**SED: Stream Editor – replace nth string with another string.**

>sed 's/unix/linux/' file.txt

Use the /1, /2 etc flags to replace the first, second occurrence of a pattern in a line. The below command replaces the second occurrence of the word "unix" with "linux" in a line.  
>sed 's/unix/linux/2' file.txt

The substitute flag /g (global replacement) specifies the sed command to replace all the occurrences of the string in the line.

>sed 's/unix/linux/g' file.txt

Use the combination of /1, /2 etc and /g to replace all the patterns from the nth occurrence of a pattern in a line. The following sed command replaces the third, fourth, fifth... "unix" word with "linux" word in a line.

>sed 's/unix/linux/3g' file.txt

The /p print flag prints the replaced line twice on the terminal. If a line does not have the search pattern and is not replaced, then the /p prints that line only once.

>sed 's/unix/linux/p' file.txt

Printing only the replaced lines  
Use the -n option along with the /p print flag to display only the replaced lines. Here the -n option suppresses the duplicate rows generated by the /p flag and prints the replaced lines only one time.  
>sed -n 's/unix/linux/p' file.txt

Running multiple sed commands.  
You can run multiple sed commands by piping the output of one sed command as input to another sed command.  
>sed 's/unix/linux/' file.txt| sed 's/os/system/'

Replacing string on a specific line number.  
You can restrict the sed command to replace the string on a specific line number. An example is  
>sed '3 s/unix/linux/' file.txt

**AWK:** Awk is one of the most powerful tools in Unix used for processing the rows and columns in a file. Awk has built in string functions and associative arrays. Awk supports most of the operators, conditional blocks, and loops available in C language.   
  
One of the good things is that you can convert Awk scripts into Perl scripts using a2p utility.   
  
**The basic syntax of AWK:**  
awk 'BEGIN {start\_action} {action} END {stop\_action}' filename

awk 'BEGIN {sum=0} {sum=sum+$5} END {print sum}' input\_file

awk 'BEGIN { for(i=1;i<=5;i++) print "square of", i, "is",i\*i; }'

**NF** - Number of fileds variable:  
The NF can be used to know the number of fields in line  
awk '{print NF}' input\_file  
This will display the number of columns in each row.  
  
**NR** - number of records variable:   
The NR can be used to know the line number or count of lines in a file.  
awk '{print NR}' input\_file  
This will display the line numbers from 1.  
  
awk 'END {print NR}' input\_file  
This will display the total number of lines in the file.

**GREP:**

Grep is the frequently used command in Unix (or Linux). Most of us use grep just for finding the words in a file.

Running the last executed grep command

!grep

Search for a string in a file  
This is the basic usage of grep command. It searches for the given string in the specified file.  
grep "Error" logfile.txt

Searching for a string in multiple files.  
grep "string" file1 file2

grep "string" file\_pattern

Case insensitive search  
The -i option enables to search for a string case insensitively in the give file. It matches the words like "UNIX", "Unix", "unix".  
grep -i "UNix" file.txt

Checking for the whole words in a file.  
By default, grep matches the given string/pattern even if it found as a substring in a file. The -w option to grep makes it match only the whole words.  
grep -w "world" file.txt

Displaying the lines before the match.  
Sometimes, if you are searching for an error in a log file; it is always good to know the lines around the error lines to know the cause of the error.  
grep -B 2 "Error" file.txt

This will prints the matched lines along with the two lines before the matched lines.  
  
Displaying the lines after the match.  
grep -A 3 "Error" file.txt

This will display the matched lines along with the three lines after the matched lines.  
  
Displaying the lines around the match  
grep -C 5 "Error" file.txt

This will display the matched lines and also five lines before and after the matched lines.  
  
Searching for a sting in all files recursively  
You can search for a string in all the files under the current directory and sub-directories with the help -r option.  
grep -r "string" \*

Inverting the pattern match  
You can display the lines that are not matched with the specified search sting pattern using the -v option.  
grep -v "string" file.txt

Displaying the non-empty lines  
You can remove the blank lines using the grep command.  
grep -v "^$" file.txt

Displaying the count of number of matches.  
We can find the number of lines that matches the given string/pattern  
grep -c "sting" file.txt

Display the file names that matches the pattern.  
We can just display the files that contains the given string/pattern.  
grep -l "string" \*

Display the file names that do not contain the pattern.  
We can display the files which do not contain the matched string/pattern.  
grep -L "string" \*

