**GREP Command**

**Search for the given string in a single file**

The basic usage of grep command is to search for a specific string in the specified file as shown below.

Syntax:

grep "literal\_string" filename

$ grep "this" demo\_file

this line is the 1st lower case line in this file.

Two lines above this line is empty.

And this is the last line.

**2. Checking for the given string in multiple files.**

Syntax:

grep "string" FILE\_PATTERN

This is also a basic usage of grep command. For this example, let us copy the demo\_file to demo\_file1. The grep output will also include the file name in front of the line that matched the specific pattern as shown below. When the Linux shell sees the meta character, it does the expansion and gives all the files as input to grep.

$ cp demo\_file demo\_file1

$ grep "this" demo\_\*

demo\_file:this line is the 1st lower case line in this file.

demo\_file:Two lines above this line is empty.

demo\_file:And this is the last line.

demo\_file1:this line is the 1st lower case line in this file.

demo\_file1:Two lines above this line is empty.

demo\_file1:And this is the last line.

**3. Case insensitive search using grep -i**

Syntax:

grep -i "string" FILE

This is also a basic usage of the grep. This searches for the given string/pattern case insensitively. So it matches all the words such as “the”, “THE” and “The” case insensitively as shown below.

$ grep -i "the" demo\_file

THIS LINE IS THE 1ST UPPER CASE LINE IN THIS FILE.

this line is the 1st lower case line in this file.

This Line Has All Its First Character Of The Word With Upper Case.

And this is the last line.

**4. Match regular expression in files**

Syntax:

grep "REGEX" filename

This is a very powerful feature, if you can use use regular expression effectively. In the following example, it searches for all the pattern that starts with “lines” and ends with “empty” with anything in-between. i.e To search “lines[anything in-between]empty” in the demo\_file.

$ grep "lines.\*empty" demo\_file

Two lines above this line is empty.

From documentation of grep: A regular expression may be followed by one of several repetition operators:

* ? The preceding item is optional and matched at most once.
* \* The preceding item will be matched zero or more times.
* + The preceding item will be matched one or more times.
* {n} The preceding item is matched exactly n times.
* {n,} The preceding item is matched n or more times.
* {,m} The preceding item is matched at most m times.
* {n,m} The preceding item is matched at least n times, but not more than m times.

**5. Checking for full words, not for sub-strings using grep -w**

If you want to search for a word, and to avoid it to match the substrings use -w option. Just doing out a normal search will show out all the lines.  
  
The following example is the regular grep where it is searching for “is”. When you search for “is”, without any option it will show out “is”, “his”, “this” and everything which has the substring “is”.

$ grep -i "is" demo\_file

THIS LINE IS THE 1ST UPPER CASE LINE IN THIS FILE.

this line is the 1st lower case line in this file.

This Line Has All Its First Character Of The Word With Upper Case.

Two lines above this line is empty.

And this is the last line.

The following example is the WORD grep where it is searching only for the word “is”. Please note that this output does not contain the line “This Line Has All Its First Character Of The Word With Upper Case”, even though “is” is there in the “This”, as the following is looking only for the word “is” and not for “this”.

$ grep -iw "is" demo\_file

THIS LINE IS THE 1ST UPPER CASE LINE IN THIS FILE.

this line is the 1st lower case line in this file.

Two lines above this line is empty.

And this is the last line.

**6. Displaying lines before/after/around the match using grep -A, -B and -C**

When doing a grep on a huge file, it may be useful to see some lines after the match. You might feel handy if grep can show you not only the matching lines but also the lines after/before/around the match.

Please create the following demo\_text file for this example.

$ cat demo\_text

4. Vim Word Navigation

You may want to do several navigation in relation to the words, such as:

\* e - go to the end of the current word.

\* E - go to the end of the current WORD.

\* b - go to the previous (before) word.

\* B - go to the previous (before) WORD.

\* w - go to the next word.

\* W - go to the next WORD.

WORD - WORD consists of a sequence of non-blank characters, separated with white space.

word - word consists of a sequence of letters, digits and underscores.

Example to show the difference between WORD and word

\* 192.168.1.1 - single WORD

\* 192.168.1.1 - seven words.

