使用 sed 编辑器

作者：Emmett Dulaney

sed 编辑器是 Linux 系统管理员的工具包中最有用的资产之一，因此，有必要彻底地了解其应用

Linux 操作系统最大的一个好处是它带有各种各样的实用工具。存在如此之多不同的实用工具，几乎不可能知道并了解所有这些工具。可以简化关键情况下操作的一个实用工具是 sed。它是任何管理员的工具包中最强大的工具之一，并且可以证明它自己在关键情况下非常有价值。

sed 实用工具是一个“编辑器”，但它与其它大多数编辑器不同。除了不面向屏幕之外，它还是非交互式的。这意味着您必须将要对数据执行的命令插入到命令行或要处理的脚本中。当显示它时，请忘记您在使用 Microsoft Word 或其它大多数编辑器时拥有的交互式编辑文件功能。sed 在一个文件（或文件集）中非交互式、并且不加询问地接收一系列的命令并执行它们。因而，它流经文本就如同水流经溪流一样，因而 sed 恰当地代表了流编辑器。它可以用来将所有出现的 "Mr. Smyth" 修改为 "Mr. Smith"，或将 "tiger cub" 修改为 "wolf cub"。流编辑器非常适合于执行重复的编辑，这种重复编辑如果由人工完成将花费大量的时间。其参数可能和一次性使用一个简单的操作所需的参数一样有限，或者和一个具有成千上万行要进行编辑修改的脚本文件一样复杂。sed 是 Linux 和 UNIX 工具箱中最有用的工具之一，且使用的参数非常少。

sed 的工作方式

sed 实用工具按顺序逐行将文件读入到内存中。然后，它执行为该行指定的所有操作，并在完成请求的修改之后将该行放回到内存中，以将其转储至终端。完成了这一行上的所有操作之后，它读取文件的下一行，然后重复该过程直到它完成该文件。如同前面所提到的，默认输出是将每一行的内容输出到屏幕上。在这里，开始涉及到 两个重要的因素—首先，输出可以被重定向到另一文件中，以保存变化；第二，源文件（默认地）保持不被修改。sed 默认读取整个文件并对其中的每一行进行修改。不过，可以按需要将操作限制在指定的行上。

该实用工具的语法为：

```
sed [options] '{command}' [filename]
```

在这篇文章中，我们将浏览最常用的命令和选项，并演示它们如何工作，以及它们适于在何处使用。

替换命令

sed 实用工具以及其它任何类似的编辑器的最常用的命令之一是用一个值替换另一个值。用来实现这一目的的操作的命令部分语法是：

```
's/{old value}/{new value}/'
```

因而，下面演示了如何非常简单地将 "tiger" 修改为 "wolf"：

```
$ echo The tiger cubs will meet on Tuesday after school | sed 's/tiger/wolf/'
The wolf cubs will meet on Tuesday after school
$
```

注意如果输入是源自之前的命令输出，则不需要指定文件名—同样的原则也适用于 awk、sort 和其它大多数 Linux\UNIX 命令行实用工具程序。

多次修改

如果需要对同一文件或行作多次修改，可以有三种方法来实现它。第一种是使用 "-e" 选项，它通知程序使用了多条编辑命令。例如：

```
$ echo The tiger cubs will meet on Tuesday after school | sed -e ' s/tiger/wolf/' -e
's/after/before/'
The wolf cubs will meet on Tuesday before school
$
```

这是实现它的非常复杂的方法，因此 "-e" 选项不常被大范围使用。更好的方法是用分号来分隔命令：

```
$ echo The tiger cubs will meet on Tuesday after school | sed 's/tiger/wolf/; s/after/before/'
The wolf cubs will meet on Tuesday before school
$
```

注意分号必须是紧跟斜线之后的下一个字符。如两者之间有一个空格，操作将不能成功完成，并返回一条错误消息。这两种方法都很好，但许多管理员更喜欢另一种方法。要注意的一个关键问题是，两个撇号（''）之间的全部内容都被解释为 sed 命令。直到您输入了第二个撇号，读入这些命令的 shell 程序才会认为您完成了输入。这意味着可以在多行上输入命令—同时 Linux 将提示符从 PS1 变为一个延续提示符（通常为 ">"）—直到输入了第二个撇号。一旦输入了第二个撇号，并且按下了 Enter 键，则处理就进行并产生相同的结果，如下：

```
$ echo The tiger cubs will meet on Tuesday after school | sed '
> s/tiger/wolf/
> s/after/before/'
The wolf cubs will meet on Tuesday before school
$
```

