**Filtering commands In Linux**

Linux Filter commands accept input data from **stdin** (standard input) and produce output on **stdout** (standard output). It transforms plain-text data into a meaningful way and can be used with pipes to perform higher operations.

These filters are very small programs that are designed for a specific function which can be used as building blocks.

Linux Filter Commands

1. [cat](https://www.javatpoint.com/linux-cat-filters)
2. [cut](https://www.javatpoint.com/linux-cut)
3. [grep](https://www.javatpoint.com/linux-grep)
4. [comm](https://www.javatpoint.com/linux-comm)
5. [sed](https://www.javatpoint.com/linux-sed)
6. [tee](https://www.javatpoint.com/linux-tee)
7. [tr](https://www.javatpoint.com/linux-tr)
8. [uniq](https://www.javatpoint.com/linux-uniq)
9. [wc](https://www.javatpoint.com/linux-wc)
10. [od](https://www.javatpoint.com/linux-od)
11. [sort](https://www.javatpoint.com/linux-sort)
12. [gzip](https://www.javatpoint.com/linux-gzip)

Linux Cat Filters

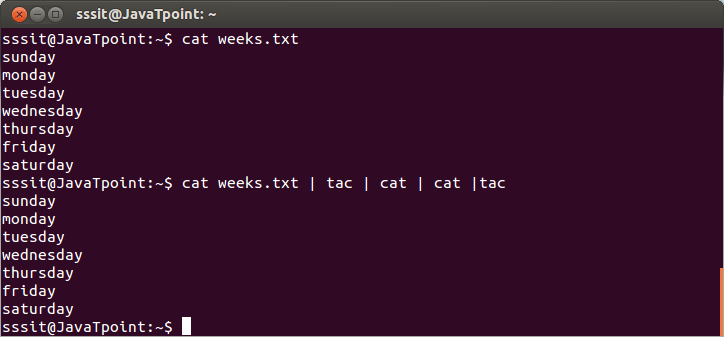
When cat command is used inside pipes, it does nothing except moving stdin to stout.

**Syntax:**

1. cat **<fileName>** | cat or tac |  cat or tac |. . .

**Example:**

1. cat weeks.txt | tac | cat | cat | tac



Look at the above snapshot, output of one 'cat' or 'tac' command is passing onto another as input.

# Linux cut Command

Linux cut command is useful for selecting a specific column of a file. It is used to cut a specific sections by byte position, character, and field and writes them to standard output. It cuts a line and extracts the text data. It is necessary to pass an argument with it; otherwise, it will throw an error message.

To cut a specific section, it is necessary to specify the delimiter. A delimiter will decide how the sections are separated in a text file. Delimiters can be a space (' '), a hyphen (-), a slash (/), or anything else. After '-f' option, the column number is mentioned.

### Syntax:

1. cut OPTION... [FILE]...

### Options:

The following command line options are used by the cut command to make it more specific:

**-b, --bytes=LIST:** It is used to cut a specific section by bytes.

**-c, --characters=LIST:** It is used to select the specified characters.

**-d, --delimiter=DELIM:** It is used to cut a specific section by a delimiter.

**-f, --fields=LIST:** It is used to select the specific fields. It also prints any line that does not contain any delimiter character, unless the -s option is specified.

**-n:** It is used to ignore any option.

**--complement:** It is used to complement the set of selected bytes, characters or fields

**-s, --only-delimited:** It is used to not print lines that do not have delimiters.

**--output-delimiter=STRING:** This option is specified to use a STRING as an output delimiter; The default is to use "input delimiter".

**-z, --zero-terminated:** It is used if line delimiter is NUL, not newline.

**--help:** It is used to display the help manual.

**--version:** It is used to display the version information.

## Examples of the cut command

Let's see the following examples of the cut command:

* [Cut by using Hyphen as delimiter](https://www.javatpoint.com/linux-cut#Hyphen)
* [Cut by using Space as delimiter](https://www.javatpoint.com/linux-cut#Space)
* [Cut by byte position](https://www.javatpoint.com/linux-cut#byte)
* [Cut by character](https://www.javatpoint.com/linux-cut#Character)
* [Cut by complement pattern](https://www.javatpoint.com/linux-cut#Complement)

### Using Hyphen (-) As Delimiter

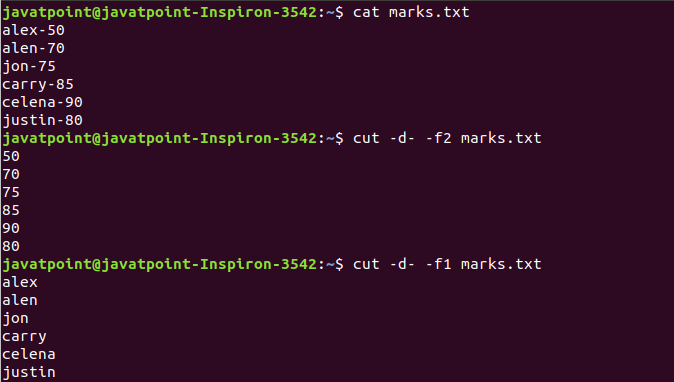
To cut by using the hyphen (-) as the delimiter, execute the below command:

1. cut -d- -f(columnNumber) **<fileName>**

Consider the following commands:

1. cut -d- -f2 marks.txt
2. cut -d- -f1 marks.txt

from the above commands, the output will be trimmed from hyphen (-). Consider the below output:



As we can see from the above output, our delimiter is the hyphen (-); hence we have used (-) after (-d). Command "cut -d- -f1 marks.txt" displays column 1 and command "cut -d- -f2 marks.txt" displays column 2.

### Using Space As Delimiter

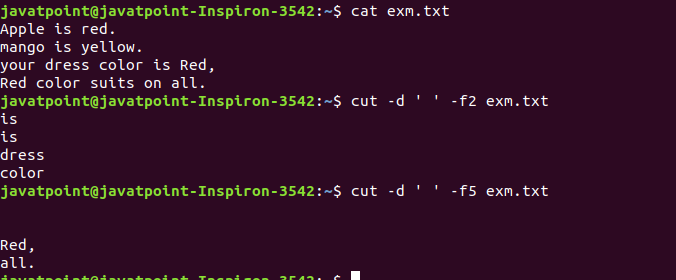
If we want to use space as a delimiter, then we have to quote the space (' ') with the cut command. To cut the output by using space as delimiter, execute the command as follows:

1. cut -d ' ' -f(columnNumber) **<fileName>**

Consider the following commands:

1. cut -d ' ' -f2 exm.txt
2. cut -d ' ' -f5 exm.txt

From the above commands, the output will be trimmed after space for the specified column. The above commands will produce the output as follows:



From the above output, our delimiter is space; hence we have used (' ') after (-d). Command "cut -d ' ' -f2 exm.txt" displays column 2, command "cut -d ' ' -f5 exm.txt" displays column 5.