**6.1 Display N lines after match**

-A is the option which prints the specified N lines after the match as shown below.

Syntax:

grep -A <N> "string" FILENAME

The following example prints the matched line, along with the 3 lines after it.

$ grep -A 3 -i "example" demo\_text

Example to show the difference between WORD and word

\* 192.168.1.1 - single WORD

\* 192.168.1.1 - seven words.

**6.2 Display N lines before match**

-B is the option which prints the specified N lines before the match.

Syntax:

grep -B <N> "string" FILENAME

When you had option to show the N lines after match, you have the -B option for the opposite.

$ grep -B 2 "single WORD" demo\_text

Example to show the difference between WORD and word

\* 192.168.1.1 - single WORD

**6.3 Display N lines around match**

-C is the option which prints the specified N lines before the match. In some occasion you might want the match to be appeared with the lines from both the side. This options shows N lines in both the side(before & after) of match.

$ grep -C 2 "Example" demo\_text

word - word consists of a sequence of letters, digits and underscores.

Example to show the difference between WORD and word

\* 192.168.1.1 - single WORD

**7. Highlighting the search using GREP\_OPTIONS**

As grep prints out lines from the file by the pattern / string you had given, if you wanted it to highlight which part matches the line, then you need to follow the following way.  
  
When you do the following export you will get the highlighting of the matched searches. In the following example, it will highlight all the this when you set the GREP\_OPTIONS environment variable as shown below.

$ export GREP\_OPTIONS='--color=auto' GREP\_COLOR='100;8'

$ grep this demo\_file

**this** line is the 1st lower case line in this file.

Two lines above **this** line is empty.

And **this** is the last line.

**8. Searching in all files recursively using grep -r**

When you want to search in all the files under the current directory and its sub directory. -r option is the one which you need to use. The following example will look for the string “ramesh” in all the files in the current directory and all it’s subdirectory.

$ grep -r "ramesh" \*

**9. Invert match using grep -v**

You had different options to show the lines matched, to show the lines before match, and to show the lines after match, and to highlight match. So definitely You’d also want the option -v to do invert match.  
  
When you want to display the lines which does not matches the given string/pattern, use the option -v as shown below. This example will display all the lines that did not match the word “go”.

$ grep -v "go" demo\_text

4. Vim Word Navigation

You may want to do several navigation in relation to the words, such as:

WORD - WORD consists of a sequence of non-blank characters, separated with white space.

word - word consists of a sequence of letters, digits and underscores.

Example to show the difference between WORD and word

\* 192.168.1.1 - single WORD

\* 192.168.1.1 - seven words.

**10. display the lines which does not matches all the given pattern.**

Syntax:

grep -v -e "pattern" -e "pattern"

$ cat test-file.txt

a

b

c

d

$ grep -v -e "a" -e "b" -e "c" test-file.txt

d

**11. Counting the number of matches using grep -c**

When you want to count that how many lines matches the given pattern/string, then use the option -c.

Syntax:

grep -c "pattern" filename

$ grep -c "go" demo\_text

6

When you want do find out how many lines matches the pattern

$ grep -c this demo\_file

3

When you want do find out how many lines that does not match the pattern

$ grep -v -c this demo\_file

4

**12. Display only the file names which matches the given pattern using grep -l**

If you want the grep to show out only the file names which matched the given pattern, use the -l (lower-case L) option.  
  
When you give multiple files to the grep as input, it displays the names of file which contains the text that matches the pattern, will be very handy when you try to find some notes in your whole directory structure.

$ grep -l this demo\_\*

demo\_file

demo\_file1

**13. Show only the matched string**

By default grep will show the line which matches the given pattern/string, but if you want the grep to show out only the matched string of the pattern then use the -o option.  
  
