Stream Editor - Basic Commands

This chapter describes several useful SED commands.

Delete Command

SED provides various commands to manipulate text. Let us first explore about the **delete** command. Here is how you execute a delete command:

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

address1 and **address2** are the starting and the ending addresses respectively, which can be either line numbers or pattern strings. Both of these addresses are optional parameters.

As the name suggests, the delete command is used to perform delete operation and since the SED operates on line, we can say that this command is used to delete lines. Note that the delete command removes lines only from the pattern buffer; the line is not sent to the output stream and the original file remains unchanged. The following example illustrates the point.

```
[jerry]$ sed 'd' books.txt
```

But where is the output? If no line address is provided, then the SED operates on every line by default. Hence, it deletes all the lines from the pattern buffer. That is why the command does not print anything on the standard output.

Let us instruct the SED to operate only on certain lines. The following example removes the 4th line only.

```
[jerry]$ sed '4d' books.txt
```

On executing the above code, you get the following result:

```
1) A Storm of Swords, George R. R. Martin, 1216
2) The Two Towers, J. R. R. Tolkien, 352
3) The Alchemist, Paulo Coelho, 197
5) The Pilgrimage, Paulo Coelho, 288
6) A Game of Thrones, George R. R. Martin, 864
```

Additionally, SED also accepts address range using comma(,). We can instruct the SED to remove N1 to N2 lines. For instance, the following example deletes all the lines from 2 through 4.

```
[jerry]$ sed '2, 4 d' books.txt
```

On executing the above code, you get the following result:

```
1) A Storm of Swords, George R. R. Martin, 1216
5) The Pilgrimage, Paulo Coelho, 288
6) A Game of Thrones, George R. R. Martin, 864
```

SED's address range is not only limited to numbers. We can also specify patterns as an address. The following example removes all the books of the author Paulo Coelho.

```
[jerry]$ sed '/Paulo Coelho/d' books.txt
```

On executing the above code, you get the following result:

```
1) A Storm of Swords, George R. R. Martin, 1216
2) The Two Towers, J. R. R. Tolkien, 352
4) The Fellowship of the Ring, J. R. R. Tolkien, 432
6) A Game of Thrones, George R. R. Martin, 864
```

We can also specify an address range using textual pattern. The following example removes all lines between the patterns **Storm** and **Fellowship**.

```
[jerry]$ sed '/Storm/,/Fellowship/d' books.txt
5) The Pilgrimage, Paulo Coelho, 288
6) A Game of Thrones, George R. R. Martin, 864
```

In addition to this, we can also use dollar($), plus(+), and tilde(~) operators with SED.

Write Command

One of the important operations we perform on any file is backup, i.e., we make another copy of the file. SED provides the **write** command to store the contents of the pattern buffer in a file. Given below is the syntax of the **write** command which is similar to the **delete** command.

```
[address1[,address2]]w file
```

Here, **address1** and **address2** are the starting and the ending address respectively, which can be either line numbers or pattern strings. Both of these addresses are optional parameters.

In the above syntax, **w** refers to the write command and **file** is the file name in which you store contents. Be careful with the **file** parameter. When a file name is provided, the SED creates a file on the fly if it is not present, and overwrites it if it is already present.

Let us make an exact copy of the file using SED. Note that there must be exactly one space between **w** and **file**.

```
[jerry]$ sed -n 'w books.bak' books.txt
```

We created another file called **books.bak.** Now verify that both the files have identical content.

```
[jerry]$ diff books.txt books.bak
[jerry]$ echo $?
```

On executing the above code, you get the following result:

```
0
```

You may assume that the **cp** command does exactly the same thing. Yes! The **cp** command does the same thing, but SED is a matured utility. It allows creating a file containing only certain lines from the source file. Let us store only even lines to another file.

```
[jerry]$ sed -n '2~2 w junk.txt' books.txt
[jerry]$ cat junk.txt
```

On executing the above code, you get the following result:

```
2) The Two Towers, J. R. R. Tolkien, 352
4) The Fellowship of the Ring, J. R. R. Tolkien, 432
6) A Game of Thrones, George R. R. Martin, 864
```

You can also use comma(,), dollar($), and plus(+) operators with the write command.