全局修改

让我们开始一次看似简单的编辑。假定在要修改的消息中出现了多次要修改的项目。默认方式下，结果可能和预期的有所不同，如下所示：

```
$ echo The tiger cubs will meet this Tuesday at the same time
as the meeting last Tuesday | sed 's/Tuesday/Thursday/'
The tiger cubs will meet this Thursday at the same time
as the meeting last Tuesday
$
```

与将出现的每个 "Tuesday" 修改为 "Thursday" 相反，sed 编辑器在找到一个要修改的项目并作了修改之后继续处理下一行，而不读整行。sed 命令功能大体上类似于替换命令，这意味着它们都处理每一行中出现的第一个选定序列。为了替换出现的每一个项目，在同一行中出现多个要替换的项目的情况下， 您必须指定在全局进行该操作：

```
$ echo The tiger cubs will meet this Tuesday at the same time as the meeting last Tuesday | sed
's/Tuesday/Thursday/g' The tiger cubs will meet this Thursday at the same time as the meeting
last Thursday
$
```

请记住不管您要查找的序列是否仅包含一个字符或词组，这种对全局化的要求都是必需的。

sed 还可以用来修改记录字段分隔符。例如，以下命令将把所有的 tab 修改为空格：

```
sed 's/ / /g'
```

其中，第一组斜线之间的项目是一个 tab，而第二组斜线之间的项目是一个空格。作为一条通用的规则，sed 可以用来将任意的可打印字符修改为任意其它的可打印字符。如果您想将不可打印字符修改为可打印字符—例如，铃铛修改为单词 "bell"—sed 不是适于完成这项工作的工具（但 tr 是）。

有时，您不想修改在一个文件中出现的所有指定项目。有时，您只想在满足某些条件时才作修改—例如，在与其它一些数据匹配之后才作修改。为了说明这一点，请考虑以下文本文件：

```
$ cat sample_one
one     1
two     1
three   1
one     1
two     1
two     1
three   1
$
```

假定希望用 "2" 来替换 "1"，但仅在单词 "two" 之后才作替换，而不是每一行的所有位置。通过指定在给出替换命令之前必须存在一次匹配，可以实现这一点：

```
$ sed '/two/ s/1/2/' sample_one
one     1
two     2
three   1
one     1
two     2
two     2
three   1
$
```

现在，使其更加准确：

```
$ sed '
> /two/ s/1/2/
> /three/ s/1/3/' sample_one
one     1
two     2
three   3
one     1
two     2
two     2
three   3
$
```

请再次记住唯一改变了的是显示。如果您查看源文件，您将发现它始终保持不变。您必须将输出保存至另一个文件，以实现永久保存。值得重复的是，不对源文件作修改实际是祸中有福—它让您能够对文件进行试验而不会造成任何实际的损害，直到让正确命令以您预期和希望 的方式进行工作。

以下命令将修改后的输出保存至一个新的文件：

```
$ sed '
> /two/ s/1/2/
> /three/ s/1/3/' sample_one > sample_two
```

该输出文件将所有修改合并在其中，且这些修改通常将在屏幕上显示。现在可以用 head、cat 或任意其它类似的实用工具来进行查看。

脚本文件

sed 工具允许您创建一个脚本文件，其中包含从该文件而不是在命令行进行处理的命令，并且 sed 工具通过 "-f" 选项来引用。通过创建一个脚本文件，您能够一次又一次地重复运行相同的操作，并指定比每次希望从命令行进行处理的操作详细得多的操作。

考虑以下脚本文件：

```
$ cat sedlist
/two/ s/1/2/
/three/ s/1/3/
$
```

现在可以在数据文件上使用脚本文件，获得和我们之前看到的相同的结果：

```
$ sed -f sedlist sample_one
one     1
two     2
three   3
one     1
two     2
two     2
three   3
$
```

注意当调用 "-f" 选项时，在源文件内或命令行中不使用撇号。脚本文件，也称为源文件，对于想重复多次的操作和从命令行运行可能出错的复杂命令很有价值。编辑源文件并修改一个字符比在命令行中重新输入一条多行的项目要容易得多。

限制行

编辑器默认查看输入到流编辑器中的每一行，且默认在输入到流编辑器中的每一行上进行编辑。这可以通过在发出命令之前指定约束条件来进行修改。例如，只在此示例文件的输出的第 5 和第 6 行中用 "2" 来替换 "1"，命令将为：

```
$ sed '5,6 s/1/2/' sample_one
one     1
two     1
three   1
one     1
two     2
two     2
three   1
$
```