It might not be that much useful when you give the string straight forward. But it becomes very useful when you give a regex pattern and trying to see what it matches as

$ grep -o "is.\*line" demo\_file

is line is the 1st lower case line

is line

is is the last line

**14. Show the position of match in the line**

When you want grep to show the position where it matches the pattern in the file, use the following options as

Syntax:

grep -o -b "pattern" file

$ cat temp-file.txt

12345

12345

$ grep -o -b "3" temp-file.txt

2:3

8:3

**Note:** The output of the grep command above is not the position in the line, it is byte offset of the whole file.

**15. Show line number while displaying the output using grep -n**

To show the line number of file with the line matched. It does 1-based line numbering for each file. Use -n option to utilize this feature.

$ grep -n "go" demo\_text

5: \* e - go to the end of the current word.

6: \* E - go to the end of the current WORD.

7: \* b - go to the previous (before) word.

8: \* B - go to the previous (before) WORD.

9: \* w - go to the next word.

10: \* W - go to the next WORD.

# Cut Command in Unix ( Linux) Examples

Cut command in unix (or linux) is used to select sections of text from each line of files. You can use the cut command to select fields or columns from a line by specifying a delimiter or you can select a portion of text by specifying the range or characters. Basically the cut command slices a line and extracts the text.  
  
**Unix Cut Command Example**  
  
We will see the usage of cut command by considering the below text file as an example

> cat file.txt

unix or linux os

is unix good os

is linux good os

**1.** Write a unix/linux cut command to print characters by position?  
  
The cut command can be used to print characters in a line by specifying the position of the characters. To print the characters in a line, use the -c option in cut command

cut -c4 file.txt

x

u

l

The above cut command prints the fourth character in each line of the file. You can print more than one character at a time by specifying the character positions in a comma separated list as shown in the below example

cut -c4,6 file.txt

xo

ui

ln

This command prints the fourth and sixth character in each line.  
  
**2.**Write a unix/linux cut command to print characters by range?  
  
You can print a range of characters in a line by specifying the start and end position of the characters.

cut -c4-7 file.txt

x or

unix

linu

The above cut command prints the characters from fourth position to the seventh position in each line. To print the first six characters in a line, omit the start position and specify only the end position.

cut -c-6 file.txt

unix o

is uni

is lin

To print the characters from tenth position to the end, specify only the start position and omit the end position.

cut -c10- file.txt

inux os

ood os

good os

If you omit the start and end positions, then the cut command prints the entire line.

cut -c- file.txt

**3.**Write a unix/linux cut command to print the fields using the delimiter?  
  
You can use the cut command just as awk command to extract the fields in a file using a delimiter. The -d option in cut command can be used to specify the delimiter and -f option is used to specify the field position.

cut -d' ' -f2 file.txt

or

unix

linux

This command prints the second field in each line by treating the space as delimiter. You can print more than one field by specifying the position of the fields in a comma delimited list.

cut -d' ' -f2,3 file.txt

or linux

unix good

linux good

The above command prints the second and third field in each line.  
  
**Note:** If the delimiter you specified is not exists in the line, then the cut command prints the entire line. To suppress these lines use the -s option in cut command.  
  
**4.** Write a unix/linux cut command to display range of fields?  
  
You can print a range of fields by specifying the start and end position.

cut -d' ' -f1-3 file.txt

The above command prints the first, second and third fields. To print the first three fields, you can ignore the start position and specify only the end position.

cut -d' ' -f-3 file.txt

To print the fields from second fields to last field, you can omit the last field position.

cut -d' ' -f2- file.txt

**5.** Write a unix/linux cut command to display the first field from /etc/passwd file?  
  
The /etc/passwd is a delimited file and the delimiter is a colon (:). The cut command to display the first field in /etc/passwd file is

cut -d':' -f1 /etc/passwd

**6.** The input file contains the below text

> cat filenames.txt

logfile.dat

sum.pl

add\_int.sh

Using the cut command extract the portion after the dot.  
  
First reverse the text in each line and then apply the command on it.

rev filenames.txt | cut -d'.' -f1

"**tar**" stands for *tape archive*. It is an archiving [file format](http://www.computerhope.com/jargon/f/fileform.htm).  
  