In addition to this, SED also supports pattern matching with the write command. Suppose you want to store all the books of individual authors into a separate file. One boring and lengthy way is do it manually, and the smarter way is to use SED.

```
[jerry]$ sed -n -e '/Martin/ w Martin.txt' -e '/Paulo/ w Paulo.txt' -e '/Tolkien/ w
Tolkien.txt' books.txt
```

In the above example, we are matching each line against a pattern and storing the matched line in a particular file. It is very simple. To specify multiple commands, we used **-e** switch of the SED command. Now let use see what each file contains:

```
[jerry]$ cat Martin.txt
```

On executing the above code, you get the following result:

```
1) A Storm of Swords, George R. R. Martin, 1216
6) A Game of Thrones, George R. R. Martin, 864
```

Let us display the file contents.

```
[jerry]$ cat Paulo.txt
```

On executing the above code, you get the following result:

```
3) The Alchemist, Paulo Coelho, 197
5) The Pilgrimage, Paulo Coelho, 288
```

Let us display the file contents.

```
[jerry]$ cat Tolkien.txt
```

On executing the above code, you get the following result:

```
2) The Two Towers, J. R. R. Tolkien, 352
4) The Fellowship of the Ring, J. R. R. Tolkien, 432
```

Excellent! We got the expected result. SED is really an amazing utility.

Append Command

One of the most useful operations of any text editor is to provide append functionality. SED supports this operation through its append command. Given below is the syntax of append:

```
[address]a\
Append text
```

Let us append a new book entry after line number 4. The following example shows how to do it

```
[jerry]$ sed '4 a 7) Adultry, Paulo Coelho, 234' books.txt
```

On executing the above code, you get the following result:

```
1) A Storm of Swords, George R. R. Martin, 1216
2) The Two Towers, J. R. R. Tolkien, 352
3) The Alchemist, Paulo Coelho, 197
4) The Fellowship of the Ring, J. R. R. Tolkien, 432
7) Adultry, Paulo Coelho, 234
5) The Pilgrimage, Paulo Coelho, 288
6) A Game of Thrones, George R. R. Martin, 864
```

In the command section, **4** implies the line number, **a** is the append command, and the remaining part is the text to be appended.

Let us insert a text line at the end of the file. To do this, use **$** as the address. The following example illustrates this:

```
[jerry]$ sed '$ a 7) Adultry, Paulo Coelho, 234' books.txt
```

On executing the above code, you get the following result:

```
1) A Storm of Swords, George R. R. Martin, 1216
2) The Two Towers, J. R. R. Tolkien, 352
3) The Alchemist, Paulo Coelho, 197
4) The Fellowship of the Ring, J. R. R. Tolkien, 432
5) The Pilgrimage, Paulo Coelho, 288
6) A Game of Thrones, George R. R. Martin, 864
7) Adultry, Paulo Coelho, 234
```

Apart from line number, we can also specify an address using textual pattern. For instance, the following example appends text after matching the string **The Alchemist**.

```
[jerry]$ sed '/The Alchemist/ a 7) Adultry, Paulo Coelho, 234' books.txt
```

On executing the above code, you get the following result:

```
1) A Storm of Swords, George R. R. Martin, 1216
2) The Two Towers, J. R. R. Tolkien, 352
3) The Alchemist, Paulo Coelho, 197
7) Adultry, Paulo Coelho, 234
4) The Fellowship of the Ring, J. R. R. Tolkien, 432
5) The Pilgrimage, Paulo Coelho, 288
6) A Game of Thrones, George R. R. Martin, 864
```

Note that if there are multiple patterns matching, then the text is appended after each match. The following example illustrates this scenario.

```
[jerry]$ sed '/The/ a 7) Adultry, Paulo Coelho, 234' books.txt
```

On executing the above code, you get the following result:

```
1) A Storm of Swords, George R. R. Martin, 1216
2) The Two Towers, J. R. R. Tolkien, 352
7) Adultry, Paulo Coelho, 234
3) The Alchemist, Paulo Coelho, 197
7) Adultry, Paulo Coelho, 234
4) The Fellowship of the Ring, J. R. R. Tolkien, 432
7) Adultry, Paulo Coelho, 234
5) The Pilgrimage, Paulo Coelho, 288
7) Adultry, Paulo Coelho, 234
6) A Game of Thrones, George R. R. Martin, 864
```

Change Command

SED provides **change** or **replace** command which is represented by c. This command helps replace an existing line with new text. When line range is provided, all the lines are replaced as a group by a single text line. Given below is the syntax of the change command:

```
[address1[,address2]]c\
Replace text
```