### Cut by byte

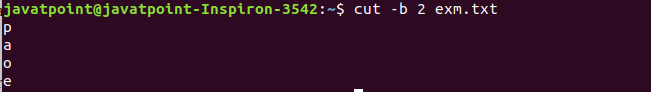
The '-b' option is used to cut a section of line by byte. To cut a file by its byte position, execute the command as follows:

1. cut -b **<byte** number**>** **<file** name**>**

Consider the below command:

1. cut -b 2 exm.txt

The above command will cut the line by a specified byte position. Consider the below output:



### Cut by Character

The '-c' option is used to cut a specific section by character. However, these character arguments can be a number or a range of numbers, a list of comma-separated numbers, or any other character.

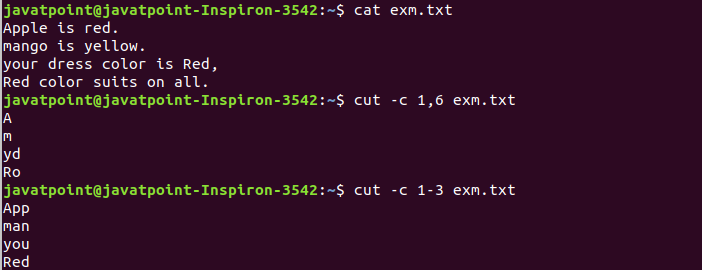
To cut by specified character, execute the command as follows:

1. cut -c **<** **characters>** **<file** name**>**

Consider the below commands:

1. cut -c 1,6 exm.txt
2. cut -c 1-3 exm.txt

The above commands will cut the line by the specified characters. Consider the below output:



From the above output, we can see the first command is cutting the first and sixth character from each line, and the second command is cutting the first to the third character from each line.

### Cut by Complement Pattern

The '--complement' option is used to cut by the complement. This option is supported by the BSD version of the cut. To cut by the complement pattern, execute the command as follows:

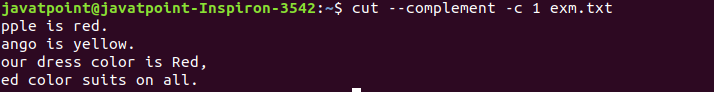
1. cut --complement **<** **complement** pattern**>** **<file** name**>**

The compliment pattern can be a list of bytes, characters, or fields.

Consider the below command:

1. cut --complement -c 1 exm.txt

The above command will cut the file by the first character. Consider the below output:



# Linux grep

The 'grep' command stands for **"global regular expression print"**. grep command filters the content of a file which makes our search easy.

**grep with pipe**

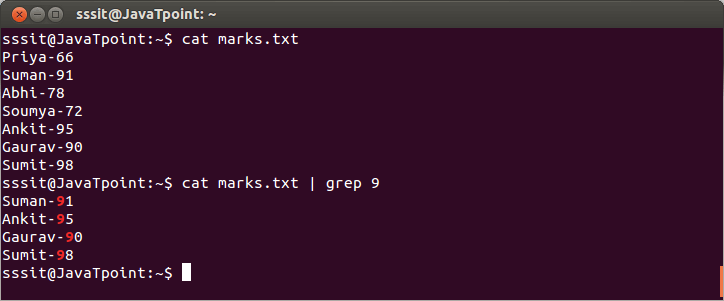
The 'grep' command is generally used with pipe **(|)**.

**Syntax:**

1. command | grep **<searchWord>**

**Example:**

1. cat marks.txt | grep 9



Look at the above snapshot, grep command filters all the data containing '9'.

grep without pipe

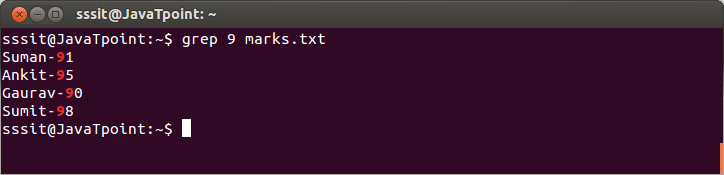
It can be used without pipe also.

**Syntax:**

1. grep **<searchWord>** **<file** name**>**

**Example:**

1. grep 9 marks.txt



Look at the above snapshot, grep command do the same work as earlier example but without pipe.

grep options

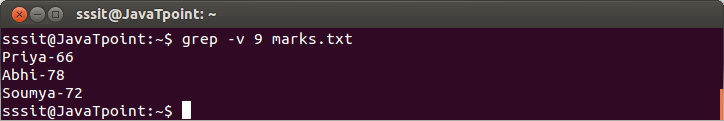
* **grep -vM**: The 'grep -v' command displays lines not matching to the specified word.

**Syntax:**

* 1. grep -v **<searchWord>** **<fileName>**

**Example:**

* 1. grep -v 9 marks.txt



Look at the above snapshot, command **"grep -v 9 marks.txt"** displays lines hwich don't contain our search word '9'.

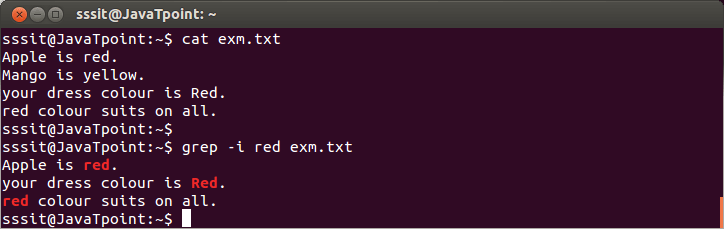
* **grep -i**: The 'grep -i' command filters output in a case-insensitive way.

**Syntax:**

* 1. grep -i **<searchWord>** **<fileName>**

**Example:**

* 1. grep -i red exm.txt



Look at the above snapshot, command **"grep -i red exm.txt"** displays all lines containing 'red' whether in upper case or lower case.

* **grep -A/ grep -B/ grep -C**

grep -A command is used to display the **line after the result**.

grep -B command is used to display the **line before the result**.

grep -C command is used to display the **line after and line before** the result.

