**SED(*stream editor) search tool***

* **grep**. A fast and powerful pattern search tool that can be easily combined with other filters to find results and customize the display, even if the main aim is to search for matches. Its main usage consists in narrowing search results by forcing the match with the given pattern.
* **awk**. A loosely typed programming language for stream processing, where the basic unit is the String (intended as an array of characters) that can be i. matched, ii. substituted and iii. worked around; most of the times, it is no really needed to combine awk with other filters, since its reporting capabilities are very powerful (the [*printf*](http://www.codingunit.com/printf-format-specifiers-format-conversions-and-formatted-output)built-in function allows to format the output text as in C). Its main usage consists in perform *fine-grained*(variables can be defined and modified incrementally) and *programmatic manipulations* (flow control statements) to the input stream.
* **sed**. A more advanced, fast stream editor, able to search for a pattern and apply the given transformations and/or commands; still **easy to combine in sophisticated filters**, but serving a different aim: **modifying the text in the stream**. Its main usage consists in editing in-memory a stream according to the given pattern.

### 1. Viewing a range of lines of a document

Tools such as [head and tail](https://www.tecmint.com/view-contents-of-file-in-linux/) allow us to view the bottom or the top of a file. What if we need to view a section in the middle? The following sed one-liner will return lines **5** through **10** from **myfile.txt**:

* # sed -n '5,10p' /etc/passwd

### 2. Viewing the entire file except a given range

On the other hand, it’s possible that you want to print the entire file except a certain range. To exclude lines **20** through **35** from **myfile.txt**, do:

# sed '2,7d' myfile.txt

### 3. Viewing non-consecutive lines and ranges

It’s possible that you’re interested in set of non-consecutive lines, or in more than one range. Let’s display lines **5-7** and **10-13** from **myfile.txt**:

# sed -n -e '5,7p' -e '10,13p' myfile.txt

4. Replacing words or characters (basic substitution)

To replace every instance of the word **raunak** with **garg** in myfile.txt, do:

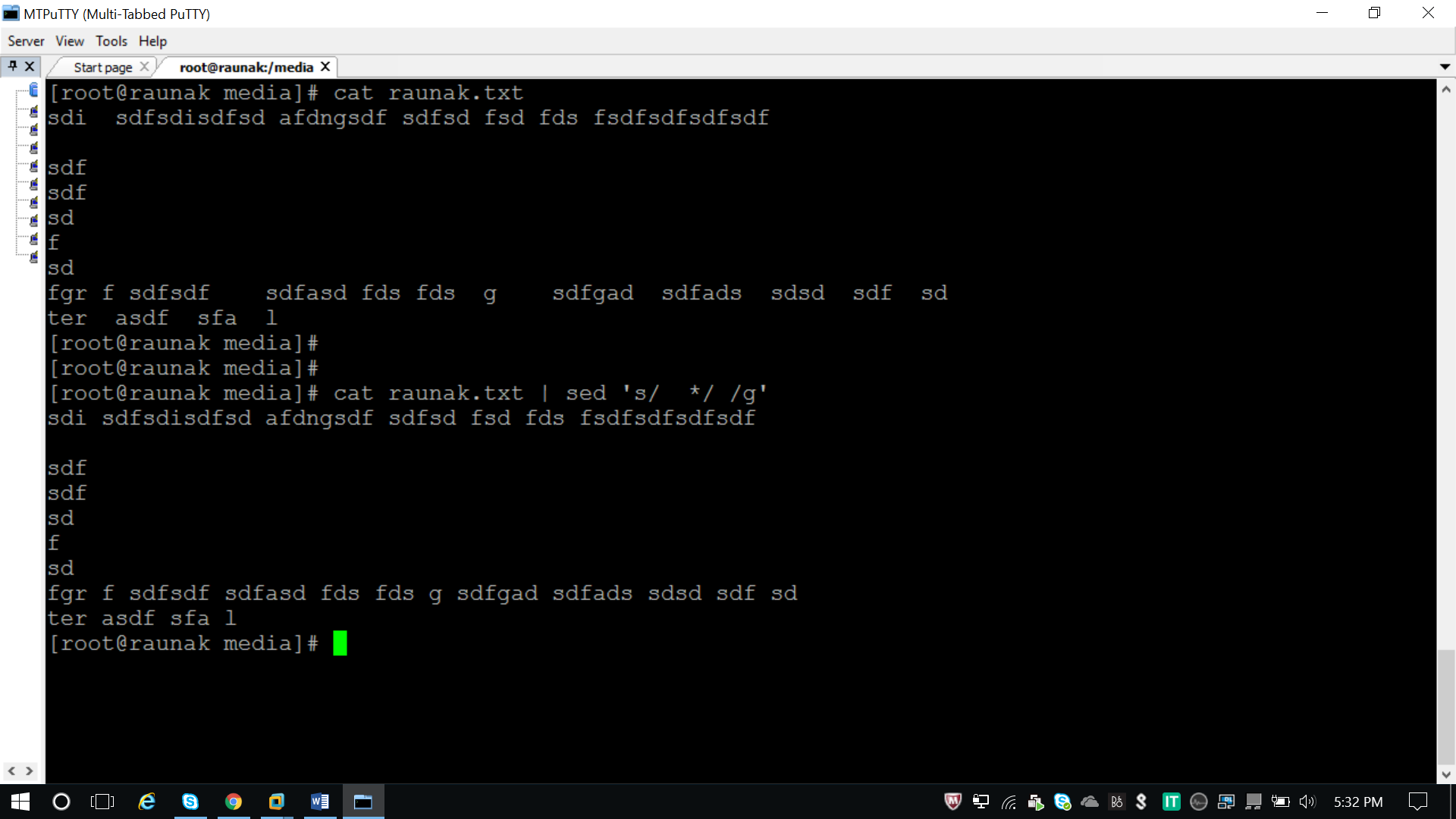
# sed 's/raunak/garg/g' myfile.txt

Additionally, you may want to consider using **gi** instead of **g** in order to ignore character case:

# sed 's/version/story/gi' myfile.txt

To replace multiple blank spaces with a single space, we will use the output of **raunak.txt** and a pipeline:

# cat raunak.txt | sed 's/ \*/ /g' Comp



5. Replacing words or characters inside a range

If you’re interested in replacing words only within a line range (3 through 7, for example), you can do:

# sed '3,7 s/Raunak/garg/g' myfile.txt

Of course, you can indicate a single line through its corresponding number instead of a range.

6. Using regular expressions (advanced substitution) – I

Sometimes configuration files are loaded with comments. While this is certainly useful, it may be helpful to display only the configuration directives sometimes if you want to view them all at a glance.

To remove empty lines or those beginning with **#** from the myfile.txt file, do:

# sed '/^#\|^$\| \*#/d' myfile.txt

The caret sign followed by the number sign **(^#)** indicates the beginning of a line, whereas **^$**represents blank lines. The vertical bars indicate boolean operations, whereas the backward slash is used to escape the vertical bars.

In this particular case, the Apache configuration file has lines with **#’s** not at the beginning of some lines, so **\*#** is used to remove those as well.

7. Using regular expressions (advanced substitution) – II

To replace a word beginning with uppercase or lowercase with another word, we can also use sed. To illustrate, let’s replace the word zip or Zip with rar in myfile.txt:

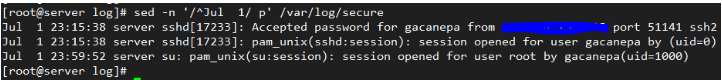
# sed 's/[Zz]ip/rar/g' myfile.txt

8. Viewing lines containing with a given pattern

Another use of sed consists in printing the lines from a file that match a given regular expression. For example, we may be interested in viewing the authorization and authentication activities that took place on July 2, as per the /var/log/secure log in a CentOS 7 server.

In this case, the pattern to search for is Jul 2 at the beginning of each line:

# sed -n '/^Jul 1/ p' /var/log/secure

[](https://www.tecmint.com/wp-content/uploads/2016/07/View-Logs-of-Particular-Date.png)

*View Logs (Lines) of Particular Date*

9. Inserting spaces in files

With sed, we can also insert spaces (blank lines) for each non-empty line in a file. To insert one blank line every other line in LICENSE, a plain text file, do:

# sed G myfile.txt

To insert two blank lines, do:

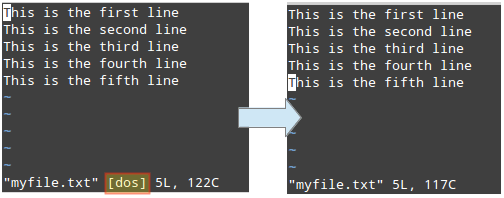
# sed 'G;G' myfile.txt

### 10. Emulating dos2unix with inline editing

The **dos2unix** program converts plain text files from Windows/Mac formatting to Unix/Linux, removing hidden newline characters inserted by some text editors used in those platforms. If it is not installed in your Linux system, you can mimic its functionality with **sed** instead of installing it.

In the image at the left we can see several DOS newline characters **(^M)**, which were later removed with:

# sed -i 's/\r//' myfile.txt

[](https://www.tecmint.com/wp-content/uploads/2016/07/Covert-Text-Files-from-Windows-to-Linux.png)

*Covert Text Files from Windows to Linux*

Please note that the **-i** option indicate in-place editing. Then changes will not be returned to the screen, but will be saved to the file.

**Note**: You can insert DOS newline characters while [editing a file in vim editor](https://www.tecmint.com/learn-vi-and-vim-editor-tips-and-tricks-in-linux/) with **Ctrl+V** and **Ctrl+M**.

### 11. In-place editing and backing up original file

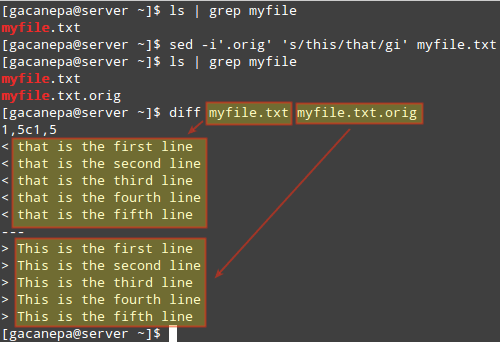
In the previous tip we used **sed** to modify a file but did not save the original file. Sometimes it’s a good idea to save a backup copy of the original file just in case.

