Sed

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 LinuxUNIX 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。如果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，如下所示：

```
$ 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，并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。