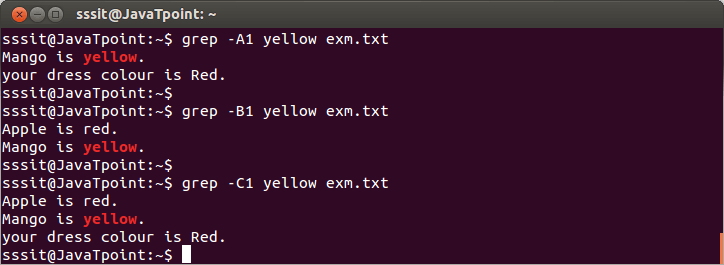
You can use (A1, A2, A3.....)(B1, B2, B3....)(C1, C2, C3....) to display any number of lines.

**Syntax:**

* 1. grep -A**<lineNumber>** **<searchWord>** **<fileName>**
  2. grep -B**<lineNumber>** **<searchWord>** **<fileName>**
  3. grep -C**<lineNumber>** **<searchWord>** **<fileName>**

**Example:**

* 1. grep -A1 yellow exm.txt
  2. grep -B1 yellow exm.txt
  3. grep -C1 yellow exm.txt



Look at the above snapshot, command **"grep -A1 yellow exm.txt"** displays searched line with next succeeding line, command**"grep -B1 yellow exm.txt"** displays searched line with one preceding line and command **"grep -C1 yellow exm.txt"** displays searched line with one preceding and succeeding line.

Linux comm

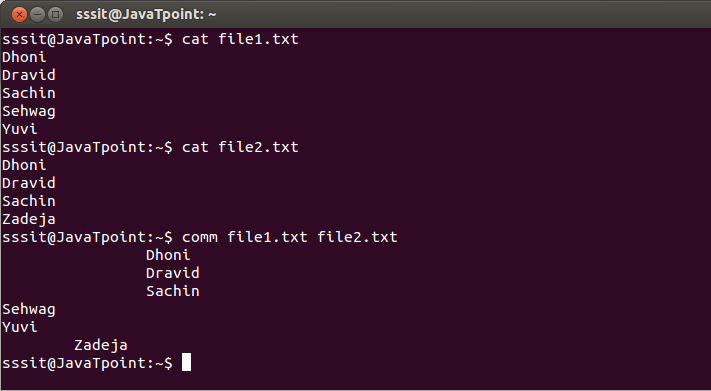
The 'comm' command compares two files or streams. By default, 'comm' will always display **three columns**. First column indicates non-matching items of first file, second column indicates non-matching items of second file, and third column indicates matching items of both the files. Both the files has to be in sorted order for 'comm' command to be executed.

**Syntax:**

1. comm **<file1>** **<file2>**

**Example:**

1. comm file1.txt file2.txt



Look at the above snapshot,

First column displays non-matching items of first file 'file1.txt' (Sehwag, Yuvi)

Second column displays non-matching items of second file 'file2.txt' (Zadeja)

Third column indicates matching items of both the files (Dhoni, Dravid, Sachin)

To Display Single Column

If you want to output a single column, you have to specify number of the columns which are not to be displayed.

**Syntax:**

comm -23 (To display first column)

comm -13 (To display second column)

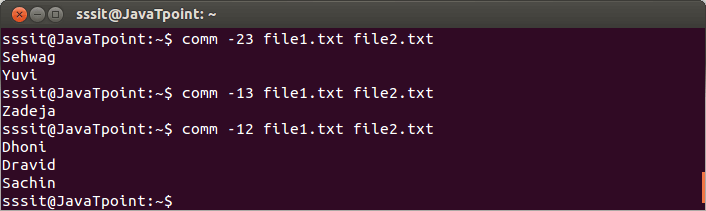
comm -12 (To display third column)

**Example:**

comm -23 file1.txt file2.txt

comm -13 file1.txt file2.txt

comm -12 file1.txt file2.txt



Look at the above snapshot, column number which needs to be displayed are not mentioned in the 'comm' command.

# Linux sed Command | Linux Stream Editor

Linux 'sed' command stands for stream editor. It is used to edit streams (files) using regular expressions. But this editing is not permanent. It remains only in display, but in actual, file content remains the same.

Primarily, it is used for text substitution; additionally, it can be used for other text manipulation operations like insert, delete, search, and more. The sed command allows us to edit files without opening them. Regular expression support makes it a more powerful text manipulation tool.

### Syntax:

1. sed [OPTION]... {script-only-if-no-other-script} [input-file]...

### Options:

The following are some command line options of the sed command:

**-n, --quiet, --silent:** It forcefully allows us to print of pattern space.

**-e script, --expression=script:** It is used to add the script to the commands to be executed.

**-f script-file, --file=script-file:** It is used to add the contents of script-file to the commands to be executed.

**--follow-symlinks:** it is used to follow symlinks when processing in place.

**-i[SUFFIX], --in-place[=SUFFIX]:** it is used to edit files in place (creates backup if SUFFIX option is supplied).

**-l N, --line-length=N:** It is used to specify the desired line-wrap length for the `l' command.

**--posix:** it is used to disable all GNU extensions.

**-E, -r, --regexp-extended:** It allows us to use the extended regular expressions in the script (for portability use POSIX -E).

**-s, --separate:** it is used for considering files as separate rather than as a single and continues the long stream.

**--sandbox:** It is used to operate in sandbox mode.

**-u, --unbuffered:** It is used for loading the minimal amounts of data from the input files and flushes the output buffers more often.

**-z, --null-data:** It is used to separate lines by NUL characters.

**--help:** it is used to display the help manual.

**--version:** It is used to display version information.

### Examples of sed Command

Let's see the following examples:

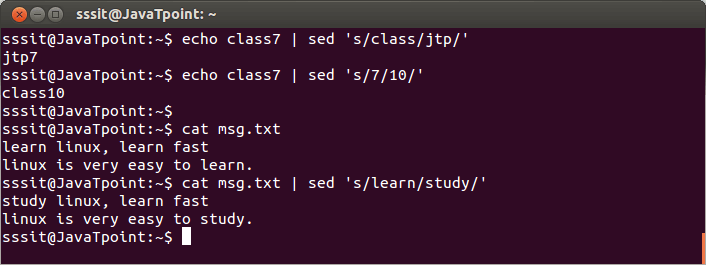
* [Applying to the STDIN directory](https://www.javatpoint.com/linux-sed#Applying)
* [Global Replacement](https://www.javatpoint.com/linux-sed#Global)
* [Removing a Line](https://www.javatpoint.com/linux-sed#Removing)
* [Using the Multiple sed Command](https://www.javatpoint.com/linux-sed#Using)
* [Reading Commands From a File](https://www.javatpoint.com/linux-sed#Reading)
* [Replacing Characters](https://www.javatpoint.com/linux-sed#Replacing)
* [Limiting the sed](https://www.javatpoint.com/linux-sed#Limiting)
* [Inserting and Appending Text](https://www.javatpoint.com/linux-sed#Inserting)
* [Modifying Lines](https://www.javatpoint.com/linux-sed#Modifying)
* [Transformation of Characters](https://www.javatpoint.com/linux-sed#Transformation)
* [Printing the Line Numbers](https://www.javatpoint.com/linux-sed#Printing)

### Applying to the STDIN directory

The sed command is not just limited to manipulate files; also, we can apply it to the STDIN directory.