Let us replace the third line with some other text.

```
[jerry]$ sed '3 c 3) Adultry, Paulo Coelho, 324' books.txt
```

On executing the above code, you get the following result:

```
1) A Storm of Swords, George R. R. Martin, 1216
2) The Two Towers, J. R. R. Tolkien, 352
3) Adultry, Paulo Coelho, 324
4) The Fellowship of the Ring, J. R. R. Tolkien, 432
5) The Pilgrimage, Paulo Coelho, 288
6) A Game of Thrones, George R. R. Martin, 864
```

SED also accepts patterns as an address. In the following example, a line is replaced when the pattern match succeeds.

```
[jerry]$ sed '/The Alchemist/ c 3) Adultry, Paulo Coelho, 324' books.txt
```

On executing the above code, you get the following result:

```
1) A Storm of Swords, George R. R. Martin, 1216
2) The Two Towers, J. R. R. Tolkien, 352
3) Adultry, Paulo Coelho, 324
4) The Fellowship of the Ring, J. R. R. Tolkien, 432
5) The Pilgrimage, Paulo Coelho, 288
6) A Game of Thrones, George R. R. Martin, 864
```

SED also allows replacement of multiple lines with a single line. The following example removes lines from fourth through sixth and replaces them with new text.

```
[jerry]$ sed '4, 6 c 4) Adultry, Paulo Coelho, 324' books.txt
```

On executing the above code, you get the following result:

```
1) A Storm of Swords, George R. R. Martin, 1216
2) The Two Towers, J. R. R. Tolkien, 352
3) The Alchemist, Paulo Coelho, 197
4) Adultry, Paulo Coelho, 324
```

Insert Command

The insert command works much in the same way as append does. The only difference is that it inserts a line before a specific position. Given below is the syntax of the insert command:

```
[address]i\
Insert text
```

Let us understand the insert command with some examples. The following command inserts a new entry before the fourth line.

```
[jerry]$ sed '4 i 7) Adultry, Paulo Coelho, 324' books.txt
```

On executing the above code, you get the following result:

```
1) A Storm of Swords, George R. R. Martin, 1216
2) The Two Towers, J. R. R. Tolkien, 352
3) The Alchemist, Paulo Coelho, 197
7) Adultry, Paulo Coelho, 324
4) The Fellowship of the Ring, J. R. R. Tolkien, 432
5) The Pilgrimage, Paulo Coelho, 288
6) A Game of Thrones, George R. R. Martin, 864
```

In the above example, **4** is the location number, **i** implies the insert command, and the remaining part is the text to be inserted.

To insert text at the start of a file, provide the line address as **1**. The following command illustrates this:

```
[jerry]$ sed '1 i 7) Adultry, Paulo Coelho, 324' books.txt
```

On executing the above code, you get the following result:

```
7) Adultry, Paulo Coelho, 324
```

```
1) A Storm of Swords, George R. R. Martin, 1216
2) The Two Towers, J. R. R. Tolkien, 352
3) The Alchemist, Paulo Coelho, 197
4) The Fellowship of the Ring, J. R. R. Tolkien, 432
5) The Pilgrimage, Paulo Coelho, 288
6) A Game of Thrones, George R. R. Martin, 864
```

Additionally, we can insert multiple lines. The following command inserts two lines before the last line.

```
[jerry]$ sed '$ i 7) Adultry, Paulo Coelho, 324
```

On executing the above code, you get the following result:

```
8) Eleven Minutes, Paulo Coelho, 304' books.txt
1) A Storm of Swords, George R. R. Martin, 1216
2) The Two Towers, J. R. R. Tolkien, 352
3) The Alchemist, Paulo Coelho, 197
4) The Fellowship of the Ring, J. R. R. Tolkien, 432
5) The Pilgrimage,Paulo Coelho, 288
7) Adultry, Paulo Coelho, 324
8) Eleven Minutes, Paulo Coelho, 304
6) A Game of Thrones, George R. R. Martin, 864
```

Note that the entries to be inserted are entered on separate lines and delimited by the backslash(\) character.

Translate Command

SED provides a command to translate characters and it is represented as **y**. It transforms the characters by position. Given below is the syntax of the translate command:

```
[address1[,address2]]y/list-1/list-2/
```

Note that translation is based on the position of the character from **list 1** to the character in the same position in **list 2** and both lists must be explicit character lists. Regular expressions and character classes are unsupported. Additionally, the size of **list 1** and **list 2** must be same.

