pattern space 。
h 拷贝数据从 pattern space 至 hold space 。
H 添加资料从 pattern space 至 hold space 。
l 印出 l 资料中的 nonprinting character 用 ASCII 码。
i\ 插入添加使用者输入的数据行。
n 读入下一笔资料。
N 添加下一笔资料到 pattern space。
p 印出资料。
P 印出 pattern space 内第一个 newline 字母 \ 前的数据。
q 跳出 **sed** 编辑。
r 读入它檔内容。
s 替换字符串。
t label 先执行一替换的编辑指令 , 如果替换成牛p>;则将编辑指令跳至 : label 处执行。
w 写资料到它文件内。
x 交换 hold space 与 pattern space 内容。
y 转换(transform)字符。
虽然 , **sed** 只有上表所述几个拥有基本编辑功能的函数 , 但由指令中位址参数和指令与指令间的配合 , 也能使 **sed** 完成大部份的编辑任务。

2.3 执行档案内的编辑指令

当执行的指令太多 , 在命令列上撰写起来十分混乱 , 此时 , 可将这些指令整理储存在档案(譬如檔名为 script_file)内 , 用选项 -f script_file , 则让 **sed** 执行 script_file 内的编辑指令。其命令的格示如下 :

sed -f script_file 文件檔

其中 , 执行 script_file 内编辑指令的顺序是由上而下。例如上一节的例子 , 其可改成如下命令:

sed -f ysb.scr yel.dat

其中 , ysb.scr 檔的内容如下 :

```
1,10d
s/yellow/black/g
```

另外 , 在命令列上可混合使用选项 -e 与 -f , **sed** 执行指令顺序依然是由命令列的左到右, 如执行至 -f 后档案内的指令 , 则由上而下执行。

2.4 执行多个文件檔的编辑

在 **sed** 命令列上，一次可执行编辑多个檔檔，它们跟在编辑指令之后。例如，替换 white.dat、red.dat、black.dat 文件内的 "yellow" 字符串成 "blue"，其命令如下:

sed -e 's/yellow/blue/g' white.dat red.dat black.dat

上述命令执行时，**sed** 依 white.dat、red.dat、black.dat 顺序，执行编辑指令 s/yellow/blue/(请参照[section 4.1]，进行字符串的替换。

2.5.执行输出的控制

在命令列上的选项 -n (批注[7]) 表示输出由编辑指令控制。由前章内容得知，**sed** 会 "自动的" 将数据由 pattern space 输送到标准输出檔。但借着选项 -n，可将 **sed** 这 "自动的" 的动作改成 "被动的" 由它所执行的编辑指令(批注[8])来决定结果是否输出。

由上述可知，选项 -n 必须与编辑指令一起配合，否则无法获得结果。例如，印出 white.dat 文件内含有 "white" 字符串的数据行，其命令如下:

sed -n -e '/white/p' white.dat

上面命令中，选项 -n 与编辑指令 /white/p (参照[section4.6]) 一起配合控制输出。其中，选项 -n 将输出控制权移给编辑指令;/white/p 将数据行中含有 "white" 字符串印出屏幕。

3.范例

一般在实际使用编辑器的过程中，常需要执行替换文件中的字符串、搬移、删除、与搜寻数据行等等动作。当然，一般交谈式编辑器(如 vi、emacs)都能做得到上述功能，但文件一旦有大量上述编辑需求时，则用它们编辑十分没有效率。本章将用举例的方式说明如何用 **sed** 自动执行这些编辑功能。此外，在本章范例中，均以下述方式描述檔的需求:

将文件中...数据，执行...(动作)

如此，目的是为了能将它们迅速的转成编辑指令。其中，"...数据" 部份，转成指令中的位址参数表示;"执行...动作" 部份，则转成函数参数表示。另外，当 "执行...动作" 要由数个函数参数表示时，则可利用 "{ "与 " }" 集合这些函数参数(批注[9])，其指令形式如下:

```
地址参数{
        函数参数1
        函数参数2
        函数参数3
          .
          :
      }
```

上述指令表示，将对符合地址参数的数据，依次执行函数参数1、函数参数2、函数参数3 ... 表示的动作。下面各节，分别举例说明 **sed** 替换数据、移动、删除数据、及搜寻数据的命令。

3.1 替换文件中的数据

Sed 可替换文件中的字符串、数据行、甚至数据区。其中 ，表示替换字符串的指令中的函数参数为 s(参照[section4.1]); 表示替换数据行、或数据区的指令中的函数参数为 c(参照[section4.5])。上述情况以下面三个例子说明。上述情况以下面三个例子说明。

例一. 将文件中含 "machine" 字符串的数据行中的 "phi" 字符串 ，替换成为 "beta" 字符串。其命令列如下 ：

sed -e '/machine/s/phi/beta/g' input.dat(以后文件檔都以 input.dat 代表)

例二. 将文件中第 5 行数据 ，替换成句子 "Those who in quarrels interpose, must often wipe a bloody nose."。其命令列如下

sed -e '5c\
Those must often wipe a bloody nose.
' input.dat

例三. 将文件中 1 至 100 行的资料区 ，替换成如下两行资料 ：

How are you?
data be deleted!

则其命令列如下

sed -e '1,100c\
How are you?\
data be deleted!
' input.dat

3.2 搬动文件中的数据

使用者可用 sed 中的 hold space 暂存编辑中的数据、用函数参数 w(参照[section4.9])将文件数据搬动到它文件内储存、或用函数参数 r(参照[section4.8])将它檔内容搬到文件内。Hold space 是 sed 用来暂存 pattern space 内数据的缓存器 ，当 sed 执行函数参数 h、H(参照[section4.19])时 ，会将 pattern space 资料暂存到 hold space;当执行函数参数 x、g、G(参照[section4.22])时 ，会将暂存的资料取到 pattern space 。下面举三个例子说明。

例一. 将文件中的前 100 数据 ，搬到文件中第 300 后输出。其命令列如下 ：

sed -f mov.scr 文件檔

mov.scr 檔的内容为

```
1,100{
H
d
}
300G
```

其中 ，

```
1,100{
H
d
}
```

它表示将文件中的前 100 数据 ，先储存(参照[section4.19])在 hold space 之后删除 ;指令 300G (参照[section4.22]) 表示 ，将 hold space 内的资料 ，添加在文件中的第 300 数据后输出。

例二. 将文件中含 "phi" 字符串的数据行 , 搬至 mach.inf 檔中储存。其命令列如下 :
sed -e '/phi/w mach.inf' 文件檔
例三. 将 mach.inf 檔内容 , 搬至文件中含 "beta" 字符串的数据行。其命令列如下 :
sed -e '/beta/r mach.inf' 文件檔
另外 , 由于 **sed** 是一 stream(参照[section1.4])编辑器 , 故理论上输出后的文件数据不可能再搬回来编辑。