Displaying only the matched pattern.  
By default, grep displays the entire line which has the matched string. We can make the grep to display only the matched string by using the -o option.  
grep -o "string" file.txt

Displaying the line numbers.  
We can make the grep command to display the position of the line which contains the matched string in a file using the -n option  
grep -n "string" file.txt

Displaying the position of the matched string in the line  
The -b option allows the grep command to display the character position of the matched string in a file.  
grep -o -b "string" file.txt

Matching the lines that start with a string  
The ^ regular expression pattern specifies the start of a line. This can be used in grep to match the lines which start with the given string or pattern.  
grep "^start" file.txt

Matching the lines that end with a string  
The $ regular expression pattern specifies the end of a line. This can be used in grep to match the lines which end with the given string or pattern.  
grep "end$" file.txt

**FIND:** Find is one of the powerful utility of Unix (or Linux) used for searching the files in a directory hierarchy. The syntax of find command is  
find [pathnames] [conditions]

Let see some practical exercises on using find command.  
  
**1.** How to run the last executed find command?  
!find

This will execute the last find command. It also displays the last find command executed along with the result on the terminal.  
  
**2.** How to find for a file using name?  
find -name "sum.java"

./bkp/sum.java

./sum.java

This will find all the files with name "sum.java" in the current directory and sub-directories.  
  
**3.** How to find for files using name and ignoring case?  
find -iname "sum.java"

./SUM.java

./bkp/sum.java

./sum.java

This will find all the files with name "sum.java" while ignoring the case in the current directory and sub-directories.

**4.** How to find for a file in the current directory only?  
find -maxdepth 1 -name "sum.java"

./sum.java

This will find for the file "sum.java" in the current directory only  
  
**5.** How to find for files containing a specific word in its name?  
find -name "\*java\*"

./SUM.java

./bkp/sum.java

./sum.java

./multiply.java

It displayed all the files which have the word "java" in the filename  
  
**6.** How to find for files in a specific directory?  
find /etc -name "\*java\*"

This will look for the files in the /etc directory with "java" in the filename  
  
**7.** How to find the files whose name are not "sum.java"?  
find -not -name "sum.java"

.

./SUM.java

./bkp

./multiply.java

This is like inverting the match. It prints all the files except the given file "sum.java".  
  
**8.** How to limit the file searches to specific directories?  
find -name "sum.java"

./tmp/sum.java

./bkp/var/tmp/files/sum.java

./bkp/var/tmp/sum.java

./bkp/var/sum.java

./bkp/sum.java

./sum.java

You can see here the find command displayed all the files with name "sum.java" in the current directory and sub-directories.  
  
**a.** How to print the files in the current directory and one level down to the current directory?  
find -maxdepth 2 -name "sum.java"

./tmp/sum.java

./bkp/sum.java

./sum.java

**b.** How to print the files in the current directory and two levels down to the current directory?  
find -maxdepth 3 -name "sum.java"

./tmp/sum.java

./bkp/var/sum.java

./bkp/sum.java

./sum.java

**c.** How to print the files in the subdirectories between level 1 and 4?  
find -mindepth 2 -maxdepth 5 -name "sum.java"

./tmp/sum.java

./bkp/var/tmp/files/sum.java

./bkp/var/tmp/sum.java

./bkp/var/sum.java

./bkp/sum.java

**9.** How to find the empty files in a directory?  
find . -maxdepth 1 –empty

./empty\_file

**10.** How to find the largest file in the current directory and sub directories  
find . -type f -exec ls -s {} \; | sort -n -r | head -1

The find command "find . -type f -exec ls -s {} \;" will list all the files along with the size of the file. Then the sort command will sort the files based on the size. The head command will pick only the first line from the output of sort.  
  
**11.** How to find the smallest file in the current directory and sub directories  
find . -type f -exec ls -s {} \; | sort -n -r | tail -1

Another method using find is  
find . -type f -exec ls -s {} \; | sort -n | head -1

**12.** How to find files based on the file type?  
  
**a.** Finding socket files  
find . -type s

**b.** Finding directories  
find . -type d

**c.** Finding hidden directories  
find -type d -name ".\*"

**d.** Finding regular files  
find . -type f

**e.** Finding hidden files  
find . -type f -name ".\*"

**13.** How to find files based on the size?  
  
**a.** Finding files whose size is exactly 10M  
find . -size 10M

**b.** Finding files larger than 10M size  
find . -size +10M

**c.** Finding files smaller than 10M size  
find . -size -10M

**14.** How to find the files which are modified after the modification of a give file.  
find -newer "sum.java"

This will display all the files which are modified after the file "sum.java"  
  
**15.** Display the files which are accessed after the modification of a give file.  
find -anewer "sum.java"

**16.** Display the files which are changed after the modification of a give file.  
find -cnewer "sum.java"

**17.** How to find the files based on the file permissions?  
find . -perm 777

This will display the files which have read, write, and execute permissions. To know the permissions of files and directories use the command "ls -l".  
  