The following example converts Arabic numbers to Roman numbers.

```
[jerry]$ echo "1 5 15 20" | sed 'y/151520/IVXVXX/'
```

On executing the above code, you get the following result:

```
I V IV XX
```

l command

Can you differentiate between words separated by spaces and words separated by tab characters only by looking at them? Certainly not. But SED can do this for you. SED uses the **l** command to display hidden characters in the text. For example, tab character with **\t** and End-Of-Line with **$** character. Given below is the syntax of the **l** command.

```
[address1[,address2]]l
[address1[,address2]]l [len]
```

Let us create a file with tab characters for demonstration. For simplicity, we are going to use the same file, just by replacing spaces with tabs. Wait! But how to do that — by opening the file in a text editor and replacing each space with tab? Certainly not! We can make use of SED commands for that.

```
[jerry]$ sed 's/ /\t/g' books.txt > junk.txt
```

Now let us display the hidden characters by using the **l** command:

```
[jerry]$ sed -n 'l' junk.txt
```

On executing the above code, you get the following result:

```
1)\tA\tStorm\tof\tSwords,George\tR.\tR.\tMartin,1216$
2)\tThe\tTwo\tTowers,J.\tR.\tR.\tTolkien,352$
3)\tThe\tAlchemist,Paulo\tCoelho,197$
4)\tThe\tFellowship\tof\tthe\tRing,J.\tR.\tR.\tTolkien,432$
5)\tThe\tPilgrimage,Paulo\tCoelho,288$
6)\tA\tGame\tof\tThrones,George\tR.\tR.\tMartin\t,864$
```

Like other SED commands, it also accepts line numbers and patterns as an address. You can try it yourselves.

Let us take a close look at another interesting feature of SED. We can instruct the SED to perform line wrapping after a certain number of characters. The following example wraps lines after 25 characters.

```
[jerry]$ sed -n 'l 25' books.txt
```

On executing the above code, you get the following result:

```
1) A Storm of Swords,Geo\
rge R. R. Martin,1216$
2) The Two Towers,J. R. \
R. Tolkien,352$
3) The Alchemist,Paulo C\
oelho,197$
4) The Fellowship of the\
 Ring,J. R. R. Tolkien,4\
32$
5) The Pilgrimage,Paulo \
Coelho,288$
6) A Game of Thrones,Geo\
rge R. R. Martin ,864$
```

Note that in the above example, wrap limit is provided after l command. In this case, it is 25 characters. This option is GNU specific and may not work with other variants of the SED.

A wrap limit of 0 means never break the line unless there is a new line character. The following simple command illustrates this.

```
[jerry]$ sed -n 'l 0' books.txt
```

On executing the above code, you get the following result:

```
1) A Storm of Swords,George R. R. Martin,1216$
2) The Two Towers,J. R. R. Tolkien,352$
3) The Alchemist,Paulo Coelho,197$
4) The Fellowship of the Ring,J. R. R. Tolkien,432$
5) The Pilgrimage,Paulo Coelho,288$
6) A Game of Thrones,George R. R. Martin,864$
```

Quit Command

Quit command instructs the SED to quit the current execution flow. It is represented by the **q** command. Given below is the syntax of the quit command:

```
[address]q
[address]q [value]
```

Note that the quit command does not accept range of addresses, it only supports a single address. By default, SED follows read, execute, and repeat workflow; but when the quit command is encountered, it simply stops the current execution.

Let us print the first 3 lines from the file.

```
[jerry]$ sed '3 q' books.txt
```

On executing the above code, you get the following result:

```
1) A Storm of Swords, George R. R. Martin, 1216
2) The Two Towers, J. R. R. Tolkien, 352
3) The Alchemist, Paulo Coelho, 197
```

In addition to line number, we can also use textual patterns. The following command quits when pattern match succeeds.

```
[jerry]$ sed '/The Alchemist/ q' books.txt
```

On executing the above code, you get the following result:

```
1) A Storm of Swords, George R. R. Martin, 1216
2) The Two Towers, J. R. R. Tolkien, 352
3) The Alchemist, Paulo Coelho, 197
```

In addition to this, SED can also accept a **value** which can be used as the exit status. The following command shows its exit status as 100.

```
[jerry]$ sed '/The Alchemist/ q 100' books.txt
```

On executing the above code, you get the following result:

```
1) A Storm of Swords, George R. R. Martin, 1216
2) The Two Towers, J. R. R. Tolkien, 352
3) The Alchemist, Paulo Coelho, 197
```

Now let us verify the exit status.

```
[jerry]$ echo $?
```

On executing the above code, you get the following result:

```
100
```

Read Command

We can instruct the SED to read the contents of a file and display them when a specific condition matches. The command is represented by the alphabet **r**. Given below is the syntax of the read command.

```
[address]r file
```

Note that there must be exactly one space between the **r** command and the file name.