**tar** was originally developed in the early days of [Unix](http://www.computerhope.com/jargon/u/unix.htm) for the purpose of backing up files to [tape](http://www.computerhope.com/jargon/t/tape.htm)-based [storage devices](http://www.computerhope.com/jargon/s/stordevi.htm). It was later formalized as part of the [POSIX](http://www.computerhope.com/jargon/p/posix.htm) standard, and today is used to collect, distribute, and archive files, while preserving [file system](http://www.computerhope.com/jargon/f/filesyst.htm) [attributes](http://www.computerhope.com/jargon/a/attribut.htm) such as [user](http://www.computerhope.com/jargon/u/user.htm) and [group](http://www.computerhope.com/jargon/g/group.htm) [permissions](http://www.computerhope.com/jargon/p/permissi.htm), [access](http://www.computerhope.com/jargon/a/access.htm) and [modification](http://www.computerhope.com/jargon/m/modify.htm) dates, and [directory structures](http://www.computerhope.com/jargon/h/hierfile.htm).   
  
This documentation covers the [GNU](http://www.computerhope.com/jargon/g/gnu.htm) version of **tar**, which is included with most modern variants of the [Linux](http://www.computerhope.com/jargon/l/linux.htm) [operating system](http://www.computerhope.com/os.htm).

## Operation

The first [argument](http://www.computerhope.com/jargon/a/argument.htm) to **tar** should be a *function* specification: either one of the letters **A**, **c**, **d**, **r**, **t**, **u**, or **x**, or one of the long function names. A function letter does not need to be prefixed with a dash ("**-**"), and may be combined with other single-letter options. A long function name must be prefixed with a double dash ("**--**"). Some options take a [parameter](http://www.computerhope.com/jargon/p/paramete.htm); with the single-letter form these must be given as separate arguments. With the long form, they may be given by appending "**=***value*" to the option.  
  
For example, the following commands are all equivalent:

tar --create --file=archive.tar file1 file2

tar -c -f archive.tar file1 file2

tar -cf archive.tar file1 file2

tar cf archive.tar file1 file2

## Functions

Specifying one of the following functions selects what **tar**'s main mode of operation will be:

|  |  |
| --- | --- |
| **A**, **--catenate**, **--concatenate** | [Append](http://www.computerhope.com/jargon/a/append.htm) **tar** files to an archive. |
| **c**, **--create** | Create a new archive. |
| **d**, **--diff**, **--compare** | Calculate any differences between the archive and the file system. |
| **--delete** | Delete from the archive. (This function doesn't work on magnetic tapes). |
| **r**, **--append** | Append files to the end of a **tar** archive. |
| **t**, **--list** | List the contents of an archive. |
| **--test-label** | Test the archive label, and exit. |
| **u**, **--update** | Append files, but only those that are newer than the copy in the archive. |
| **x**, **--extract**, **--get** | Extract files from an archive. |

## Editing A File

The most common way to start a **vi** session is to tell it which file to edit. To edit a file named **filename**, use the command:

vi filename

The screen will clear and the text of your file will appear on the screen. If **filename** doesn't exist yet, **vi** will start you in a new file, and when you tell it to save your work, it will use the **filename** that you specified.

## The Editor's Copy: The "Buffer"

The editor does not directly modify the file you are editing. Instead, it makes a copy of this file in memory called the *buffer*. You do not actually affect the contents of the file until you write the changes you've made back into the original file.

## Arrow Keys

On most terminals, you can use the arrow keys to move the cursor around. Left and right moves the cursor left or right one character, and up and down move the cursor up or down one line. The other way to move the cursor is with the **h**, **j**, **k**, and **l** keys:

|  |  |
| --- | --- |
| **h** | left |
| **j** | down |
| **k** | up |
| **l** | right |

A bit of historical trivia: the **hjkl** keys are used because the original computer system Bill Joy developed with did not have separate arrow keys. Instead, the keyboard used **hjkl** keys as the arrow keys.  
  
Hardcore **vi** users will tell you that the **hjkl** keys are the only "true" way to move the cursor because you can keep your hands in the typing position, but you should use whatever you're comfortable with.

## Special Keys: Esc, Return, and Delete

The Escape key is very important in **vi**: it is used to cancel any command you've started typing, and to return to command-mode after you've been inserting or adding text. Escape is a safe key to hit if you've lost track of what mode you're in or what you're doing. You can hit it several times to return to command mode and cancel anything that you've been doing.  
  