**18.** Find the files which are modified within 30 minutes.  
find . -mmin -30

**19.** Find the files which are modified within 1 day.  
find . -mtime -1

**20.** How to find the files which are modified 30 minutes back  
find . -not -mmin -30

**21.** How to find the files which are modified 1 day back.  
find . -not -mtime -1

**22.** Print the files which are accessed within 1 hour.  
find . -amin -60

**23.** Print the files which are accessed within 1 day.  
find . -atime -1

**24.** Display the files which are changed within 2 hours.  
find . -cmin -120

**25.** Display the files which are changed within 2 days.  
find . -ctime -2

**26.** How to find the files which are created between two files.  
find . -cnewer f1 -and ! -cnewer f2

So far we have just find the files and displayed on the terminal. Now we will see how to perform some operations on the files.  
  
**1.** How to find the permissions of the files which contain the name "java"?  
find -name "\*java\*"|xargs ls -l

Alternate method is  
find -name "\*java\*" -exec ls -l {} \;

**2.** Find the files which have the name "java" in it and then display only the files which have "class" word in them?  
find -name "\*java\*" -exec grep -H class {} \;

**3.** How to remove files which contain the name "java".  
find -name "\*java\*" -exec rm -r {} \;

This will delete all the files which have the word “java" in the file name in the current directory and sub-directories.

**FTP:**

1. Connect to a FTP site

Connect to a particular FTP server using ftp command as shown below.

Syntax:

$ ftp IP/hostname

or

$ ftp

ftp> open IP/hostname

You can directly open connection with a remote host using it’s IP or host name from the command line. You can also go to ftp prompt and use open command to connect with remote host.

It will ask you for the user name and password to login. On some public domain FTP server, you can use “anonymous” username with any email address as the password to connect.

2. Download a file using ftp

Use the get command to download file from a remote ftp server as shown below.

ftp> get FILENAME

You have to be in the right mode to download files. i.e binary or ascii mode. Use ascii mode for transferring text files, and binary mode for all other type of files.

Download the file and save it with another name. In the following example, index.html file will be downloaded and saved as my.html on the local server.

ftp> get index.html my.html

Fetching /home/groups/index.html to my.html

/home/groups/index.html 100% 2886 1.4KB/s 00:02

3. Changing FTP Mode to binary or ascii

Go to ftp Ascii mode

ftp> ascii

200 Type set to A.

Go to ftp Binary mode

ftp> binary

200 Type set to I.

4. Uploading a file to FTP server

Use put command to upload a file to a remote ftp server as shown below.

ftp> put filename

5. Changing the remote and local directory

Apart from downloading or uploading a file, you may want to change either the remote or local directory, which you can do using cd and lcd respectively.

Change the remote server current directory using cd command

ftp> pwd

257 "/myftpserver" is current directory.

ftp> cd dir1

250 CWD command successful. "/myftpserver/dir1" is current directory.

ftp> pwd

257 "/myftpserver/dir1" is current directory.

Change the local machine current directory using lcd command

ftp> !

$ pwd

/home/sathiya/FTP

$ exit

exit

ftp> lcd /tmp

Local directory now /tmp

ftp> !

$ pwd

/tmp

Note:

executing ! takes you to the shell.

prompt starts with ftp> is ftp prompt.

prompt starts with $ is shell command line.

6. Listing the contents of remote directory from FTP

You can view the content of a remote directory using the ls / dir command.

ftp> ls

7. FTP Help

Type help or ? to view list of all available ftp commands.