Let us understand it with a simple example. Create a sample file called **junk.txt**.

```
[jerry]$ echo "This is junk text." > junk.txt
```

The following command instructs the SED to read the contents of **junk.txt** and insert them after the third line.

```
[jerry]$ sed '3 r junk.txt' books.txt
```

On executing the above code, you get the following result:

```
1) A Storm of Swords, George R. R. Martin, 1216
2) The Two Towers, J. R. R. Tolkien, 352
3) The Alchemist, Paulo Coelho, 197
This is junk text.
4) The Fellowship of the Ring, J. R. R. Tolkien, 432
5) The Pilgrimage, Paulo Coelho, 288
6) A Game of Thrones, George R. R. Martin, 864
```

In the above example, 3 implies the line address, **r** is the command name, and **junk.txt** is the file name the contents of which are to be displayed. Additionally, the GNU SED also accepts a range of addresses. For instance, the following command inserts the contents of **junk.txt** after the third, fourth, and fifth lines.

```
[jerry]$ sed '3, 5 r junk.txt' books.txt
```

On executing the above code, you get the following result:

```
1) A Storm of Swords, George R. R. Martin, 1216
2) The Two Towers, J. R. R. Tolkien, 352
3) The Alchemist, Paulo Coelho, 197
This is junk text.
4) The Fellowship of the Ring, J. R. R. Tolkien, 432
This is junk text.
5) The Pilgrimage, Paulo Coelho, 288
This is junk text.
6) A Game of Thrones, George R. R. Martin, 864
```

Like other SED commands, the read command also accepts pattern as an address. For instance, the following command inserts the contents of **junk.txt** when the pattern match succeeds.

```
[jerry]$ sed '/Paulo/ r junk.txt' books.txt
```

On executing the above code, you get the following result:

```
1) A Storm of Swords, George R. R. Martin, 1216
2) The Two Towers, J. R. R. Tolkien, 352
3) The Alchemist, Paulo Coelho, 197
This is junk text.
4) The Fellowship of the Ring, J. R. R. Tolkien, 432
5) The Pilgrimage, Paulo Coelho, 288
This is junk text.
6) A Game of Thrones, George R. R. Martin, 864
```

Execute Command

We can execute external commands from SED using the **execute** command. It is represented by **e**. Given below is the syntax of the execute command.

```
[address1[,address2]]e [command]
```

Let us illustrate the execute command with a simple example. The following SED command executes the UNIX **date** command before the third line.

```
[jerry]$ sed '3 e date' books.txt
```

On executing the above code, you get the following result:

```
1) A Storm of Swords, George R. R. Martin, 1216
2) The Two Towers, J. R. R. Tolkien, 352
Sun Sep  7 18:04:49 IST 2014
3) The Alchemist, Paulo Coelho, 197
4) The Fellowship of the Ring, J. R. R. Tolkien, 432
```

```
5) The Pilgrimage, Paulo Coelho, 288
6) A Game of Thrones, George R. R. Martin, 864
```

Like other commands, it also accepts patterns as an address. For example, the following example executes **date** command when a pattern match succeeds. Note that after each pattern match, first the command is executed and then the contents of the pattern buffer are displayed.

```
[jerry]$ sed '/Paulo/ e date' books.txt
```

On executing the above code, you get the following result:

```
1) A Storm of Swords, George R. R. Martin, 1216
2) The Two Towers, J. R. R. Tolkien, 352
Sun Sep  7 18:06:04 IST 2014
3) The Alchemist, Paulo Coelho, 197
4) The Fellowship of the Ring, J. R. R. Tolkien, 432
Sun Sep  7 18:06:04 IST 2014
5) The Pilgrimage, Paulo Coelho, 288
6) A Game of Thrones, George R. R. Martin, 864
```