在这种情况下，因为要修改的行是专门指定的，所以不需要替换命令。因此，您可以灵活地根据匹配准则（可以是行号或一种匹配模式）来选择要修改哪些行（从根本上限制修改）。

禁止显示

sed 默认将来自源文件的每一行显示到屏幕上（或重定向到一个文件中），而无论该行是否受到编辑操作的影响，"-n" 参数覆盖了这一操作。"-n" 覆盖了所有的显示，并且不显示任何一行，而无论它们是否被编辑操作修改。例如：

```
$ sed -n -f sedlist sample_one
```

```
$

$ sed -n -f sedlist sample_one > sample_two
$ cat sample_two
$
```

在第一个示例中，屏幕上不显示任何东西。在第二个示例中，不修改任何东西，因此不将任何东西写到新的文件中—它最后是空的。这不是否定了编辑的全部目的吗？为什么这是有用的？它是有用的仅因为 "-n" 选项能够被一条显示命令 (-p) 覆盖。为了说明这一点，假定现在像下面这样对脚本文件进行了修改：

```
$ cat sedlist
/two/ s/1/2/p
/three/ s/1/3/p
$
```

然后下面是运行它的结果：

```
$ sed -n -f sedlist sample_one
two     2
three   3
two     2
two     2
three   3
$
```

保持不变的行全部不被显示。只有受到编辑操作影响的行被显示了。在这种方式下，可以仅取出这些行，进行修改，然后把它们放到一个单独的文件中：

```
$ sed -n -f sedlist sample_one > sample_two
$

$ cat sample_two
two     2
three   3
two     2
two     2
three   3
$
```

利用它的另一种方法是只显示一定数量的行。例如，只显示 2-6 行，同时不做其它的编辑修改：

```
$ sed -n '2,6p' sample_one
two     1
three   1
one     1
two     1
two     1
$
```

其它所有的行被忽略，只有 2-6 行作为输出显示。这是一项出色的功能，其它任何工具都不能容易地实现。Head 将显示一个文件的顶部，而 tail 将显示一个文件的底部，但 sed 允许从任意位置取出想要的任意内容。

删除行

用一个值替换另一个值远非流编辑器可以执行的唯一功能。它还具有许多的潜在功能，在我看来第二种最常用的功能是删除。删除与替换的工作方式相同，只是它删除指定的行（如您想删除一个单词而不是一行，不要考虑删除，而应考虑用空的内容来替换它—s/cat//）。

该命令的语法是：

```
'{what to find} d'
```

从 sample_one 文件中删除包含 "two" 的所有行：

```
$ sed '/two/ d' sample_one
one     1
three   1
one     1
three   1
$
```

从显示屏中删除前三行，而不管它们的内容是什么：

```
$ sed '1,3 d' sample_one
one     1
two     1
two     1
three   1
$
```

只显示剩下的行，前三行不在显示屏中出现。对于流编辑器，一般当它们涉及到全局表达式时，特别是应用于删除操作时，有几点要记住：

1. 上三角号（^）表示一行的开始，因此，如果 "two" 是该行的头三个字符，则

   ```
   sed '/^two/ d' sample_one
   ```

   将只删除该行。

2. 美元符号（$）代表文件的结尾，或一行的结尾，因此，如果 "two" 是该行的最后三个字符，则

   ```
   sed '/two$/ d' sample_one
   ```

   将只删除该行。

将这两者结合在一起的结果：

```
sed '/^$/ d' {filename}
```

删除文件中的所有空白行。例如，以下命令将 "1" 替换为 "2"，以及将 "1" 替换为 "3"，并删除文件中所有尾随的空行：

```
$ sed '/two/ s/1/2/; /three/ s/1/3/; /^$/ d' sample_one
one     1
two     1
three   1
one     1
two     2
two     2
three   1
```

```
$
```

其通常的用途是删除一个标题。以下命令将删除文件中所有的行，从第一行直到第一个空行：

```
sed '1,/^$/ d' {filename}
```

添加和插入文本

可以结合使用 sed 和 "a" 选项将文本添加到一个文件的末尾。实现方法如下：

```
$ sed '$a\
> This is where we stop\
> the test' sample_one
one     1
two     1
three   1
one     1
two     1
two     1
three   1
This is where we stop
the test
$
```