1. echo class7 | sed 's/class/jtp/'
2. echo class7 | sed 's/7/10/'
3. cat msg.txt | sed 's/learn/study/'

The above commands will replace the first text with the second text pattern. Consider the below output:



From the above output, first, we have performed 'sed' command on a string 'class7' where 'class' is changed into 'jtp' and 7 into 10. Then we have performed 'sed' command on a stream 'msg.txt' where 'learn' is converted into 'study.'

### Global Replacement

In the earlier example, all 'learn' words were not edited into 'study'. To edit every word, we have to use a global replacement 'g'. It will edit all the specified words in a file or string.

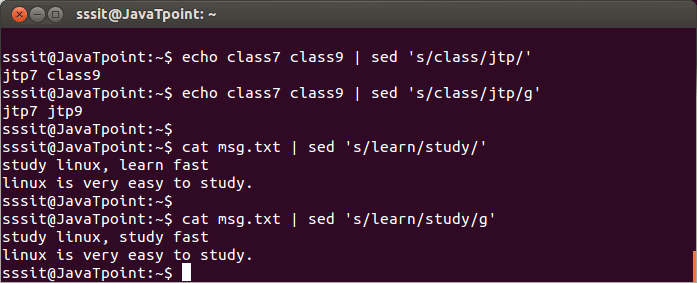
**Syntax:**

1. command | sed 's/**<oldWord>**/**<newWord>**/g'

Consider the below examples:

1. echo class7 class9 | sed 's/class/jtp/g'
2. cat msg.txt | sed 's/learn/study/g'

The above commands will replace all the specified text pattern. Consider the below output:



From the above output, by executing the command "echo class7 class9 | sed 's/class/jtp/g'" all the 'class' is converted into 'jtp' and with command "cat msg.txt | sed 's/learn/study/g'" all the 'learn' was converted into 'study'.

### Removing a Line

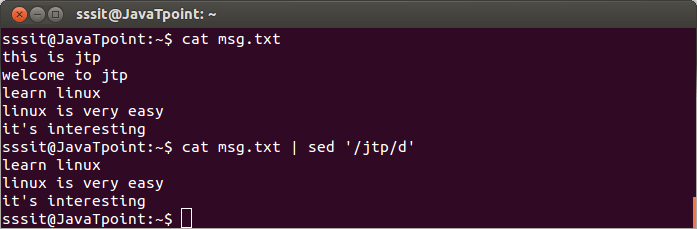
The 'd' option will let us remove a complete line from a file. We only need to specify a word from that line with 'd' option, and that line will be deleted. But, note that all the lines having that same word will be deleted. It will be executed as:

1. cat **<fileName>** | sed '/**<Word>**/d'

Consider the below command:

1. cat msg.txt | sed '/jtp/d'

The above command will delete the lines having the word 'jtp'. Consider the below output:



From the above output, by executing the command "cat msg.txt | sed '/jtp/d'" all lines containing the word 'jtp' are deleted.

### Using the Multiple sed Command

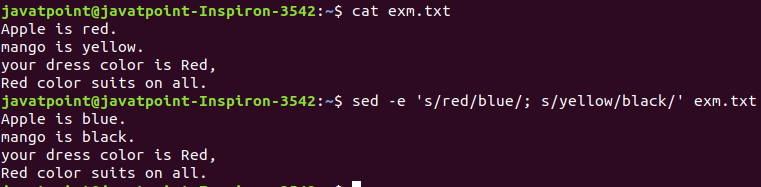
The '-e' option allows us to execute the multiple sed commands at once. We can perform more than one sed operation by executing the command as:

1. sed -e '**<script** 1**>** ; **<script** 2**>**' **<file** name**>**

Consider the below command:

1. sed -e 's/red/blue/; s/yellow/black/' exm.txt

The above command will apply all the specified operations in file 'exm.txt'. Consider the below output:



As we can see from the above output, all the 'red' words are replaced with 'blue,' and all the 'yellow' words are replaced with 'black.' We can also separate commands like this:

1. sed -e '
2. **>**s/red/blue/;
3. **>**s/yellow/black/' exm.txt

The result will be the same as the above command.

### Reading Commands From a File

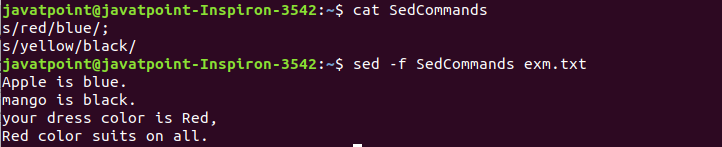
We can save the sed commands in a file and apply them at once in any file. It can be done by specifying the '-f' option as follows:

1. sed -f **<sed** file**>** **<file** name**>**

From the above command, the '<sed file>' is a file that has a sed command list. Consider the below command:

1. sed -f SedCommands exm.txt

The above command will apply all the specified commands in the 'SedCommand' file on 'exm.txt'. Consider the below output:



From the above output, we have used commands applied to the earlier example. So, the output is the same as the previous example.

### Replacing Characters

We can use the exclamation mark (!) as a string delimiter. For example, we want to replace bash shell and replace it with csh shell in the "/etc/passwd". To do so, execute the below command:

1. sed 's/\/bin\/bash/\/bin\/csh/' /etc/passwd

We can achieve the same result by executing the below command:

1. sed 's!/bin/bash!/bin/csh!' /etc/passwd

### Limiting the sed

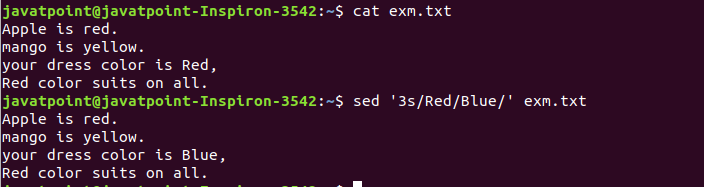
The basic use of the sed command process the entire file. But, we can limit the sed command and specify any line. There are two ways to limit the sed command:

* A range of lines.
* A pattern that matches a specific line.