3.3 删除文件中的数据

因为 **sed** 是一行编辑器 , 所以 **sed** 很容易删除个别数据行或整个数据区。一般用函数参数 d(参照[section4.2])或 D(参照[section4.17]) 来表示。下面举两个例子说明。
将檔内所有空白行全部删除。其命令列为

sed -e '/^$/d' 文件檔

regular expression(批注[附录 A]) , ^$ 表示空白行。 其中 , ^ 限制其后字符串必须在行首; $ 限制其前字符串必须在行尾。
将文件内连续的空白行 , 删除它们成为一行。其命令列为

```
sed -e '/^$/{
N
/^$/D
}' 文件檔
```

其中 , 函数参数 N(参照[section4.16])表示 , 将空白行的下一行资料添加至 pattern space 内。函数参数 /^$/D 表示 , 当添加的是空白行时 , 删除第一行空白行 , 而且剩下的空白行则再重新执行指令一次。指令重新执行一次 , 删除一行空白行 , 如此反复直至空白行后添加的为非空白行为止 , 故连续的空白行最后只剩一空白行被输出。

3.4 搜寻文件中的数据

Sed 可以执行类似 UNIX 命令 grep 的功能。理论上 , 可用 regular expression(参照[附录 A])。例如 , 将檔中含有 "gamma" 字符串的数据行输出。则其命令列如下:

sed -n -e '/gamma/p' 文件檔

但是 , **sed** 是行编辑器 , 它的搜寻基本上是以一行为单位。因此 , 当一些字符串因换行而被拆成两部份时 , 一般的方法即不可行。此时 , 就必须以合并两行的方式来搜寻这些数据。其情况如下面例子:
例. 将文件中含 "omega" 字符串的数据输出。其命令列如下
sed -f gp.scr 文件檔
gp.scr 檔的内容如下 :

```
/omega/b
N
h
s/.*\n//
/omega/b
g
D
```

在上述 **sed** script(批注[10]), 因借着函数参数 b 形成类似 C 语言中的 case statement 结构 , 使得 **sed** 可分别处理当数据内含 "omega" 字符串 ; 当 "omega" 字符串被拆成两行 ; 以及数据内没有"omega" 字符串的情况。接下来就依上述的三种情况 , 将 **sed** script 分成下面三部份来讨论。
当数据内含 "omega", 则执行编辑指令
/omega/b
它表示当资料内含 "omega" 字符串时 , **sed** 不用再对它执行后面的指令 , 而直接将它输出。
当数据内没有"omega", 则执行编辑指令如下
N
h
s/.*\n//
/omega/b
其中 , 函数参数 N(参照[section 4.16]), 它表示将下一行资料读入使得 pattern space 内含前后两行数据 。函数参数 h(参照[section 4.19]), 它表示将 pattern space 内的前后两行资料存入 hold space 。函数参数 s/.*\n//, 它表示将 pattern space 内的前后两行资料合并(批注[11])成一行。/omega/b, 它表示如果合并后的数据内含 "omega" 字符串 , 则不用再执行它之后的指令 , 而将此数据自动输出 ;
当合并后的数据依旧不含 "omega", 则执行编辑指令如下
g
D
其中 , 函数参数 g(参照[section4.21]), 它表示将 hold space 内合并前的两行资料放回 pattern space。 函数参数 D(参照[section4.17]), 它表示删除两行资料中的第一行资料 , 并让剩下的那行数据 , 重新执行 **sed** script。如此 , 无论的资料行内或行间的字符串才可搜寻完全。

4 介绍函数参数

本章将以一节一个函数参数的方式 ,介绍所有 **sed** 提供的函数参数 , 其中有
| s | d | a | i | c | p | l | r | w | y | ! | n | q | = | # | N | D | P | h | H | g | G | x | b | t |
另外 , 在各节中 , 首先简单介绍函数参数功能 , 接着说明函数参数与地址参数配合的格式 , 而其中也一并描述 **sed** 执行此函数参数的工作情形。

4.1 s

函数参数 s 表示替换(substitute)文件内字符串。其指令格式如下 :
[address1[,address2]] s/pattern/replacemen/[flag]
对上述格式有下面几点说明 :
函数参数 s 最多与两个地址参数配合。
关于 "s/pattern/replacement/[flag]"(批注[12]) 有下面几点说明:
pattern : 它为 reguler expression 字符串。它表示文件中要被替换的字符串。
replacement : 它为一般字符串。但其内出现下列字符有特别意义 :
& : 代表其前 pattern 字符串。例如
sed -e 's/test/& my car/' 资料文件名

指令中 ,& 代表 pattern 字符串 "test"。故执行后 , 数据文件的 "test" 被替换成 "test my car"。
\n : 代表 pattern 中被第 n 个 \(、\)(参照[附录 A]) 所括起来的字符串。例如
sed -e 's/\(test\) \(my\) \(car\)/[\2 \3 \1]/' 资料文件名
指令中 ,\1 表示 "test"、\2 表示 "my"、\1 表示 "car" 字符串。故执行后 , 数据文件的 "test my car" 被替换成 "[my car test]"。
\ : 可用它来还原一些特殊符号(如上述的 & 与 \ )本身字面上的意义 , 或用它来代表换行。
flag : 主要用它来控制一些替换情况 :
当 flag 为 g 时 , 代表替换所有符合(match)的字符串 。
当 flag 为十进制数 m 时 , 代表替换行内第 m 个符合的字符串。
当 flag 为 p 时 , 代表替换第一个符合 pattern 的字符串后 , 将数据输出标准输出文件。
当 flag 为 w wfile 时 , 代表替换第一个符合 pattern 的字符串后 , 输出到 wfile 檔内(如果 wfile 不存在 , 则会重新开启名为 wfile 的档案)。
当没有 flag 时 , 则将资料行内第一个符合 pattern 的字符串以 replacement 字符串来替换 。
delimiter : 在 "/pattern/replace/[flag] " 中 "/" 被当成一 delimiter。除了空白(blank)、换行(newline) 之外 , 使用者可用任何字符作为 delimiter。例如下述编辑指令

```
s#/usr#/usr1#g
```

上述命令中 \verb|#| 为 delimiter。如果用 "/" 做 delimiter , 则 **sed** 会将 pattern 与 replacement 中的 "/" 当成 delimiter 而发生错误。
范例:
题目 : 替换 input.dat 檔(后面如果没有特别指定 , 均假设檔檔名为 input.dat)内 "1996" 字符串成 "1997", 同时将这些数据行存入 year97.dat 檔内。
说明 : 用函数参数 s 指示 **sed** 将 "1996" 字符串替换成 "1997", 另外用 s argument 中的 flag w 指示 **sed** 将替换过的资料行存入 year97.dat 檔内。
sed 命令列:
sed -e 's/1996/1997/w year97.dat' input.dat

4.2 d

函数参数 d 表示删除数据行 , 其指令格式如下:

```
[address1[ ,address2]] d
```

对上述格式有下面几点说明:
函数参数 d 最多与两个地址参数配合。
sed 执行删除动作情况如下 :
将 pattern space 内符合地址参数的数据删除。
将下一笔资料读进 pattern space 。
重新执行 **sed** script。
范例 : 可参考 section 3.3。

4.3 a

函数参数 a 表示将资料添加到文件中。其指令格式如下:

[address1] a\ 使用者所输入的数据

对上述格式有下面几点说明:

函数参数 a 最多与一个地址参数配合。

函数参数 a 紧接着 "\" 字符用来表示此行结束 , 使用者所输入的数据必须从下一行输入。如果数据超过一行 , 则须在每行的结尾加入"\"。

sed 执行添加动作情况如下 : 当 pattern space 内数据输出后 , **sed** 跟着输出使用者所输入的数据。

范例 :

题目: 添加 "多任务操作系统" 在含 "UNIX" 字符串的数据行后。假设 input.dat 檔的内容如下 :

UNIX

说明: 用函数参数 a 将所输入的数据添加在含 "UNIX" 字符串的数据行后。

sed 命令列如下 :

sed -e '/UNIX/a\

多任务操作系统

' input.dat

执行上述命令后 , 其输出结果如下 :

UNIX

多任务操作系统

4.4 i

函数参数 i 表示将资料插入文件中。其指令格式如下:

[address1] i\ 使用者所输入的数据

对上述格式有下面几点说明:

函数参数 i 最多与一个地址参数配合。

函数参数 i 紧接着 "\" 字符用来表示此行结束 , 使用者所输入的数据必须从下一行输入。如果数据超过一行 , 则须在每行的结尾加入"\"。

sed 执行插入动作的情况如下 : 在 pattern space 内数据输出前 , **sed** 先输出使用者所输入的数据。

范例 :

题目: 将 "文章版权属于中央研究院" 插在 input.dat 檔中含 "院长 : 李远哲" 的数据行之前。假设 input.dat 檔内容如下 :

院长 : 李远哲

说明: 用函数参数 i 将数据行 "文章版权属于中央研究院" 插在含 "院长 : 李远哲" 的数据行之前。

sed 命令列如下:

sed -e '/院长 : 李远哲/i\

文章版权属于中央研究院

' input.dat

执行上述命令后的输出如下 :

文章版权属于中央研究院

院长 : 李远哲

4.5 c

函数参数 c 表示改变文件中的数据。其格式如下:

[address1[,address2]]c\　　使用者所输入的数据

对上述格式有下面几点说明:

函数参数 c 最多与两个地址参数配合。

函数参数 c 紧接着 "\" 字符用来表示此行结束，使用者所输入的数据必须从下一行输入。如果数据超过一行，则须在每行的结尾加入"\"。

sed 执行改变动作的情况：在 pattern space 内数据输出时，**sed** 改变它成为使用者所输入的数据。

范例：参考 section 3.1 之例二、三。

4.6 p

函数参数 p 表示印出资料。其指令格式如下：

[address1[, address2]] p

对于上述格式有下面几点说明：

函数参数 p 最多与两个地址参数配合。

sed 执行印出动作的情况如下：**sed** 拷备一份 pattern space 内容至标准输出檔。

范例：参考 section 3.4 开头的内容。

4.7 l

函数参数 l，除可将资料中的 nonprinting character 以 ASCII码列出外，其于均与函数参数 p 相同。例如，将下面 input.dat 檔中的 ^[以 ASCII 码印出

The Great ^[is a movie starring Steve McQueen.

执行命令 **sed** -e 'l' input.dat 后，则输出结果如下：

The Great \003 is a movie starring Steve McQueen.

The Great　　is a movie starring Steve McQueen.

上述第二行数据为 **sed** 的自动输出(请参照批注[]。

4.8 r

函数参数 r 表示读入它档案内容到檔中。其指令格式如下：

[address1] r 它檔名称

对于上述格式有下面几点说明：

函数参数 r 最多与一个地址参数配合。

在指令中，函数参数 r 与它檔名称间，只能有一空格。

sed 执行读入动作的情况如下 ： 在 pattern space 内数据输出后 , **sed** 读出它檔的内容跟着输出。当它檔不存在时 , **sed** 照样执行其它指令而不会有任何错误讯息产生。
范例 ： 参考 section 3.1 之例三。

4.9 w

函数参数 w 表示将檔中的写到它檔内。其指令格式如下 ：
[address1[,address2]] w 它檔名称
对于上述格式有下面几点说明 ：
函数参数 w 最多与两个地址参数配合。
在指令中 , 函数参数 w 与它檔名称间 , 只能有一空格。
sed 执行写出动作的情况如 ： 将 pattern space 内资料写到它文件内。数据写入时 , 会取代(overwrite)原来档案内的数据。另外 , 当它檔不存在时 , **sed** 会重新产生(creat)它。
范例:参考 section 3.1 之例二。

4.10 y

函数参数 y 表示转换数据中的字符。其指令格式如下 ：
[address1[,address2]]y /xyz.../abc.../
对于上述格式有下面几点说明 ：
函数参数最多配合两个地址参数。
指令中 , /abc.../xyz.../(x、y、z、a、b、c 代表某些字符) 为 y 的 argument 。其中 abc... 与 xyz... 的字符个数必须相同。
sed 执行转换时 , 将 pattern space 内数据内的 a 字符转换成 x 字符 、b 字符转换成 y 字符 、c 字符转换成 z 字符 ...。
范例:
题目: 将 input.dat 文件中的小写字母改成大写。假设 input.dat 檔的内容如下 ：

```
Sodd's Second Law:
        Sooner or later, the worst possible set of
        circumstances is bound to occur.
```

说明:利用函数参数 y 指示 **sed** 做字母大小的转换。
sed 命令列如下 ：

```
sed -e '
y/abcdefghijklmnopqrstuvwxyz/ABCDEFGHIJKLMNOPQRSTUVWXYZ/
' input.dat
```

执行上述命令输出结果如下 ：

```
SODD'S SECOND LAW:
        SOONER OR LATER, THE WORST POSSIBLE SET OF
        CIRCUMSTANCES IS BOUND TO OCCUR.
```

4.11 !

函数参数 ！表示不执行函数参数。当有如下指令时 ，

[address1[, address2]] ！ 函数参数

表示 ，对符合地址参数之数据不执行函数参数。例如删除 ，除了含 "1996" 字符串 ，所有数据行 ，则执行如下命令

sed -e '/1996/!d' input.dat

4.12 n

函数参数 n 表示读入下一行资料。其指令格式如下:

[address1[,address2]] n

对上述格式有下面几点说明 :

函数参数 n 最多配合两个地址参数。

sed 执行读入下一行动作的情况如下 :

输出在 pattern space 的数据。

将下一笔资料读到 pattern space。

执行下一个编辑指令。

范例(可与[section4.18]中的范例比较):

题目 : 输出 input.dat 文件内偶数行资料。假设 input.dat 檔内容如下:

The

UNIX

Operation

System

说明: 在命令列上

以选项 -n , 将数据输出的控制权(参照[section2.5])转给指令。

利用函数参数 n 将下一行数据(偶数行)取代 pattern space 内的资料行(奇数行)。

利用函数参数 p 将 pattern space 内的数据(偶数行)输出。

最后 ，整个输出只有原先文件内的偶数行数据。

sed 命令列如下 :

sed -n -e 'n' -e 'p' infro.dat

执行上述命令后 ，输出的结果如下 :

UNIX

System

4.13 q

函数参数 q 表示跳离 **sed** 。其指令格式如下:

[address1] q

对上述格式有下面几点说明 ：
函数参数 q 最多配合一个地址参数。
sed 执行跳离动作时 ，它停止输入 pattern space 数据 ，同时停止数据送到标准输出文件。
范例 ：
题目: 对文件文件执行 script_file 内的编辑指令 ，除非遇到 "Linux" 字符串。
说明: 无论 script_file 内是何种指令 ，使用者只要在命令列上用指令/Linux/q，函数参数 q 会强迫 **sed** 遇到 "Linux" 时做跳离动作。
sed 命令列如下 ：
sed -e '/Linux/q' -f script_file input.dat

4.14 =

函数参数 = 表示印出资料的行数。其指令格式如下:
[address1 ,[address2]] =
对上述格式有下面几点说明 ：
函数参数 = 最多配合两个地址参数。
执行时 ，行数将在数据输出前先输出。
范例 ：
题目: 印出 input.dat 文件内资料行数。假设 input.dat 的内容如下 ：
The UNIX
Operating System
说明 ：用函数参数 = 来印出资料的行数。
sed 命令列如下 ：
sed -e '=' input.dat
执行上述命令后 ，输出的结果如下 ：
1
The UNIX
2
Operating System

4.15

在 script file 内 ，函数参数 # 后的文字为注解。当注解文字超过多行时 ，其行间须以 "\" 换行字符相隔。

4.16 N

函数参数 N 表示添加下一笔资料在 pattern space 内。其指令格式如下:
[address1 ,[address2]] N
对上述格式有下面几点说明 ：

函数参数 N 最多配合两个地址参数。
sed 执行时 ，将下一行数据读入并添加在 pattern space 内 ，数据行间以换行字符(embedded newline character)分隔。此外 ，在替换时 ，换行字符可用 \n 来 match。
范例 ：
题目: 将下述两行数据合并。假设 input.dat 的内容如下 ：
The UNIX
Operating System
说明 ：先利用函数参数 N 将两行数据置于 pattern space 内 ，在利用函数参数 s/\n/ / 将两行数据间的分隔号 \n 以空白替代 ，如此两行数据变成一行输出。
sed 命令列如下 ：
sed -e 'N' -e 's/\n/ /' input.dat
执行上述命令后 ，其输出的结果如下:
The UNIX Operating System

4.17 D

函数参数 D 表示删除 pattern space 内的第一行资料。其指令格式如下:
[address1,address2]D
对上述格式有下面几点说明 ：
函数参数 D 最多配合两个地址参数。
函数参数 D 与 d 的比较如下 ：
当 pattern space 内只有一数据行时 , D 与 d 作用相同。
当 pattern space 内有多行资料行时
D 表示只删除 pattern space 内第一行资料 ; d 则全删除。
D 表示执行删除后 , pattern space 内不添加下一笔数据 ，而将剩下的数据重新执行 **sed** script ; d 则读入下一行后执行 **sed** script。
范例 ：参考 section 3.3 的第二个例子。

4.18 P

函数参数 P 表示印出 pattern space 内的第一行资料。其指令格式如下:
[address1,address2] P
对上述格式有下面几点说明 ：
函数参数 P 最多配合两个地址参数。
P 与 p , 除了面对的 pattern space 内的数据行数不同外 ，其它均相同。
范例(可与[section4.12]中的范例):
题目 ：输出 input.dat 文件内奇数行资料。假设 input.dat 檔内容如下:
The
UNIX
System

说明: 在命令列上
以选项 -n , 将数据输出的控制权(参照[section2.5])转给指令。
利用函数参数 N 将偶数行添加至 pattern space 内奇数行后。
利用函数参数 P 将 pattern space 内的第一行(奇数行)输出。
在奇数行输出后 , pattern space 内剩下的数据行(偶数行)则被放弃输出。最后 , 整个输出只有原先的奇数行数据。
sed 命令列 :
sed -n -e 'N' -e 'P' infro.dat
执行上述命令后 , 输出的结果如下 :
The
System

4.19 h

函数参数 h 表示暂存 pattern space 的资料至 hold space。其指令格式如下:
[address1 ,[address2]] h
对上述格式有下面几点说明 :
函数参数 h 最多配合两个地址参数。
sed 执行暂存动作时 , 会盖掉(overwrite) hold space 内原来的数据。
当 **sed** 全部执行结束时 , hold space 内数据会自动清除。
范例 :参考 section 3.4 的例子。

4.20 H

函数参数 H 与 h 唯一差别是 , **sed** 执行 h 时 , 数据盖掉(overwrite) hold space 内原来的数据 , 而 H , 数据则是 "添加(append)" 在 hold space 原来数据后。例题请参考 section 3.2 之例一。

4.21 g

函数参数 g 表示与函数参数 h 相反的动作 , 它表示将 hold space 内资料放回 pattern space 内。其指令格式如下 :
[address1,address2]g
函数参数 g 最多配合两个地址参数。
sed 执行放回动作时 , 数据盖掉(overwrite)(批注[13]) pattern space 内原来的数据。
例题 :参考 section 3.4 的例子。

4.22 G

函数参数 G 与 g 唯一差别是 , **sed** 执行 g 时 , 数据盖掉(overwrite) pattern space 内原来的数据 , 而 G , 数据则是 "添加(append)" 在 pattern space 原来数据后。例子请参考 section 3.2 例一。

4.23 x

函数参数 x 表示交换 hold space 与 pattern space 内的数据。其指令格式如下 :

[address1 ,[address2]] x

函数参数 x 大部份与其它处理 hold space 的函数参数一起配合。例如 , 将 input.dat 文件内第 1 行资料取代第 3 行资料。此时 , 用函数参数 h 与 x 来配合。其中 , 以函数参数 h 将第 1 资料存入 hold space ; 当第 3 行数据出现在 pattern space , 以函数参数 x 交换 hold space 与 pattern space 的内容。如此 , 第 3 行资料就被第 1 资料替代。其命令列如下:

sed -e '1h' -e '3x' input.dat

4.24 b、:label

函数参数 : 与函数参数 b 可在 **sed** script 内建立类似 BASIC 语言中 GOTO 指令的功能。其中 , 函数参数 : 建立标记;函数参数 b 将下一个执行的指令 branch 到标记处执行。函数参数 : 与 b , 在 script file 内配合的情况如下

.
.
.
编辑指令m1
:记号
编辑指令m2
.
.
.
[address1,[address2]]b [记号]

其中 , 当 **sed** 执行至指令 [address1,[address2]]b [记号] 时 , 如 pattern space 内的数据符合地址参数 , 则 **sed** 将下一个执行的位置 branch 至由 :记号(批注[14])设定的标记处 , 也就是再由 "编辑指令m2" ... 执行。另外 , 如果指令中函数参数 b 后没有记号 , 则 **sed** 将下一个执行的指令 branch 到 script file 的最后 , 利用此可使 **sed** script 内有类似 C 语言中的 case statement 结构。

范例 :

题目: 将 input.dat 文件内数据行的开头字母重复印 40 次。假设 input.dat 檔的内容如下 :

A
B
C

说明: 用指令 b p1 与 :p1 构成执行增加字母的循环(loop) , 同时在字母出现 40 个时 , 也用指令 b 来跳出循环。下面就以文件内第一行数据 "A" 为例 , 描述它如何连续多添加 39 个 "A" 在同一行:

用指令 s/A/AA/(参照 section4.1)将 "A" 替换成 "AA"。

用指令 b p1 与 :p1 构成循环(loop) , 它目的使上述动作被反复的执行。每执行一次循环 , 则数据行上的 "A" 就多出一个。例如 , 第一次循环数据行变成 "AA" , 第二次循环资料行变成 "AAA" ...。

用指令 [ABC]\{40\}/b(批注[15]) 来作为停止循环的条件。当数据行有连续 40 个 A 出现时 ，函数参数 b 将执行的指令跳到最后 ，停止对此行的编辑。
同样 ，对其他数据行也如同上述的方式执行。
sed 命令列如下 ：

```
sed -e '{
:p1
/A/s/A/AA/
/B/s/B/BB/
/C/s/C/CC/
/[ABC]\{40\}/b
b p1
}' input.dat
```

4.25 t

基本上 ，函数参数 t 与 函数参数 b 的功能类似 ，除了在执行 t 的 branch 前 ，会先去测试其前的替换指令有没有执行替换成功外。在 script file 内的情况如下:

```
        .
        .
        .
    编辑指令m1
    :记号
    编辑指令m2
        .
        .
        .
    s/.../.../
    [address1,[address2]]t [记号]
    编辑指令m3
```

其中 ，与函数参数 b 不同处在于 ，执行函数参数 t branch 时 ，会先检查其前一个替换指令成功与否。如成功 ，则执行 branch ; 不成功 ，则不 branch , 而继续执行下一个编辑指令 ，例如上面的编辑指令m3。
范例:
题目 ：将 input.dat 文件中资料 A1 替换成 C1、C1 替换成 B1、B1 替换成 A1。input.dat 檔的内容如下:

```
代号
B1
A1
B1
C1
A1
C1
```

说明 : input.dat 文件中全部数据行只需要执行一次替换动作 , 但为避免数据被替换多次 , 所以利用函数参数 t 在 **sed** script 内形成一类似 C 语言中 case statement 结构 , 使每行数据替换一次后能立即用函数参数 t 跳离替换编辑。

sed 命令列 :

```
sed -e '{
s/A1/C1/
t
s/C1/B1/
t
s/B1/A1/
t
}' input.dat
```

常用的 regular expression

常用的 regular expression

普通字符 由普通字符所组成的 regular expression 其意义与原字符串字面意义相同。

^字符串 限制字符串必须出现于行首 。

$字符串 限制字符串必须出现行尾。

. 表示任意一字符。

[...] 字符集合, 用以表示两中括号间所有字符当中的任一个 ,如 [^...]表示两中括号间所有字符以外的字符。

-& 字符集合中可用"&"指定字符的范围。

* 用以形容其前的字符(或字符集合)可重复任意多次 。

\n 表示嵌入新行字符(imbedded new line character)。

\(...\) 于 regular expression 中使用"\(" "\)"来括住一部份的 regular expression ; 其后可用"\1"来表示第一个被"\(" "\)"括住的部份。若 regular expression 中使用数次的"\(" "\)"来括住不同的部份 , 则依次使用"\1","\2","\3",...(最多可到"\9")。

另外 , 在不同平台上 , regular expression 会有一些不同的限制 , 详细情况参照 appendix B。

HP-UX Release 9.01 与 SunOS 5.4 内 **sed** 对 regular expression 中各种特殊字符的接受能力

regular expression 的特殊字符	HP-UX Release 9.01	SunOS 5.4
.	接受	接受
*	接受	接受
^	接受	接受
$	接受	接受
\	接受	接受
[ ]	接受	接受
\(\)	与 \1 ... \9 合用	接受
\{重复次数\}	接受	接受
\{下限,上限\}	接受	接受
\下限,\}	接受	接受
\;	不接受	接受
+	不接受	不接受
?	不接受	不接受
|	不接受	不接受

( ) 不接受 不接受
& 接受 接受

批注

批注一.
就是后面将会提到的 **sed** script。
批注二.
指令 s/Unix/UNIX/ 表示将 "Unix" 替换成 "UNIX"。请参照 section 4.1。
批注三.
在指令中有 20 几个函数参数可供选择。
批注四.
以后这档案称作 script file。
批注五.
编辑指令 1,10d 中 , 地址参数为 1,10 , 故 1 至 10 行的数据执行函数参数 d 所指定的删除动作。
批注六.
编辑指令 s/yellow/black/g 中 , 由于没有地址参数 , 故所有的数据行都要执行函数参数 s/yellow/black/g 所指定替换动作。在函数参数 s/yellow/black/g 中 , /yellow/black/g 为 s 的 argument , 其表示替换资料行中所有的 "yellow" 成 "black"。
批注七.
其命令格式如下 :
sed -n [-e 编辑指令].. [-f script_file].. [文件檔..]
批注八.
这些编辑指令中的函数参数可能是 p、l、s 的其中之一。
批注九.
在有些情况下 , 也可用编辑指令代替函数参数。例如 section3.3 之例二。
批注十.
这里 , **sed** script 是指 gp.scr 檔的内容。它表示这一次 **sed** 执行的编辑指令。
批注十一.
此函数参数 , 表示替换掉(除掉) pattern space 内两行间的换行记号。 故 pattern space 内只有一行资料。
批注十二.
/pattern/replacement/[flag] 为函数参数 s 的 argument。
批注十三.
注意此时 , 虽然资料是放回 pattern space , 但 hold space 的内容还是不变。
批注十四.
注意 ":" 与记号间不可有空格。
批注十五.
地址参数 [ABC]\{40\} , 表示 40 个 A 字母或 40 个 B 字母或 40 个 C 字母。其中 [ABC] 表示 "A" 或 "B" 或 "C"; 其后的 \{40\} 表示其前的字母有 40 个。regular expression 请参照附录 A

附录：SED单行脚本快速参考（Unix 流编辑器）

sed(stream editor)

功能说明：利用script来处理文本文件。

语　　法：sed [-hnV][-e<script>][-f<script文件>][文本文件]

补充说明：sed可依照script的指令，来处理、编辑文本文件。

参　　数：

-e<script>或--expression=<script> 以选项中指定的script来处理输入的文本文件。
-f<script文件>或--file=<script文件> 以选项中指定的script文件来处理输入的文本文件。
-h或--help 显示帮助。
-n或--quiet或--silent 仅显示script处理后的结果。
-V或--version 显示版本信息。

文本间隔：

```
--------
  # 在每一行后面增加一空行
 sed G
  # 将原来的所有空行删除并在每一行后面增加一空行。
 # 这样在输出的文本中每一行后面将有且只有一空行。
 sed '/^$/d;G'
  # 在每一行后面增加两行空行
 sed 'G;G'
  # 将第一个脚本所产生的所有空行删除（即删除所有偶数行）
 sed 'n;d'
  # 在匹配式样“regex”的行之前插入一空行
 sed '/regex/{x;p;x;}'
```

```
 # 在匹配式样“regex”的行之后插入一空行
sed '/regex/G'
 # 在匹配式样“regex”的行之前和之后各插入一空行
sed '/regex/{x;p;x;G;}'
```