For a detailed help on a particular ftp command use:

ftp> help COMMAND

8. Downloading multiple files with mget command

mget is for fetching multiple files from ftp server. You can use globs to download multiple files. For example, \*.html will download all html files. The glob expansion are done on the remote server. So, it depends on the operating system of the remote server.

ftp> mget \*.html

Fetching /ftptest/features.html to features.html

/ftptest/features.html 100% 2256 2.2KB/s 00:01

Fetching /ftptest/index.html to index.html

/ftptest/index.html 100% 2886 2.8KB/s 00:01

Fetching /ftptest/othertools.html to othertools.html

/ftptest/othertools.html 100% 2282 2.2KB/s 00:01

Fetching /ftptest/samplereport.html to samplereport.html

/ftptest/samplereport.html 100% 15KB 7.3KB/s 00:02

Fetching /ftptest/usage.html to usage.html

/ftptest/usage.html 100% 2340 2.3KB/s 00:01

To view the file names before downloading, you can also use mls command as shown below.

ftp> mls \*.html -

/ftptest/features.html

/ftptest/index.html

/ftptest/othertools.html

/ftptest/samplereport.html

/ftptest/usage.html

9. Uploading multiple files with mput command

Use mput to upload multiple files together. This works similar to the mget command. The following example uploads all the \*.html file from local server to remote server.

ftp> mput \*.html

10. Close a FTP connection

Without exiting the ftp prompt you may want to open a connection to another server. In that case, execute close command.

ftp> open ftp.your\_server.com

Already connected to NNN.com, use close first.

ftp> close

221 Goodbye.

ftp> open ftp.your\_server.com

**The vi Editor**

There are many ways to edit files in Unix and for me one of the best ways is using screen-oriented text editor **vi**. This editor enable you to edit lines in context with other lines in the file.

Now a days you would find an improved version of vi editor which is called **VIM**. Here VIM stands for **V**i **IM**proved.

The vi is generally considered the de facto standard in Unix editors because −

* It's usually available on all the flavors of Unix system.
* Its implementations are very similar across the board.
* It requires very few resources.
* It is more user friendly than any other editors like ed or ex.

You can use **vi** editor to edit an existing file or to create a new file from scratch. You can also use this editor to just read a text file.

## **Starting the vi Editor**

There are following way you can start using vi editor −

|  |  |
| --- | --- |
| **Command** | **Description** |
| **vi filename** | Creates a new file if it already does not exist, otherwise opens existing file. |
| **vi -R filename** | Opens an existing file in read only mode. |
| **view filename** | Opens an existing file in read only mode. |

Following is the example to create a new file **testfile** if it already does not exist in the current working directory $vi testfile

So now you have opened one file to start with. Before proceeding further let us understanding few minor but important concepts explained below.

## **Operation Modes**

While working with vi editor you would come across following two modes −

* **Command mode** − This mode enables you to perform administrative tasks such as saving files, executing commands, moving the cursor, cutting (yanking) and pasting lines or words, and finding and replacing. In this mode, whatever you type is interpreted as a command.
* **Insert mode** − This mode enables you to insert text into the file. Everything that's typed in this mode is interpreted as input and finally it is put in the file .

The vi always starts in command mode. To enter text, you must be in insert mode. To come in insert mode you simply type **i**. To get out of insert mode, press the **Esc** key, which will put you back into command mode.

**Hint** − If you are not sure which mode you are in, press the Esc key twice, and then you'll be in command mode. You open a file using vi editor and start type some characters and then come in command mode to understand the difference.

## **Getting Out of vi**

The command to quit out of vi is :q. Once in command mode, type colon, and 'q', followed by return. If your file has been modified in any way, the editor will warn you of this, and not let you quit. To ignore this message, the command to quit out of vi without saving is **:q!**. This lets you exit vi without saving any of the changes.

The command to save the contents of the editor is **:w**. You can combine the above command with the quit command, or :wq and return.

The easiest way to save your changes and exit out of vi is the **ZZ** command. When you are in command mode, type ZZ and it will do the equivalent of :wq.

You can specify a different file name to save to by specifying the name after the :w. For example, if you wanted to save the file you were working as another filename called filename2, you would type **:w filename2** and return. Try it once.

## **Moving within a File**

To move around within a file without affecting your text, you must be in command mode (press Esc twice). Here are some of the commands you can use to move around one character at a time −

|  |  |
| --- | --- |
| **Command** | **Description** |
| **k** | Moves the cursor up one line. |
| **j** | Moves the cursor down one line. |
| **h** | Moves the cursor to the left one character position. |
| **l** | Moves the cursor to the right one character position. |

There are following two important points to be noted −

* The vi is case-sensitive, so you need to pay special attention to capitalization when using commands.
* Most commands in vi can be prefaced by the number of times you want the action to occur. For example, 2j moves cursor two lines down the cursor location.