We can provide a number to specify a line as follows:

1. sed '3s/Red/Blue/' exm.txt

The above command will apply the specified operation on the third line. Consider the below output:

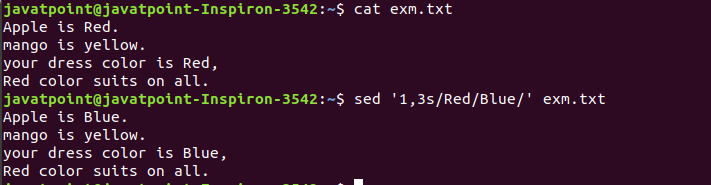


From the above output, only the line three is modified.

We can also specify a range of lines. To specify a range of lines, execute the command as follows:

1. sed '1,3s/Red/Blue/' exm.txt

The above command will update the specified text in lines 1 and 3. Consider the below output:



### Inserting and Appending Text

The 'i' and 'a' flag is used to insert and append the text on a file. The 'i' flag will add the text before the string, and the 'a' flag is used to add text after the string. Consider the below command:

1. echo "Another Demo" | sed 'i\First Demo'

The above command will insert the text before the text "Another Demo". Consider the below output:

Linux Sed Filter

To append text, execute the command as follows:

1. echo "Another Demo" | sed 'a\First Demo'

The above command will append the text. Consider the below output:

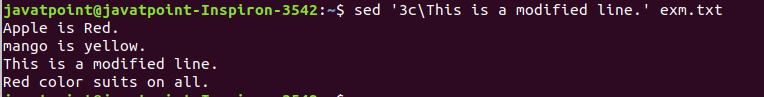
Linux Sed Filter

### Modifying Lines

The 'c' flag is used to modify a specific line. To modify a line, execute the command as follows:

1. sed '3c\This is a modified line.' exm.txt

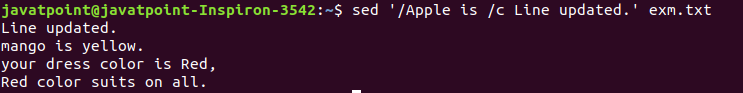
The above command will update the line three. Consider the below output:



We can also use a regular expression to update more than one lines having the same pattern. Consider the below command:

1. sed '/Apple is /c Line updated.' exm.txt

The above command will update all the lines having string 'Apple is'. Consider the below output:

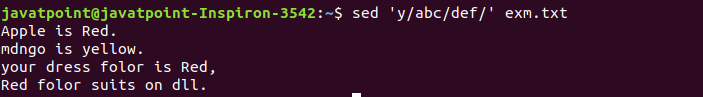


### Transformation of Characters

The 'y' flag is used to transform the characters. The transformation of characters cannot be limited to specific occurrences. To transform characters, execute the command as follows:

1. sed 'y/abc/def/' exm.txt

The above command will transform the characters 'a', 'b', 'c' into 'd', 'e', 'f'. consider the below output:

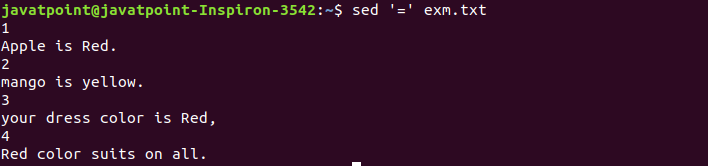


### Printing the Line Numbers

The '=' sign is used to print the line number. To print the line number, execute the command as follows:

1. sed '=' exm.txt

The above command will display the line number of file content. Consider the below output:



The equal sign with the '-n' option specifies the line number that contains a matching script. Consider the below output:

1. sed -n '/mango/=' exm.txt

The above command will display the line number that contains the word 'mango'. Consider the below output:

Linux Sed Filter

From the above output, we can see the line number 2 has the 'mango' word.

# Linux tee Command

Linux tee command is quite similar to the 'cat' command, with only one difference. It puts stdin on stdout and also put them into a file. It is one of the most used commands with other commands through piping. It allows us to write whatever is provided from std input to std output. Optionally, it provides writing to one or more files. The command name tee comes from T splitter used in plumbing.

### Syntax:

1. tee **<options>** **<file** name**>**

### Options:

The following are some useful options that can be used with the tee command to make it more specific:

**-a, --append:** It is used to append the data to the given files, it does not overwrite data.

**-i, --ignore-interrupts:** It is used to ignore the interrupt signals.

**-p:** It is used to diagnose errors writing to non-pipes.

**--output-error[=MODE]:** It is used to set behavior on write error mode.

**--help:** It is used to display the help documentation.

**--version:** it is used to display the version information.

### Examples of the tee Command

Let's see the following examples of the tee command:

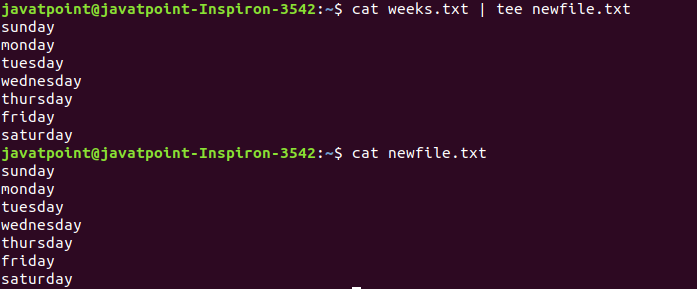
* [How to use the tee command](https://www.javatpoint.com/linux-tee#How)
* [Write a file and append output](https://www.javatpoint.com/linux-tee#append)
* [Write the state of Data to a File](https://www.javatpoint.com/linux-tee#state)
* [Write to multiple files](https://www.javatpoint.com/linux-tee#multiple)
* [Write to a privileged file](https://www.javatpoint.com/linux-tee#privileged)
* [Ignoring Interrupts](https://www.javatpoint.com/linux-tee#Ignoring)
* [Hide the output](https://www.javatpoint.com/linux-tee#Hide)

### How to use the tee command

The tee command is used to write a standard input to standard output and a file. It is used after a pipe. To write to standard output and a file, specify the tee command after a pipe and provide the file(s) name. Consider the below command:

1. cat weeks.txt | tee newfile.txt

The above command will write the input of the 'weeks.txt' to 'newfile.txt.' Consider the below output:



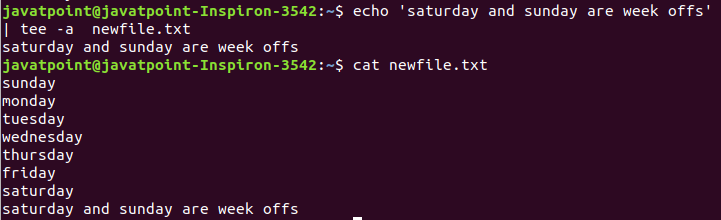
From the above output, a file 'newfile.txt' is created using the tee command.