The Return (or "Enter") key is used to complete certain commands and execute them.  
  
The Delete key is another "safe" key to press. It interrupts anything that the editor has been doing and brings you back to the default state of the editor, sometimes also called the "quiescent" state.

## Exiting vi

The command **ZZ** will write the contents of the buffer into the file you are editing, and quit **vi**.  
  
If there are no changes to be saved, you can also quit the editor with the command **:q**. If you have made changes that haven't been saved yet, the editor won't let you exit this way.  
  
If you want to quit and discard the changes you've made, you can tell the editor you're sure you know what you're doing by quitting with the command **:q!**. This will exit **vi** immediately, and any unsaved changes will be lost.

# Moving Around In Your File

## Scrolling And Paging

The editor has several commands for moving around within your file. One of the most useful of these is **Control-D** (also represented as **^D**). This command "scrolls" down in the file (the **D** stands for "down"). **^U** ("up") is similar: it scrolls up one page in the file. Both of these commands will also move the cursor to the new location.  
  
If you just want to see more of the file, but leave the cursor where it is, you can use **^E** to "expose" one more line at the bottom of the screen. **^Y** (which doesn't stand for anything, but it's next to the **U** on the keyboard) exposes one line at the top of the screen.  
  
There are other ways to move around in the file. The keys **^F** and **^B** move forward and backward one page (this is called "paging"), keeping a couple of lines of continuity between screens so that it is possible to read through a file using these rather than **^D** and **^U** if you wish.  
  
Notice the difference between "scrolling" (**^U**, **^D**) and "paging" (**^B**, **^F**). Scrolling leaves more context, but paging only leaves you a couple of lines. It's up to you which one is easier for moving around and reading your file.

## Searching

Another way to position yourself in the file is by giving the editor a [string](http://www.computerhope.com/jargon/s/string.htm) to search for. If you type "**/**" followed by a string of characters and press Enter, the editor will search for the next occurrence of this string in your file, and place the cursor there. Pressing **n** will go to the next occurrence after that.  
  
If instead of forward, you want to search backward for a string, use **?** instead of **/**. In a **?** string search, pressing **n** will take you to successive occurrences in the same (backward) direction.  
  
If the search string you give the editor is not present in the file, the editor will print a message letting you know it couldn't find it, and your cursor will not be moved.  
  
If you want your search string only to match if it's at the beginning of a line, you should begin the search string with a carat ("**^**"). To match only at the end of a line, end the search string with a dollar sign ("**$**"). So using the command

/^mystring

will only find a match if "**mystring**" is at the very begining of a line. Similarly, the command

/mystring$

will only match any occurrences of "**mystring**" that are at the very end of a line.

## "Goto"-ing

The command **G**, when preceded by a number, will "goto" a line: it will go to that line number and position the cursor there. If you simply use **G** with no number, it will move the cursor to the last line of the file.  
  
So, using the command

1G

Will move your cursor to the first line in the file;

30G

Will move your cursor to line 30, and

G

Will move your cursor to the last line, whatever number that may be.

## Finding Out Where You Are In The File

You can find out where you're at in the file by pressing **^G**. This sill show you the name of the file you are editing, the number of the current line, the number of lines in the buffer, and the percentage of the way through the buffer your cursor is currently at.

## Getting Back To Where You Just Were

Any time you move the cursor in any of these ways, you can move the cursor back to where it just was by typing **``** (two [back quotes](http://www.computerhope.com/jargon/b/backquot.htm)). So, if you type

G

to move to the end of the file, but then you realize you need to get back to where you were (whether you remember where that is or not), you can type

``

To get back to your original location. This also works after performing a search. Remember this command, because it's very useful!

## Lines Past The End Of Your File

If you're near the end of your file, and the last line is not at the bottom of the screen, **vi** will print a [tilde](http://www.computerhope.com/jargon/t/tilde.htm) ("**~**") on any lines after the end of your actual file. This indicates that the last line of your file is on the screen; that is, any lines showing "**~**" are past the end of your file. You'll notice this, for instance, when you start editing a new file: every line will begin with a tilde.