编号：

```
# 为文件中的每一行进行编号（简单的左对齐方式）。这里使用了“制表符”
# （tab，见本文末尾关于'\t'的用法的描述）而不是空格来对齐边缘。
sed = filename | sed 'N;s/\n/\t/'

# 对文件中的所有行编号（行号在左，文字右端对齐）。
sed = filename | sed 'N; s/^/         /; s/ *\(.\{6,\}\)\n/\1    /'

# 对文件中的所有行编号，但只显示非空白行的行号。
sed '/./=' filename | sed '/./N; s/\n/ /'

# 计算行数 （模拟 "wc -l"）
sed -n '$='
```

文本转换和替代：

```
 # Unix环境：转换DOS的新行符（CR/LF）为Unix格式。
sed 's/.$//'                               # 假设所有行以CR/LF结束
sed 's/^M$//'                              # 在bash/tcsh中，将按Ctrl-M改为按Ctrl-V
sed 's/\x0D$//'                            # ssed、gsed 3.02.80，及更高版本

# Unix环境：转换Unix的新行符（LF）为DOS格式。
sed "s/$/`echo -e \\\r`/"                  # 在ksh下所使用的命令
sed 's/$'"/`echo \\\r`/"                   # 在bash下所使用的命令
sed "s/$/`echo \\\r`/"                     # 在zsh下所使用的命令
sed 's/$/\r/'                              # gsed 3.02.80 及更高版本
```

```
# DOS环境：转换Unix新行符（LF）为DOS格式。
sed "s/$//"                                  # 方法 1
sed -n p                                     # 方法 2
 # DOS环境：转换DOS新行符（CR/LF）为Unix格式。
# 下面的脚本只对UnxUtils sed 4.0.7 及更高版本有效。要识别UnxUtils版本的
#  sed可以通过其特有的“--text”选项。你可以使用帮助选项（“--help”）看
# 其中有无一个“--text”项以此来判断所使用的是否是UnxUtils版本。其它DOS
# 版本的的sed则无法进行这一转换。但可以用“tr”来实现这一转换。
sed "s/\r//" infile >outfile         # UnxUtils sed v4.0.7 或更高版本
tr -d \r <infile >outfile             # GNU tr 1.22 或更高版本

# 将每一行前导的“空白字符”（空格，制表符）删除
# 使之左对齐
sed 's/^[ \t]*//'                            # 见本文末尾关于'\t'用法的描述

# 将每一行拖尾的“空白字符”（空格，制表符）删除
sed 's/[ \t]*$//'                            # 见本文末尾关于'\t'用法的描述

# 将每一行中的前导和拖尾的空白字符删除
sed 's/^[ \t]*//;s/[ \t]*$//'

# 在每一行开头处插入5个空格（使全文向右移动5个字符的位置）
sed 's/^/     /'

# 以79个字符为宽度，将所有文本右对齐
sed -e :a -e 's/^.\{1,78\}$/ &/;ta'   # 78个字符外加最后的一个空格

# 以79个字符为宽度，使所有文本居中。在方法1中，为了让文本居中每一行的前
# 头和后头都填充了空格。 在方法2中，在居中文本的过程中只在文本的前面填充
# 空格，并且最终这些空格将有一半会被删除。此外每一行的后头并未填充空格。
```

```
sed    -e :a -e 's/^.\{1,77\}$/ & /;ta'                          # 方法1
sed    -e :a -e 's/^.\{1,77\}$/ &/;ta' -e 's/\( *\)\1/\1/'    # 方法2

# 在每一行中查找字串“foo”，并将找到的“foo”替换为“bar”
sed 's/foo/bar/'                          # 只替换每一行中的第一个“foo”字串
sed 's/foo/bar/4'                         # 只替换每一行中的第四个“foo”字串
sed 's/foo/bar/g'                         # 将每一行中的所有“foo”都换成“bar”
sed 's/\(.*\)foo\(.*foo\)/\1bar\2/' # 替换倒数第二个“foo”
sed 's/\(.*\)foo/\1bar/'                  # 替换最后一个“foo”

# 只在行中出现字串“baz”的情况下将“foo”替换成“bar”
sed '/baz/s/foo/bar/g'

# 将“foo”替换成“bar”，并且只在行中未出现字串“baz”的情况下替换
sed '/baz/!s/foo/bar/g'

# 不管是“scarlet”“ruby”还是“puce”，一律换成“red”
sed 's/scarlet/red/g;s/ruby/red/g;s/puce/red/g'   #对多数的sed都有效
gsed 's/scarlet\|ruby\|puce/red/g'                # 只对GNU sed有效

# 倒置所有行，第一行成为最后一行，依次类推（模拟“tac”）。
# 由于某些原因，使用下面命令时HHsed v1.5会将文件中的空行删除
sed '1!G;h;$!d'                   # 方法1
sed -n '1!G;h;$p'                 # 方法2

# 将行中的字符逆序排列，第一个字成为最后一字，……（模拟“rev”）
sed '/\n/!G;s/\(.\)\(.*\n\)/&\2\1/;//D;s/.//'

# 将每两行连接成一行（类似“paste”）
sed '$!N;s/\n/ /'
```

```
# 如果当前行以反斜杠“\”结束，则将下一行并到当前行末尾
# 并去掉原来行尾的反斜杠
sed -e :a -e '/\\$/N; s/\\\n//; ta'

# 如果当前行以等号开头，将当前行并到上一行末尾
# 并以单个空格代替原来行头的“=”
sed -e :a -e '$!N;s/\n=/ /;ta' -e 'P;D'

# 为数字字串增加逗号分隔符号，将“1234567”改为“1,234,567”
gsed ':a;s/\B[0-9]\{3\}\>/,&/;ta'                    # GNU sed
sed -e :a -e 's/\(.*[0-9]\)\([0-9]\{3\}\)/\1,\2/;ta'   # 其他sed

# 为带有小数点和负号的数值增加逗号分隔符（GNU sed）
gsed -r ':a;s/(^|[^0-9.])([0-9]+)([0-9]{3})/\1\2,\3/g;ta'

# 在每5行后增加一空白行 （在第5，10，15，20，等行后增加一空白行）
gsed '0~5G'                          # 只对GNU sed有效
sed 'n;n;n;n;G;'                     # 其他sed
```