### Write a file and append output

The '-a' option is used with the tee command to append the output and write it to a file. Consider the below command:

1. echo 'Saturday and Sunday are week offs' | tee -a newfile.txt

The above command will append the specified input to the 'newfile.txt'. Consider the below output:



As from the above output, the specified input is appended to 'newfile.txt'.

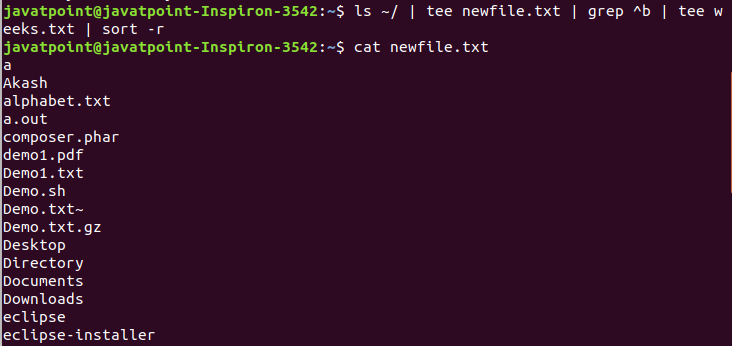
### Write the State of Data to a File

Writing the state of data is very useful for taking backup or creating a snap of the data for the debugging purpose. It can be easily done by using the tee command.

To write the state of data to a file, execute the below command:

1. ls ~/ | tee pipe1.txt | grep ^b | tee pipe2.txt | sort -r

The above command will write the data to the pipe1.txt. Consider the below output:



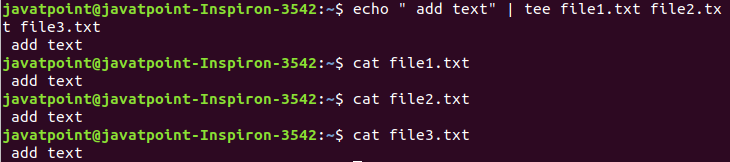
From the above output, the snap of current working directory data is stored in 'newfile.txt'.

### Write to multiple files

The tee command allows us to write to multiple files. To write to the multiple files, specify the names of the files after the tee command as follows:

1. echo " add text" | tee file1.txt file2.txt file3.txt

The above command will create all the specified files. Consider the below output:



### Write to a privileged file

The tee command allows us to write to a file having sudo privilege. If we try to write a file owned by the root user will through the permission error. But, we can elevate the sudo permission by executing the tee command as follows:

1. echo "some text" | sudo tee -a **<file** name**>**

### Ignoring Interrupts

The '-i' option is used to ignore the interrupts. This is useful if we want to stop and exit the command gracefully during the execution. It is used as follows:

1. command | tee -i **<filename>**

### Hide the output

To restrict the tee command not to write to the standard output, redirect it to "/dev/null". Execute the command as follows:

1. echo "Text" | tee newfile.txt **>**/dev/null

Consider the below output:

Linux tee Filters

Linux tr

The command 'tr' stands for **'translate'**. It is used to translate, like from lowercase to uppercase and vice versa or new lines into spaces.

**Syntax:**

1. command | tr **<**'old'**>** **<**'new'**>**

Change Case

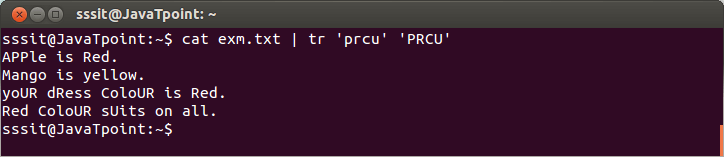
The 'tr' command can change case.

**Syntax:**

1. command | tr **<**'old'**>** **<**'new'**>**

**Example:**

1. cat exm.txt | tr 'prcu' 'PRCU'



Look at the above snapshot,**all p,r,c,u** are converted into upprecase **P,R,C,U.**

Remove New Lines

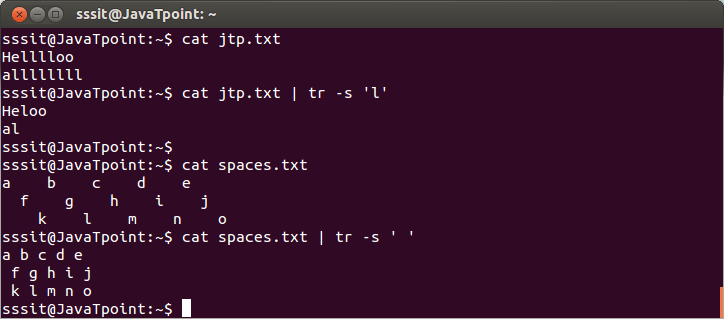
To write all the lines into a single line we have to translate all new lines into spaces.

**Syntax:**

1. command | tr **<**'\n'**>** **<**' '**>**

**Example:**

1. cat exm.txt | tr '\n' ' '



tr Options

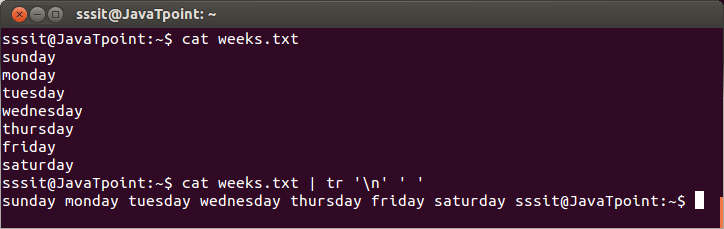
* **tr -s**: The 'tr -s' command squeezes the occurence of multiple characters into one.