If you observe the syntax of the **e** command carefully, you will notice that **command** is optional. When no command is provided after **e,** it treats the contents of the pattern buffer as an external command. To illustrate this, let us create a commands.txt file with a few simple commands.

```
[jerry]$ echo -e "date\ncal\nuname" > commands.txt
[jerry]$ cat commands.txt
```

On executing the above code, you get the following result:

```
date
cal
uname
```

Commands from the file are self-explanatory. In the absence of **command** after **e,** SED executes all these commands one by one. The following simple example illustrates this.

```
[jerry]$ sed 'e' commands.txt
```

On executing the above code, you get the following result:

```
Sun Sep  7 18:14:20 IST 2014
   September 2014
Su Mo Tu We Th Fr Sa
    1  2  3  4  5  6
 7  8  9 10 11 12 13
14 15 16 17 18 19 20
21 22 23 24 25 26 27
28 29 30

Linux
```

Like other SED commands, the execute command also accepts all valid ranges of addresses.

Miscellaneous Commands

By default, SED operates on single line, however it can operate on multiple lines as well. Multi-line commands are denoted by uppercase letters. For example, unlike the **n** command, the **N** command does not clear and print the pattern space. Instead, it adds a newline (\n) at the end of the current pattern space and appends the next line from the input-file to the current pattern space and continues with the SED's standard flow by executing the rest of the SED commands. Given below is the syntax of the **N** command.

```
[address1[,address2]]N
```

Let us print a comma-separated list of book titles and their respective authors. The following example illustrates this.

```
[jerry]$ sed 'N; s/\n/, /g' books.txt
```

On executing the above code, you get the following result:

```
A Storm of Swords, George R. R. Martin
The Two Towers, J. R. R. Tolkien
The Alchemist, Paulo Coelho
The Fellowship of the Ring, J. R. R. Tolkien
The Pilgrimage, Paulo Coelho
A Game of Thrones, George R. R. Martin
```

Let us understand how the above example works. The **N** command reads the first line, i.e.,A Storm of Swords into the pattern buffer and appends \n followed by the next line. The pattern space now contains A Storm of Swords**\n**George R. R. Martin. In the next step, we are replacing the newline with a comma.

Like **p** command, we have a **P** command to print the first part (up to embedded newline) of the multi-line pattern space created by the **N** command. Given below is the syntax of the **P** command which is similar to the **p** command.

```
[address1[,address2]]P
```

In the previous example, we saw that the **N** command creates a newline- separated list of book titles and their authors. Let us print only the first part of it, i.e., only the titles of the book. The following command illustrates this.

```
[jerry]$ sed -n 'N;P' books.txt
```

On executing the above code, you get the following result:

```
A Storm of Swords
The Two Towers
The Alchemist
The Fellowship of the Ring
The Pilgrimage
A Game of Thrones
```

Note that in the absence of **N**, it behaves same as the **p** command. The following simple command illustrates this scenario.

```
[jerry]$ sed -n 'P' books.txt
```

On executing the above code, you get the following result:

```
A Storm of Swords
George R. R. Martin
The Two Towers
J. R. R. Tolkien
The Alchemist
Paulo Coelho
The Fellowship of the Ring
J. R. R. Tolkien
The Pilgrimage
Paulo Coelho
A Game of Thrones
George R. R. Martin
```

In addition to this, SED also provides a **v** command which checks for version. If the provided version is greater than the installed SED version, then the command execution fails. Note that this option is GNU specific and may not work with other variants of SED. Given below is the syntax of the **v** command.

```
[address1[,address2]]v [version]
```

First, find out the current version of SED.

```
[jerry]$ sed --version
```

On executing the above code, you get the following result:

```
sed (GNU sed) 4.2.2
```

In the following example, the SED version is greater than version 4.2.2, hence the SED command aborts its execution.

```
[jerry]$ sed 'v 4.2.3' books.txt
```

On executing the above code, you get the following result:

```
sed: -e expression #1, char 7: expected newer version of sed
```

But if the provided version is lesser than or equal to version 4.2.2, then the command works as expected.

```
[jerry]$ sed 'v 4.2.2' books.txt
```

On executing the above code, you get the following result:

```
A Storm of Swords
George R. R. Martin
The Two Towers
```

```
J. R. R. Tolkien
The Alchemist
Paulo Coelho
The Fellowship of the Ring
J. R. R. Tolkien
The Pilgrimage
Paulo Coelho
A Game of Thrones George R. R. Martin
```