在该命令中，美元符号（$）表示文本将被添加到文件的末尾。反斜线（\）是必需的，它表示将插入一个回车符。如果它们被遗漏了，则将导致一个错误，显示该命令是错乱的；在任何要输入回车的地方您必须使用反斜线。

要将这些行添加到第 4 和第 5 个位置而不是末尾，则命令变为：

```
$ sed '3a\
> This is where we stop\
> the test' sample_one
one     1
two     1
three   1
This is where we stop
the test
one     1
two     1
two     1
three   1
$
```

这将文本添加到第 3 行之后。和几乎所有的编辑器一样，您可以选择插入而不是添加（如果您希望这样的话）。这两者的区别是添加跟在指定的行之后，而插入从指定的行开始。当用插入来代替添加时，只需用 "i" 来代替 "a"，如下所示：

```
$ sed '3i\
> This is where we stop\
> the test' sample_one
one     1
two     1
This is where we stop
the test
three   1
```

```
one     1
two     1
two     1
three   1
$
```

新的文本出现在输出的中间位置，而处理通常在指定的操作执行以后继续进行。

读写文件

重定向输出的功能已经演示过了，但需要指出的是，在编辑命令运行期间可以同步地读入和写出文件。例如，执行替换，并将 1-3 行写到名称为 sample_three 的文件中：

```
$ sed '
> /two/ s/1/2/
> /three/ s/1/3/
> 1,3 w sample_three' sample_one
one     1
two     2
three   3
one     1
two     2
two     2
three   3
$

$ cat sample_three
one     1
two     2
three   3
$
```

由于为 w (write) 命令指定了 "1,3"，所以只有指定的行被写到了新文件中。无论被写的是哪些行，所有的行都在默认输出中显示。

修改命令

除了替换项目之外，还可以将行从一个值修改为另一个值。要记住的是，替换是对字符逐个进行，而修改功能与删除类似，它影响整行：

```
$ sed '/two/ c\
> We are no longer using two' sample_one
one     1
We are no longer using two
three   1
one     1
We are no longer using two
We are no longer using two
three   1
$
```

修改命令与替换的工作方式很相似，但在范围上要更大些—将一个项目完全替换为另一个项目，而无论字符内容 或上下文。夸张一点讲，当使用替换时，只有字符 "1" 被字符 "2" 替换，而当使用修改时，原来的整行将被修改。在两种情况下，要寻找的匹配条件都仅为 "two"。

修改全部但……

对于大多数 sed 命令，详细说明各种功能要进行何种修改。利用感叹号，可以在除指定位置之外的任何地方执行修改—与默认的操作完全相反。

例如，要删除包含单词 "two" 的所有行，操作为：

```
$ sed '/two/ d' sample_one
one     1
three   1
one     1
three   1
$
```

而要删除除包含单词 "two" 的行之外的所有行，则语法变为：

```
$ sed '/two/ !d' sample_one
two     1
two     1
two     1
$
```

如果您有一个文件包含一系列项目，并且想对文件中的每个项目执行一个操作，那么首先对那些项目进行一次智能扫描并考虑将要做什么是很重要的。为了使事情变得更简单，您可以将 sed 与任意迭代例程（for、while、until）结合来实现这一目的。

比如说，假定您有一个名为 "animals" 的文件，其中包含以下项目：

```
pig
horse
elephant
cow
dog
cat
```

您希望运行以下例程：

```
#mcd.ksh
for I in $*
do
echo Old McDonald had a $I
echo E-I, E-I-O
done
```

结果将为，每一行都显示在 "Old McDonald has a" 的末尾。虽然对于这些项目的大部分这是正确的，但对于 "elephant" 项目，它有语法错误，因为结果应当为 "an elephant" 而不是 "a elephant"。利用 sed，您可以在来自 shell 文件的输出中检查这种语法错误，并通过首先创建一个命令文件来即时地更正它们：

```
#sublist
/ a a/ s/ a / an /
/ a e/ s/ a / an /
/a i/ s / a / an /
/a o/ s/ a / an /
/a u/ s/ a / an /
```