**Syntax:**

* 1. command | tr -s **<**'letter'**>**

**Example:**

* 1. cat jtp.txt | tr -s 'l'
  2. cat spaces.txt | tr -s ' '

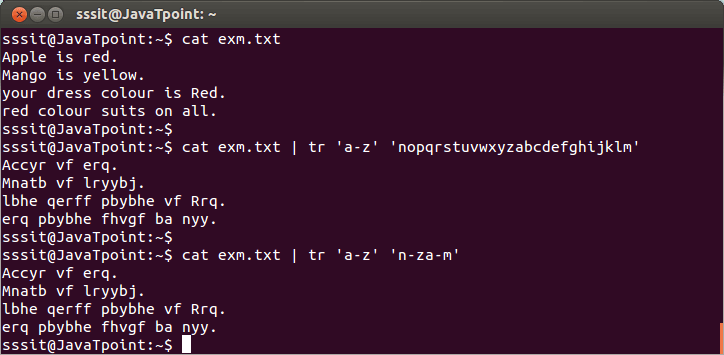


Look at the above snapshot, command **"cat jtp.txt | tr -s 'l'"** has squeezed all the letters 'l' into one and command **"cat spaces.txt | tr -s ' '"** has squeezed all the spaces into single space.

* **tr rot13**: This command encrypts the text. It is case-sensitive.

**Example:**

* 1. cat exm.txt | tr 'a-z' 'nopqrstuvwxyzabcdefghijklm'
  2. cat exm.txt | tr 'a-z' 'n-za-m'



Look at the above snapshot, all the letters are encrypted according to the command. But letter 'A' and 'M' are not encrypted as they are in upper case.

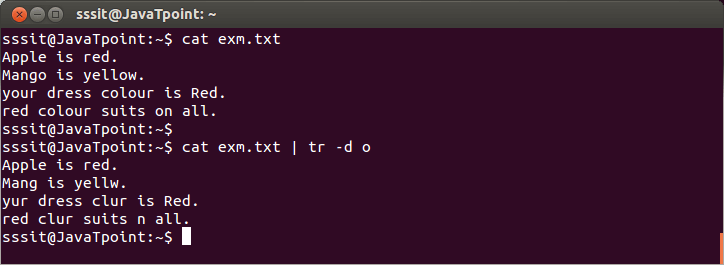
* **tr -d**: The 'tr -d' command is used to delete characters.

**Syntax:**

* 1. command | tr -d **<letter>**

**Example:**

* 1. cat exm.txt | tr -d o



Look at the above snapshot, all the **'o'** letters are deleted from the file **'exm.txt'**.

# Linux uniq Command

Linux uniq command is used to remove all the repeated lines from a file. Also, it can be used to display a count of any word, only repeated lines, ignore characters, and compare specific fields. It is one of the most frequently used commands in [the Linux](https://www.javatpoint.com/linux-tutorial) system. It is often used with the [sort command](https://www.javatpoint.com/linux-sort) because it compares adjacent characters. It discards all the identical lines and writes the output.

### Syntax:

1. uniq [OPTION]... [INPUT [OUTPUT]]

### Options:

Some useful command line options of the uniq command are as following:

**-c, --count:** it prefixes the lines by the number of occurrences.

**-d, --repeated:** it is used to print duplicate lines, one for each group.

**-D:** It is used to print all the duplicate lines.

**--all-repeated[=METHOD]:** It is quite similar to the '-D' option, the difference between both the options is that it allows separation of groups with an empty line.

**-f, --skip-fields=N:** It is used to avoid comparison of the first N fields.

**--group[=METHOD]:** It is used to display all items and separates the groups with an empty line.

**-i, --ignore-case:** It is used to ignore the differences while comparing.

**-s, --skip-chars=N:** It is used to avoid the comparison of the first N characters.

**-u, --unique:** it is used to print unique lines.

**-z, --zero-terminated:** It is used for the line delimiter is NUL and not newline mode.

**-w, --check-chars=N:** It is used to compare not more than N characters in lines.

**--help:** It is used to display help documentation.

**--version:** It is used to display the version information.

### Examples of uniq Command

Let's see the following examples of the uniq command:

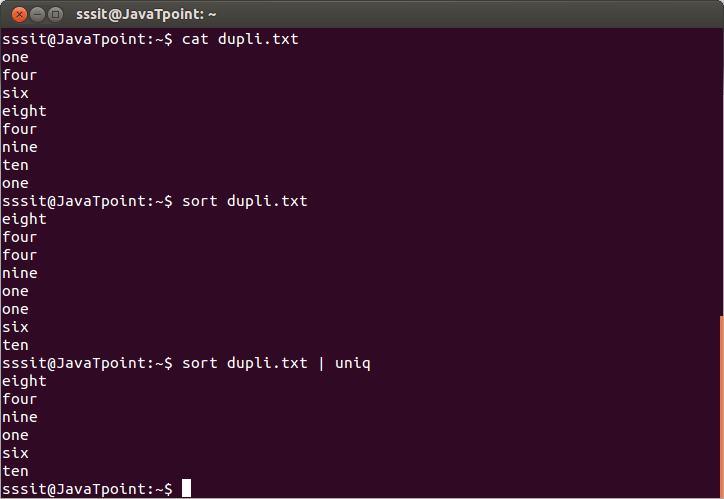
* [Remove repeated lines](https://www.javatpoint.com/linux-uniq#Remove)
* [count the number of occurrences of a word](https://www.javatpoint.com/linux-uniq#occurrences)
* [Display the repeated lines](https://www.javatpoint.com/linux-uniq#repeated)
* [Display the unique lines](https://www.javatpoint.com/linux-uniq#unique)
* [Ignore characters in comparison](https://www.javatpoint.com/linux-uniq#characters)
* [Ignore fields in comparison](https://www.javatpoint.com/linux-uniq#fields)

### Remove repeated lines

To remove repeated lines from a file, execute the basic uniq command as follows:

1. sort dupli.txt | uniq

The above command will remove the duplicate lines from the file 'dupli.txt.' Consider the below output:



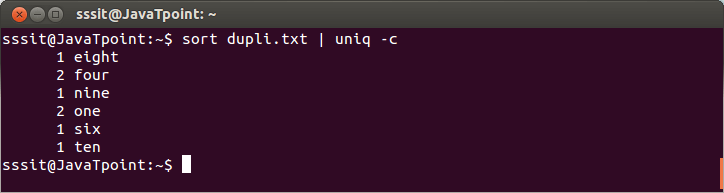
From the above output, the repeating words are ignored.