To do that, indicate a suffix following the **-i** option (inside single quotes) to be used to rename the original file.

In the following example we will replace all instances of **this** or **This** (ignoring case) with that in **myfile.txt**, and we will save the original file as **myfile.txt.orig**.

Finally, we will use [diff utility](https://www.tecmint.com/best-linux-file-diff-tools-comparison/) to identify the differences between both files:

# sed -i'.orig' 's/this/that/gi' myfile.txt

[](https://www.tecmint.com/wp-content/uploads/2016/07/Sed-Edit-and-Backup-Original-File.png)

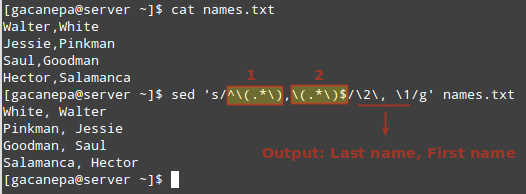
*Use sed to Edit and Backup Original File*

### 12. Switching pairs of words

Let’s suppose you have a file containing full names in the format **First name**, **Last name**. To adequately process the file, you may want to switch **Last name** and **First name**.

We can do that with **sed** fairly easily:

# sed 's/^\(.\*\),\(.\*\)$/\2\, \1/g' names.txt

[](https://www.tecmint.com/wp-content/uploads/2016/07/Switch-Words-in-File.png)

*Switch Words in File*

In the image above we can see that parentheses, being special characters, need to be escaped, as do the numbers **1** and **2**.

These numbers represent the highlighted regular expressions (which need to appear inside parentheses):

1. 1 represents the beginning of each line up to the comma.
2. 2 is a placeholder for everything that is right of the comma to the end of the line.

The desired output is indicated in the format **SecondColumn** (**Last name**) + **comma** + **space** + **FirstColumn** (**First name**). Feel free to change it to whatever you wish.

### 13. Replacing words only if a separate match is found

Sometimes replacing all instances of a given word, or a random few, is not precisely what we need. Perhaps we need to perform the replacement if a separate match is found.

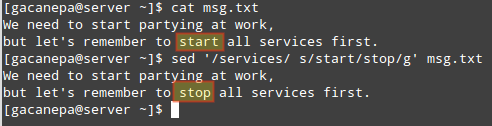
For example, we may want to replace **start** with **stop** only if the word services is found in the same line. In that scenario, here’s what will happen:

We need to start partying at work,

but let’s remember to start all services first.

In the first line, **start** will not be replaced with **stop** since the word services does not appear in that line, as opposed to the second line.

# sed '/services/ s/start/stop/g' msg.txt

[](https://www.tecmint.com/wp-content/uploads/2016/07/Replace-Words-in-File.png)

*Replace Words in File*

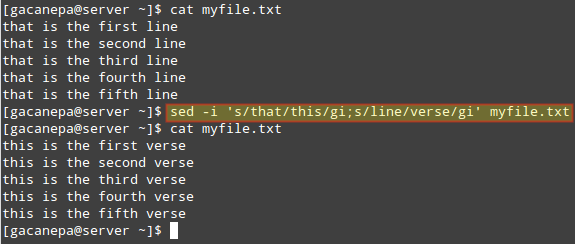
### 14. Performing two or more substitutions at once

You can combine two or more substitutions one single **sed** command. Let’s replace the words that and line in **myfile.txt** with this and verse, respectively.

Note how this can be done by using an ordinary **sed** substitution command followed by a semicolon and a second substitution command:

# sed -i 's/that/this/gi;s/line/verse/gi' myfile.txt

This tip is illustrated in the following image:

[](https://www.tecmint.com/wp-content/uploads/2016/07/Replace-Each-Word-with-New-Word.png)

*Replace Multiple Words in File*

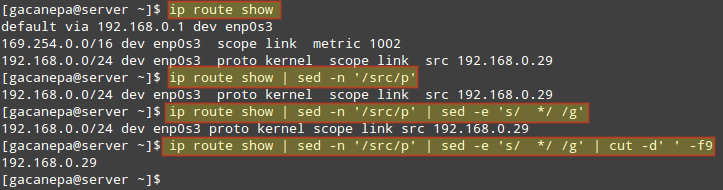
### 15. Combining sed and other commands

Of course, **sed** can be combined with other tools in order to create more powerful commands. For example, let’s use the example given in **TIP #4** and extract our IP address from the output of the **ip route** command.

We will begin by printing only the line where the word **src** is. Then we will convert multiple spaces into a single one. Finally, we will cut the **9th** field (considering a single space as field separator), which is where the IP address is:

# ip route show | sed -n '/src/p' | sed -e 's/ \*/ /g' | cut -d' ' -f9

The image below illustrates each step of the above command:

[](https://www.tecmint.com/wp-content/uploads/2016/07/Combine-sed-with-Other-Commands.png)