然后执行以下过程：

```
$ sh mcd.ksh 'cat animals' | sed -f sublist
```

现在，在运行了 mcd 脚本之后，sed 将在输出中搜索单个字母 a （空格，"a"，空格）之后紧跟了一个元音的任意位置。如果这种位置存在，它将把该序列修改为空格，"an"，空格。这样就使问题更正后才显示在屏幕上， 并确保各处的编辑人员在晚上可以更容易地入睡。结果是：

```
Old McDonald had a pig
E-I, E-I-O
Old McDonald had a horse
E-I, E-I-O
Old McDonald had an elephant
E-I, E-I-O
Old McDonald had a cow
E-I, E-I-O
Old McDonald had a dog
E-I, E-I-O
Old McDonald had a cat
E-I, E-I-O
```

提前退出

sed 默认读取整个文件，并只在到达末尾时才停止。不过，您可以使用退出命令提前停止处理。只能指定一条退出命令，而处理将一直持续直到满足调用退出命令的条件。

例如，仅在文件的前五行上执行替换，然后退出：

```
$ sed '
> /two/ s/1/2/
> /three/ s/1/3/
> 5q' sample_one
one     1
two     2
three   3
one     1
two     2
$
```

在退出命令之前的项目可以是一个行号（如上所示），或者一条查找/匹配命令：

```
$ sed '
> /two/ s/1/2/
> /three/ s/1/3/
> /three/q' sample_one
one     1
two     2
three   3
$
```

您还可以使用退出命令来查看超过一定标准数目的行，并增加比 head 中的功能更强的功能。例如，head 命令允许您指定您想要查看一个文件的前多少行—默认数为 10，但可以使用从 1 到 99 的任意一个数字。如果您想查看一个文件的前 110 行，您用 head 不能实现这一目的，但用 sed 可以：

```
sed 110q filename
```

处理问题

当使用 sed 时，要记住的重要事项是它的工作方式。它的工作方式是：读入一行，在该行上执行它已知要执行的所有任务，然后继续处理下一行。每一行都受给定的每一个编辑命令的影响。

如果您的操作顺序没有十分彻底地考虑清楚，那么这可能会很麻烦。例如，假定您需要将所有的 "two" 项目修改为 "three"，然后将所有的 "three" 修改为 "four"：

```
$ sed '
> /two/ s/two/three/
> /three/ s/three/four/' sample_one
one     1
four     1
four   1
one     1
four     1
four     1
four   1
$
```

最初读取的 "two" 被修改为 "three"。然后它满足为下一次编辑建立的准则，从而变为 "four"。最终的结果不是想要的结果—现在除了 "four" 没有别的项目了，而本来应该有 "three" 和 "four"。

当执行这种操作时，您必须非常用心地注意指定操作的方式，并按某种顺序来安排它们，使得操作之间不会互相影响。例如：

```
$ sed '
> /three/ s/three/four/
> /two/ s/two/three/' sample_one
one     1
three     1
four   1
one     1
three     1
three     1
four   1
$
```

这非常有效，因为 "three" 值在 "two" 变成 "three" 之前得到修改。

标签和注释

可以在 sed 脚本文件中放置标签，这样一旦文件变得庞大，可以更容易地说明正在发生的事情。存在各种各样与这些标签相关的命令，它们包括：

1. : 冒号表示一个标签名称。例如：

   ```
   :HERE
   ```

   以冒号开始的标签可以由 "b" 和 "t" 命令处理。

2. b {label} 充当 "goto" 语句的作用，将处理发送至前面有一个冒号的标签。例如，

   ```
   b HERE
   ```

   将处理发送给行

   ```
   :HERE
   ```

如果紧跟 b 之后没有指定任何标签，则处理转至脚本文件的末尾。

3. t {label} 只要自上次输入行或执行一次 "t" 命令以来进行了替换操作，就转至该标签。和 "b" 一样，如果没有给定标签名，则处理转至脚本文件的末尾。
4. # 符号作为一行的第一个字符将使整行被当作注释处理。注释行与标签不同，不能使用 b 或 t 命令来转到注释行上。

进一步的研究

sed 实用工具是 Linux 管理员拥有的最强大和灵活的工具之一。虽然本文覆盖了许多基础知识，但对于这一具有丰富功能的工具仅是蜻蜓点水。关于更多信息，最好的来源之一是 Dale Dougherty 和 Arnold Robbins 的著作 sed & awk，现在从 O'Reilly & Associates 出版社推出了其第二版（参加“接下来的步骤”）。该出版社还推出了一本可以随身携带的袖珍参考。

Emmett Dulaney (edulaney@iquest.net) 获得了 18 种供应商认证。他编写了数本关于 Linux、UNIX 和认证研究的书籍，并在许多会议上进行了演讲，而且他是 Mercury Technical Solutions 的前合作伙伴。