There are many other ways to move within a file in vi. Remember that you must be in command mode (press Esc twice). Here are some more commands you can use to move around the file −

|  |  |
| --- | --- |
| **Command** | **Description** |
| **0 or |** | Positions cursor at beginning of line. |
| **$** | Positions cursor at end of line. |
| **w** | Positions cursor to the next word. |
| **b** | Positions cursor to previous word. |
| **(** | Positions cursor to beginning of current sentence. |
| **)** | Positions cursor to beginning of next sentence. |
| **E** | Move to the end of Blank delimited word |
| **{** | Move a paragraph back |
| **}** | Move a paragraph forward |
| **[[** | Move a section back |
| **]]** | Move a section forward |
| **n|** | Moves to the column n in the current line |
| **1G** | Move to the first line of the file |
| **G** | Move to the last line of the file |
| **nG** | Move to nth line of the file |
| **:n** | Move to nth line of the file |
| **fc** | Move forward to c |
| **Fc** | Move back to c |
| **H** | Move to top of screen |
| **nH** | Moves to nth line from the top of the screen |
| **M** | Move to middle of screen |
| **L** | Move to botton of screen |
| **nL** | Moves to nth line from the bottom of the screen |
| **:x** | Colon followed by a number would position the cursor on line number represented by **x** |

## **Control Commands**

There are following useful command which you can use along with Control Key −

|  |  |
| --- | --- |
| **Command** | **Description** |
| CTRL+d | Move forward 1/2 screen |
| CTRL+f | Move forward one full screen |
| CTRL+u | Move backward 1/2 screen |
| CTRL+b | Move backward one full screen |
| CTRL+e | Moves screen up one line |
| CTRL+y | Moves screen down one line |
| CTRL+u | Moves screen up 1/2 page |
| CTRL+d | Moves screen down 1/2 page |
| CTRL+b | Moves screen up one page |
| CTRL+f | Moves screen down one page |
| CTRL+I | Redraws screen |

## **Editing Files**

To edit the file, you need to be in the insert mode. There are many ways to enter insert mode from the command mode −

|  |  |
| --- | --- |
| **Command** | **Description** |
| **i** | Inserts text before current cursor location. |
| **I** | Inserts text at beginning of current line. |
| **a** | Inserts text after current cursor location. |
| **A** | Inserts text at end of current line. |
| **o** | Creates a new line for text entry below cursor location. |
| **O** | Creates a new line for text entry above cursor location. |

## **Deleting Characters**

Here is the list of important commands which can be used to delete characters and lines in an opened file −

|  |  |
| --- | --- |
| **Command** | **Description** |
| **x** | Deletes the character under the cursor location. |
| **X** | Deletes the character before the cursor location. |
| **dw** | Deletes from the current cursor location to the next word. |
| **d^** | Deletes from current cursor position to the beginning of the line. |
| **d$** | Deletes from current cursor position to the end of the line. |
| **D** | Deletes from the cursor position to the end of the current line. |
| **dd** | Deletes the line the cursor is on. |

As mentioned above, most commands in vi can be prefaced by the number of times you want the action to occur. For example, **2x** deletes two character under the cursor location and 2dd deletes two lines the cursor is on.

I would highly recommend to exercise all the above commands properly before proceeding further.

## **Change Commands**

You also have the capability to change characters, words, or lines in vi without deleting them. Here are the relevant commands −

|  |  |
| --- | --- |
| **Command** | **Description** |
| **cc** | Removes contents of the line, leaving you in insert mode. |
| **cw** | Changes the word the cursor is on from the cursor to the lowercase w end of the word. |
| **r** | Replaces the character under the cursor. vi returns to command mode after the replacement is entered. |
| **R** | Overwrites multiple characters beginning with the character currently under the cursor. You must use **Esc** to stop the overwriting. |
| **s** | Replaces the current character with the character you type. Afterward, you are left in insert mode. |
| **S** | Deletes the line the cursor is on and replaces with new text. After the new text is entered, vi remains in insert mode. |

## **Copy and Paste Commands**

You can copy lines or words from one place and then you can past them at another place using following commands −

|  |  |
| --- | --- |
| **Command** | **Description** |
| **yy** | Copies the current line. |
| **yw** | Copies the current word from the character the lowercase w cursor is on until the end of the word. |
| **p** | Puts the copied text after the cursor. |
| **P** | Puts the yanked text before the cursor. |