选择性地显示特定行：

```
# 显示文件中的前10行 （模拟“head”的行为）
sed 10q

# 显示文件中的第一行 （模拟“head -1”命令）
sed q

# 显示文件中的最后10行 （模拟“tail”）
sed -e :a -e '$q;N;11,$D;ba'
```

```
# 显示文件中的最后2行（模拟“tail -2”命令）
sed '$!N;$!D'

# 显示文件中的最后一行（模拟“tail -1”）
sed '$!d'                                  # 方法1
sed -n '$p'                                # 方法2

# 显示文件中的倒数第二行
sed -e '$!{h;d;}' -e x                     # 当文件中只有一行时，输入空行
sed -e '1{$q;}' -e '$!{h;d;}' -e x   # 当文件中只有一行时，显示该行
sed -e '1{$d;}' -e '$!{h;d;}' -e x   # 当文件中只有一行时，不输出

# 只显示匹配正则表达式的行（模拟“grep”）
sed -n '/regexp/p'                         # 方法1
sed '/regexp/!d'                           # 方法2

# 只显示“不”匹配正则表达式的行（模拟“grep -v”）
sed -n '/regexp/!p'                        # 方法1，与前面的命令相对应
sed '/regexp/d'                            # 方法2，类似的语法

# 查找“regexp”并将匹配行的上一行显示出来，但并不显示匹配行
sed -n '/regexp/{g;1!p;};h'

# 查找“regexp”并将匹配行的下一行显示出来，但并不显示匹配行
sed -n '/regexp/{n;p;}'

# 显示包含“regexp”的行及其前后行，并在第一行之前加上“regexp”所
# 在行的行号 （类似“grep -A1 -B1”）
sed -n -e '/regexp/{=;x;1!p;g;$!N;p;D;}' -e h

# 显示包含“AAA”、“BBB”或“CCC”的行（任意次序）
```

```
sed '/AAA/!d; /BBB/!d; /CCC/!d'   # 字串的次序不影响结果

# 显示包含“AAA”、“BBB”和“CCC”的行（固定次序）
sed '/AAA.*BBB.*CCC/!d'

# 显示包含“AAA”“BBB”或“CCC”的行 （模拟“egrep”）
sed -e '/AAA/b' -e '/BBB/b' -e '/CCC/b' -e d      # 多数sed
gsed '/AAA\|BBB\|CCC/!d'                          # 对GNU sed有效

# 显示包含“AAA”的段落 （段落间以空行分隔）
# HHsed v1.5 必须在“x;”后加入“G;”，接下来的3个脚本都是这样
sed -e '/./{H;$!d;}' -e 'x;/AAA/!d;'

# 显示包含“AAA”“BBB”和“CCC”三个字串的段落 （任意次序）
sed -e '/./{H;$!d;}' -e 'x;/AAA/!d;/BBB/!d;/CCC/!d'

# 显示包含“AAA”、“BBB”、“CCC”三者中任一字串的段落 （任意次序）
sed -e '/./{H;$!d;}' -e 'x;/AAA/b' -e '/BBB/b' -e '/CCC/b' -e d
gsed '/./{H;$!d;};x;/AAA\|BBB\|CCC/b;d'           # 只对GNU sed有效

# 显示包含65个或以上字符的行
sed -n '/^.\{65\}/p'

# 显示包含65个以下字符的行
sed -n '/^.\{65\}/!p'                 # 方法1，与上面的脚本相对应
sed '/^.\{65\}/d'                     # 方法2，更简便一点的方法

# 显示部分文本——从包含正则表达式的行开始到最后一行结束
sed -n '/regexp/,$p'

# 显示部分文本——指定行号范围（从第8至第12行，含8和12行）
```

```
sed -n '8,12p'                    # 方法1
sed '8,12!d'                      # 方法2

# 显示第52行
sed -n '52p'                      # 方法1
sed '52!d'                        # 方法2
sed '52q;d'                       # 方法3, 处理大文件时更有效率

# 从第3行开始，每7行显示一次
gsed -n '3~7p'                    # 只对GNU sed有效
sed -n '3,${p;n;n;n;n;n;n;}'      # 其他sed

# 显示两个正则表达式之间的文本（包含）
sed -n '/Iowa/,/Montana/p'        # 区分大小写方式
```

选择性地删除特定行：

```
# 显示通篇文档，除了两个正则表达式之间的内容
sed '/Iowa/,/Montana/d'

# 删除文件中相邻的重复行（模拟“uniq”）
# 只保留重复行中的第一行，其他行删除
sed '$!N; /^\(.*\)\n\1$/!P; D'

# 删除文件中的重复行，不管有无相邻。注意hold space所能支持的缓存
# 大小，或者使用GNU sed。
sed -n 'G; s/\n/&&/; /^\([ -~]*\n\).*\n\1/d; s/\n//; h; P'

# 删除除重复行外的所有行（模拟“uniq -d”）
sed '$!N; s/^\(.*\)\n\1$/\1/; t; D'
```

```
# 删除文件中开头的10行
sed '1,10d'

# 删除文件中的最后一行
sed '$d'

# 删除文件中的最后两行
sed 'N;$!P;$!D;$d'

# 删除文件中的最后10行
sed -e :a -e '$d;N;2,10ba' -e 'P;D'     # 方法1
sed -n -e :a -e '1,10!{P;N;D;};N;ba'    # 方法2

# 删除8的倍数行
gsed '0~8d'                                  # 只对GNU sed有效
sed 'n;n;n;n;n;n;n;d;'                       # 其他sed

# 删除匹配式样的行
sed '/pattern/d'                             # 删除含pattern的行。当然pattern
                                                 # 可以换成任何有效的正则表达式

# 删除文件中的所有空行（与“grep '.' ”效果相同）
sed '/^$/d'                                  # 方法1
sed '/./!d'                                  # 方法2

# 只保留多个相邻空行的第一行。并且删除文件顶部和尾部的空行。
# （模拟“cat -s”）
sed '/./,/^$/!d'            #方法1，删除文件顶部的空行，允许尾部保留一空行
sed '/^$/N;/\n$/D'          #方法2，允许顶部保留一空行，尾部不留空行
```

```
# 只保留多个相邻空行的前两行。
sed '/^$/N;/\n$/N;//D'

# 删除文件顶部的所有空行
sed '/./,$!d'

# 删除文件尾部的所有空行
sed -e :a -e '/^\n*$/{$d;N;ba' -e '}'   # 对所有sed有效
sed -e :a -e '/^\n*$/N;/\n$/ba'             # 同上，但只对 gsed 3.02.*有效

# 删除每个段落的最后一行
sed -n '/^$/{p;h;};/./{x;/./p;}'
```