### Count the number of occurrences of a word

We can count the number of occurrences of a word by using the uniq command. The '-c' option is used to count the word. Execute it as follows:

1. sort dupli.txt | uniq -c

The above command will count the words which come in 'dupli.txt'. Consider the below output:



From the above output, the command "sort dupli.txt | uniq -c" counts the number of times a word is repeating.

### Display the repeated lines

The '-d' option is used to display only the repeated lines. It will only display the lines that will be more than once in a file and write the output to standard output. Consider the below command:

1. sort dupli.txt | uniq -d

The above command will display only the repeated lines. Consider the below output:

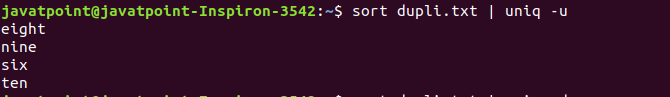
Linux Uniq Filters

### Display the unique lines

The '-u' option is used to display only the unique lines ( which are not repeated). It will only display the lines that occur only once and write the result to standard output. Consider the below command:

1. sort dupli.txt | uniq -u

The above command will display only the unique lines from the file 'dupli.txt'. Consider the below output:

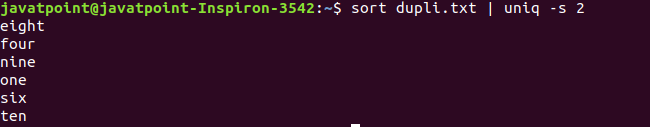


### Ignore characters in comparison

The '-s' option is used to ignore the characters in comparison. It will ignore the specified number of characters and display the result to standard output. Consider the below command:

1. sort dupli.txt | uniq -s 2

The above command will ignore the first two characters in comparison from the file 'dupli.txt'. Consider the below output:

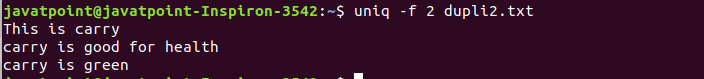


### Ignore fields in comparison

The '-f' option is used to ignore the fields. Consider the below command:

1. uniq -f 2 dupli2.txt

The above command will not compare the first two fields from the file 'dupli2.txt'. Consider the below output:



From the above output, the first two fields are skipped, and the rest of all fields are compared from the file 'dupli2.txt'.

# Linux wc Command

Linux wc command helps in counting the lines, words, and characters in a file. It displays the number of lines, number of characters, and the number of words in a file. Mostly, it is used with pipes for counting operation.

### Syntax:

1. wc [OPTION]... [FILE]...
2. wc [OPTION]... --files0-from=F

### Options:

Some useful command line options supported by the wc command are as following:

**-c, --bytes:** It is used to print the byte counts.

**-m, --chars:** It is used to print the character counts.

**-l, --lines:** It is used to print the newline counts.

**--files0-from=F:** It is used to read input from specified files.

**-L, --max-line-length:** It is used to print the maximum display width.

**-w, --words:** It is used to print the word counts.

**--help:** It is used to display the help manual.

**--version:** It is used to display the version information.

### Examples of the wc Command

Let's see the following examples of the wc command:

* [Display count information of a file](https://www.javatpoint.com/linux-wc#file)
* [Display count information of multiple files](https://www.javatpoint.com/linux-wc#multiple)
* [Display the number of lines in a file](https://www.javatpoint.com/linux-wc#lines)
* [Display the number of characters in a file](https://www.javatpoint.com/linux-wc#characters)
* [Display the number of bytes in a file](https://www.javatpoint.com/linux-wc#bytes)
* [Display the number of words in a file](https://www.javatpoint.com/linux-wc#words)
* [Count the number files in a directory](https://www.javatpoint.com/linux-wc#directory)
* [Display the length of the longest line](https://www.javatpoint.com/linux-wc#longest)

### Display count information of a file

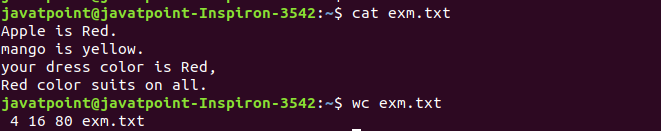
To display the complete count information of a file, execute the command without any argument. It will display the number of lines, words, and bytes from the file. Execute the command as follows:

1. wc **<file** name**>**

Consider the below command:

1. wc exm.txt

The above command will display the number of lines, number of words, number of bytes, and file name from the file 'exm.txt'. Consider the below output:



### Display count information of multiple files

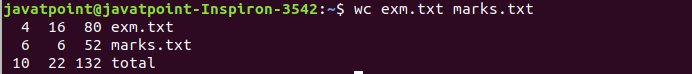
To display the complete count information of multiple files at once, specify the file names after space (' '). It is executed as follows:

1. wc **<file1>** **<file2>**

Consider the below example:

1. wc exm.txt marks.txt

The above command will display the number of words, the number of characters, and the number of the bytes from the files 'exm.txt' and 'marks.txt'. Consider the below output:



### Display the number of lines in a file

The '-l' option is used to display the number of lines in a file. It is executed as follows:

1. wc - l **<file** name**>**

Consider the below example:

1. wc -l exm.txt

The above command will display the number of lines from 'exm.txt'. Consider the below output:

Linux wc Filters

### Display the number of characters in a file

The '-m' option is used to display the number of characters in a file. It is executed as follows:

1. wc -m **<file** name**>**

Consider the below example:

1. wc -m exm.txt

The above command will display the number of words from the file 'exm.txt'. Consider the below output:

Linux wc Filters

### Display the number of bytes in a file

The '-c' option is used to display the number of bytes in a file. It is executed as follows:

1. wc -c **<file** name**>**

Consider the below example:

1. wc -c exm.txt

The above command will display the number of bytes in a file. Consider the below output:

Linux wc Filters

### Display the number of words in a file

The '-w' option is used to display the total number of words from a file. It is executed as follows:

1. wc -w **<file** name**>**

Consider the below example:

1. wc -w exm.txt

The above command will display the total number of words from the file 'exm.txt'. Consider the below output:

Linux wc Filters

### Count the number files in a directory

To count the number of files and folders in a directory, combine the wc command with the [ls command](https://www.javatpoint.com/linux-ls). Execute it as follows:

1. ls | wc -l

The above command will display the count of the files from the current working directory. Consider the below output:

Linux wc Filters

### Display the length of the longest line

The '-L' option is used to display the length of the longest line from a file. It is executed as follows:

1. wc -L **<file** name**>**

Consider the below example:

1. wc -L exm.txt

The above command will display the length of the longest line of the file 'exm.txt'. Consider the below output:

Linux wc Filters