## Moving Around On The Screen

As we've already mentioned, you can use the arrow keys or **h**, **j**, **k**, and **l** to move the cursor left and right one cursor, or up and down one line.  
  
You can also use **+** to move to the next line or **-** to move to the previous line. This is similar to using **up** and **down** (or **j** and **k**), except that your cursor will automatically be placed at the first non-[whitespace](http://www.computerhope.com/jargon/w/whitspac.htm) character on the line.  
  
Any of these keys, if they would take you before the first line on the screen or after the last line, will scroll the display one line in the correct direction. Pressing the **Return** key is the same as pressing **+**. You can also position the cursor at the top, middle, or bottom of the currently-displayed screen of text:  
  
**H** will take you to the top ("home") line on the screen. Preceding **H** with a number will take you to the home line plus that number of lines. For instance, **3H** will move your cursor to the third line of text displayed on the screen.  
  
**M** is a similar command: it moves your cursor to the middle of the screen.  
  
**L** moves your cursor to the last line of the screen. If you precede it with a number, it will move your cursor to that number of lines before the last line displayed. So **5L** will move your cursor to the fifth-to-last line currently displayed.

## Moving Within A Line

**w** will advance the cursor to the next word on the line, and **b** will back up the cursor to the previous word.  
  
**e** advances you to the end of the current word rather than the beginning of the next word.  
  
If words are punctuated, for instance with an apostrophe or a comma, **w** and **b** will stop at the punctuation. If you use **W** and **B** instead, they will move the cursor while ignoring punctuation.  
  
These commands all wrap at the end (or beginning) of a line, moving to the previous or next line if their search for the next or previous word goes that far.

## Summary: Movement Keys

Here's a recap of the movement keys we just discussed, and a few extras.

|  |  |
| --- | --- |
| SPACE BAR | advance the cursor one position |
| **^B** | page backward |
| **^D** | scroll down |
| **^E** | expose a line at the bottom of the screen |
| **^F** | page forward |
| **^G** | Positional status check |
| **^N** | next line, same column |
| **^P** | previous line, same column |
| **^U** | scroll up |
| **^Y** | expose a line at the top of the screen |
| **+** | next line, at the first non-whitespace character |
| **-** | previous line, at the first non-whitespace character |
| **/***string* | Scan forward for *string*. If no *string* is specified, scans for the last *string* scanned for. |
| **?** | Scan backwards for *string*. If no *string* is specified, scans for the last *string* scanned for. |
| **B** | backwards one word, ignoring punctuation |
| *linenumber***G** | Go to specified *linenumber*. Defaults to the last line. |
| **H** | move cursor to first line of screen |
| **M** | middle line of screen |
| **L** | last line of screen |
| **W** | forward a word, ignoring punctuation |
| **b** | backwards a word |
| **e** | end of current word |
| **n** | next instance of **/** or **?** search pattern |
| **w** | next word |

# Making Simple Changes

The most basic command for making changes to your file is the *i* (insert) command. After you type *i*, everything you type until you hit **ESC** is inserted at your cursor position into the buffer.  
  
The companion to the *i* command is the **a** ("append") command. This works exactly like *i*, except it starts inserting the text *after* the cursor location, not *at* the cursor location.   
  
Sometimes, you want to start editing on a new line right after the line your cursor is at. To do this, use the *o* command. To start a new line *before* the line your cursor is at, use the **O** command. Just like with *i* and *a*, any text you type will be inserted into the buffer until you press **ESC**.  
  
While you are inserting text with any of these commands, you can use **^W** to erase a whole word and leave your cursor one space after the previous word.  
One thing to note is that when you backspace, you will not erase any characters that you haven't inserted. Also, you can't backspace to the previous line; backspacing will stop when you get to the beginning of the current line.

## Making Small Corrections

It's easy to make small corrections in **vi**. If you need to remove a character that shouldn't be there, move the cursor over it and press **x** to delete it. You can also supply a number before pressing **x**, and it will delete that many characters. So if you have the word "**discovery**", and you move the cursor over the **v** and type

4x

...you will be left with the word "**disco**".  
  