## **Advanced Commands**

There are some advanced commands that simplify day-to-day editing and allow for more efficient use of vi −

|  |  |
| --- | --- |
| **Command** | **Description** |
| **J** | Join the current line with the next one. A count joins that many lines. |
| **<<** | Shifts the current line to the left by one shift width. |
| **>>** | Shifts the current line to the right by one shift width. |
| **~** | Switch the case of the character under the cursor. |
| **^G** | Press CNTRL and G keys at the same time to show the current filename and the status. |
| **U** | Restore the current line to the state it was in before the cursor entered the line. |
| **u** | Undo the last change to the file. Typing 'u' again will re-do the change. |
| **J** | Join the current line with the next one. A count joins that many lines. |
| **:f** | Displays current position in the file in % and file name, total number of file. |
| **:f filename** | Renames current file to filename. |
| **:w filename** | Write to file filename. |
| **:e filename** | Opens another file with filename. |
| **:cd dirname** | Changes current working directory to dirname. |
| **:e #** | Use to toggle between two opened files. |
| **:n** | In case you open multiple files using vi, use :n to go to next file in the series. |
| **:p** | In case you open multiple files using vi, use :p to go to previous file in the series. |
| **:N** | In case you open multiple files using vi, use :N to go to previous file in the series. |
| **:r file** | Reads file and inserts it after current line |
| **:nr file** | Reads file and inserts it after line n. |

## **Word and Character Searching**

The vi editor has two kinds of searches: string and character. For a string search, the / and ? commands are used. When you start these commands, the command just typed will be shown on the bottom line, where you type the particular string to look for.

These two commands differ only in the direction where the search takes place −

* The / command searches forwards (downwards) in the file.
* The ? command searches backwards (upwards) in the file.

The n and N commands repeat the previous search command in the same or opposite direction, respectively. Some characters have special meanings while using in search command and preceded by a backslash (\) to be included as part of the search expression.

|  |  |
| --- | --- |
| **Character** | **Description** |
| **^** | Search at the beginning of the line. (Use at the beginning of a search expression.) |
| **.** | Matches a single character. |
| **\*** | Matches zero or more of the previous character. |
| **$** | End of the line (Use at the end of the search expression.) |
| **[** | Starts a set of matching, or non-matching expressions. |
| **<** | Put in an expression escaped with the backslash to find the ending or beginning of a word. |
| **>** | See the '<' character description above. |

The character search searches within one line to find a character entered after the command. The f and F commands search for a character on the current line only. f searches forwards and F searches backwards and the cursor moves to the position of the found character.

The t and T commands search for a character on the current line only, but for t, the cursor moves to the position before the character, and T searches the line backwards to the position after the character.

## **Set Commands**

You can change the look and feel of your vi screen using the following **:set**commands. To use these commands you have to come in command mode then type **:set** followed by any of the following options −

|  |  |
| --- | --- |
| **Command** | **Description** |
| **:set ic** | Ignores case when searching |
| **:set ai** | Sets autoindent |
| **:set noai** | To unset autoindent. |
| **:set nu** | Displays lines with line numbers on the left side. |
| **:set sw** | Sets the width of a software tabstop. For example you would set a shift width of 4 with this command: **:set sw=4** |
| **:set ws** | If *wrapscan* is set, if the word is not found at the bottom of the file, it will try to search for it at the beginning. |
| **:set wm** | If this option has a value greater than zero, the editor will automatically "word wrap". For example, to set the wrap margin to two characters, you would type this: **:set wm=2** |
| **:set ro** | Changes file type to "read only" |
| **:set term** | Prints terminal type |
| **:set bf** | Discards control characters from input |

## **Running Commands**

The vi has the capability to run commands from within the editor. To run a command, you only need to go into command mode and type **:!** command.

For example, if you want to check whether a file exists before you try to save your file to that filename, you can type **:! ls** and you will see the output of ls on the screen.

When you press any key (or the command's escape sequence), you are returned to your vi session.

## **Replacing Text**

The substitution command (**:s/**) enables you to quickly replace words or groups of words within your files. Here is the simple syntax −

:s/search/replace/g

The g stands for globally. The result of this command is that all occurrences on the cursor's line are changed.

## **IMPORTANT**

Here are the key points to your success with vi −

* You must be in command mode to use commands. (Press Esc twice at any time to ensure that you are in command mode.)
* You must be careful to use the proper case (capitalization) for all commands.
* You must be in insert mode to enter text.