特殊应用：

```
# 移除手册页（man page）中的nroff标记。在Unix System V或bash shell下使
# 用'echo'命令时可能需要加上 -e 选项。
sed "s/.`echo \\\b`//g"      # 外层的双括号是必须的（Unix环境）
sed 's/.^H//g'                  # 在bash或tcsh中，按 Ctrl-V 再按 Ctrl-H
sed 's/.\x08//g'                # sed 1.5，GNU sed，ssed所使用的十六进制的表示方法

# 提取新闻组或 e-mail 的邮件头
sed '/^$/q'                      # 删除第一行空行后的所有内容

# 提取新闻组或 e-mail 的正文部分
sed '1,/^$/d'                   # 删除第一行空行之前的所有内容

# 从邮件头提取“Subject”（标题栏字段），并移除开头的“Subject:”字样
sed '/^Subject: */!d; s///;q'
```

```
# 从邮件头获得回复地址
sed '/^Reply-To:/q; /^From:/h; /./d;g;q'

# 获取邮件地址。在上一个脚本所产生的那一行邮件头的基础上进一步的将非电邮
# 地址的部分剔除。（见上一脚本）
sed 's/ *(.*)//; s/>.*//; s/.*[:<] *//'

# 在每一行开头加上一个尖括号和空格（引用信息）
sed 's/^/> /'

# 将每一行开头处的尖括号和空格删除（解除引用）
sed 's/^> //'

# 移除大部分的HTML标签（包括跨行标签）
sed -e :a -e 's/<[^>]*>//g;/</N;//ba'

# 将分成多卷的uuencode文件解码。移除文件头信息，只保留uuencode编码部分。
# 文件必须以特定顺序传给sed。下面第一种版本的脚本可以直接在命令行下输入；
# 第二种版本则可以放入一个带执行权限的shell脚本中。（由Rahul Dhesi的一
# 个脚本修改而来。）
sed '/^end/,/^begin/d' file1 file2 ... fileX | uudecode     # vers. 1
sed '/^end/,/^begin/d' "$@" | uudecode                      # vers. 2

# 将文件中的段落以字母顺序排序。段落间以（一行或多行）空行分隔。GNU sed使用
# 字元“\v”来表示垂直制表符，这里用它来作为换行符的占位符——当然你也可以
# 用其他未在文件中使用的字符来代替它。
sed '/./{H;d;};x;s/\n/={NL}=/g' file | sort | sed '1s/={NL}=//;s/={NL}=/\n/g'
gsed '/./{H;d};x;y/\n/\v/' file | sort | sed '1s/\v//;y/\v/\n/'

# 分别压缩每个.TXT文件，压缩后删除原来的文件并将压缩后的.ZIP文件
# 命名为与原来相同的名字（只是扩展名不同）。（DOS环境：“dir /b”
```

```
# 显示不带路径的文件名)。
echo @echo off >zipup.bat
dir /b *.txt | sed "s/^\(.*\)\.TXT/pkzip -mo \1 \1.TXT/" >>zipup.bat
```

使用SED：Sed接受一个或多个编辑命令，并且每读入一行后就依次应用这些命令。当读入第一行输入后，sed对其应用所有的命令，然后将结果输出。接着再读入第二行输入，对其应用所有的命令……并重复这个过程。上一个例子中sed由标准输入设备（即命令解释器，通常是以管道输入的形式）获得输入。在命令行给出一个或多个文件名作为参数时，这些文件取代标准输入设备成为sed的输入。sed的输出将被送到标准输出（显示器）。因此：

```
cat filename | sed '10q'          # 使用管道输入
sed '10q' filename                  # 同样效果，但不使用管道输入
sed '10q' filename > newfile        # 将输出转移（重定向）到磁盘上
```

要了解sed命令的使用说明，包括如何通过脚本文件（而非从命令行）来使用这些命令，请参阅《sed & awk》第二版，作者Dale Dougherty和Arnold Robbins（O'Reilly，1997；http://www.ora.com)，《 UNIX Text Processing》，作者Dale Dougherty和Tim O'Reilly（Hayden Books，1987）或者是Mike Arst写的教程——压缩包的名称是“U-SEDIT2.ZIP”（在许多站点上都找得到）。要发掘sed的潜力，则必须对“正则表达式”有足够的理解。正则表达式的资料可以看《Mastering Regular Expressions》作者Jeffrey Friedl（O'reilly 1997)。Unix系统所提供的手册页（“man”）也会有所帮助（试一下这些命令“man sed”、“man regexp”，或者看“man ed”中关于正则表达式的部分），但手册提供的信息比较“抽象”——这也是它一直为人所诟病的。不过，它本来就不是用来教初学者如何使用sed或正则表达式的教材，而只是为那些熟悉这些工具的人提供的一些文本参考。

括号语法：前面的例子对sed命令基本上都使用单引号（'...'）而非双引号（"..."）这是因为sed通常是在Unix平台上使用。单引号下，Unix的shell（命令

解释器）不会对美元符（$）和后引号（`...`）进行解释和执行。而在双引号下美元符会被展开为变量或参数的值，后引号中的命令被执行并以输出的结果代替后引号中的内容。而在“csh”及其衍生的shell中使用感叹号（!）时需要在其前面加上转义用的反斜杠（就像这样：\!）以保证上面所使用的例子能正常运行（包括使用单引号的情况下）。DOS版本的Sed则一律使用双引号（"..."）而不是引号来圈起命令。

'\t'的用法：为了使本文保持行文简洁，我们在脚本中使用'\t'来表示一个制表符。但是现在大部分版本的sed还不能识别'\t'的简写方式，因此当在命令行中为脚本输入制表符时，你应该直接按TAB键来输入制表符而不是输入'\t'。下列的工具软件都支持'\t'做为一个正则表达式的字元来表示制表符：awk、perl、HHsed、sedmod以及GNU sed v3.02.80。

不同版本的SED：不同的版本间的sed会有些不同之处，可以想象它们之间在语法上会有差异。具体而言，它们中大部分不支持在编辑命令中间使用标签（:name）或分支命令（b,t），除非是放在那些的末尾。这篇文档中我们尽量选用了可移植性较高的语法，以使大多数版本的sed的用户都能使用这些脚本。不过GNU版本的sed允许使用更简洁的语法。想像一下当读者看到一个很长的命令时的心情：

```
sed -e '/AAA/b' -e '/BBB/b' -e '/CCC/b' -e d
```

好消息是GNU sed能让命令更紧凑：

```
sed '/AAA/b;/BBB/b;/CCC/b;d'        # 甚至可以写成
sed '/AAA\|BBB\|CCC/b;d'
```

此外，请注意虽然许多版本的sed接受象“/one/ s/RE1/RE2/”这种在's'前带有空格的命令，但这些版本中有些却不接受这样的命令:“/one/! s/RE1/RE2/”。这时只需要把中间的空格去掉就行了。

速度优化：当由于某种原因（比如输入文件较大、处理器或硬盘较慢等）需要提高

命令执行速度时，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
```

当只需要显示文件的前面的部分或需要删除后面的内容时，可以在脚本中使用“q”命令（退出命令）。在处理大的文件时，这会节省大量时间。因此：

```
sed -n '45,50p' filename                # 显示第45到50行
sed -n '51q;45,50p' filename          # 一样，但快得多
```

如果你有其他的单行脚本想与大家分享或者你发现了本文档中错误的地方，请发电子邮件给本文档的作者（Eric Pement）。邮件中请记得提供你所使用的sed版本、该sed所运行的操作系统及对问题的适当描述。本文所指的单行脚本指命令行的长度在65个字符或65个以下的sed脚本〔译注1〕。本文档的各种脚本是由以下所列作者所写或提供：

```
Al Aab                                  # 建立了“seders”邮件列表
Edgar Allen                             # 许多方面
Yiorgos Adamopoulos                     # 许多方面
Dale Dougherty                          # 《sed & awk》作者
Carlos Duarte                           # 《do it with sed》作者
Eric Pement                             # 本文档的作者
Ken Pizzini                             # GNU sed v3.02 的作者
S.G. Ravenhall                          # 去html标签脚本
Greg Ubben                              # 有诸多贡献并提供了许多帮助
```

译注1：大部分情况下，sed脚本无论多长都能写成单行的形式（通过`-e'选项和`;'

号）——只要命令解释器支持，所以这里说的单行脚本除了能写成一行还对长度有所限制。因为这些单行脚本的意义不在于它们是以单行的形式出现。而是让用户能方便地在命令行中使用这些紧凑的脚本才是其意义所在。