If you want to replace a single character with another character, move the cursor over it and press **r** and then the character that should replace it. So, if you type "**mistoke**", and move the cursor over the "**o**" and type

ra

...the **o** will be replaced with an **a**, leaving you with the word "**mistake**."  
  
The **R** command is similar, but you can use it to replace more than one character; if you press **R** and start typing, every character you type will replace every character that was already there. So, if you have the word "**mistake**", and you move the cursor over the letter **t** and type

Rsile

...you will see that the word "**mistake**" has turned into the word "**missile**". You might also notice that if you backspace in this mode, your original text will be restored, letter by letter. When you're done, press **ESC** to stop replacing characters.

## Operators

The command **d** acts as a *delete operator*: presing **d** and then a movement key will delete wherever the cursor would have moved. For instance, typing **dw** will delete the next word, and typing **db** will delete backwards to the previous word. Typing **d**SPACE will delete one character; this is the same as typing **x**.  
  
Another useful operator is **c**, which stands for *change*. The command **cw** will change a single word: it will delete everything to the end of a word, and place you into insert mode to type the text that should replace it.

## Repeating A Change

Any time you make a change with a command, you can type "**.**" to repeat the change. So if you delete a word with **dw**, typing **.** will delete the next word as well. You can repeat this over and over by typing **.** several times, like an ellipsis ("**...**").

## Operating On Lines

Typing **dd** will delete an entire line.  
  
Typing **cc** will change an entire line, deleting it and placing you in insert mode.  
  
You can delete or change more than one line at a time by preceding **dd** or **cc** with a number. For instance, **5dd** will delete 5 lines. You can also give a command like **dL** which will delete every line from the cursor up to and including the last line on the screen, or **d3L** which will delete every line from the cursor to the third-to-last line. When you make big changes like this, especially if you make changes that go beyond what you can see in one screen, the editor will give you a message telling you what's been done.

## Undoing

To undo the last change, type **u**. To re-do (to undo an undo), type **u** again.  
  
Unlike [**vim**](http://www.computerhope.com/jargon/v/vim.htm), **vi** only offers one level of undo. However, you can use **U** (capital U) to undo any changes that were made to the current line; pressing **U** restores the current line to the way it was before you started changing it.

## Summary: Making Changes

|  |  |
| --- | --- |
| SPACE | advance the cursor one position |
| **^W** | erase a word during insert |
| **.** | repeats the last change command |
| **O** | opens and inputs a new line, above the current line |
| **U** | undoes the changes you made to the current line |
| **a** | appends text after the cursor |
| **c** | changes the object you specify to the text you type in after |
| **d** | deletes the object that you specify |
| **i** | inserts text before the cursor |
| **o** | opens and inputs new lines, below the current line |
| **u** | undoes the last change |

# SSH Command Examples - Unix / Linux Tutorials

SSH client utility in unix or linux server is used to logging into a remote host and execute commands on the remote machine. The rlogin and rsh commands can also be used to login into the remote machine. However these are not secure. The ssh command provides a secure connection between two hosts over a insecure network.  
  
The syntax ssh command is

ssh [-l username] hostname | user@remote-hostname [command]

Let see the examples of ssh command.  
  
**SSH Command Examples**:  
  
1. Logging to a remote server  
  
You can login to a remote server from the local host as shown below:

localhost:[~]> ssh -l username remote-server

username@remote-server password:

remote-server:[~]>

Alternatively you can use the below ssh command for connecting to remote host:

localhost:[~]> ssh username@remote-server

username@remote-server password:

remote-server:[~]>

**Note**: If you are logging for the first time, then it will prints a message that host key not found and you can give yes to continue. The host key of the remote server will be cached and added to the .ssh2/hostkeys directory in your home directory. From second time onwards you just need to enter the password.  
  
2. Logging out from remote server  
  
Simply enter the exit command on the terminal to close the connection. This is shown below:

remote-server:[~]>exit

logout

Connection to remote-server closed.

localhost:[~]>

3. Running remote commands from local host  
  
Sometimes it is necessary to run the unix commands on the remote server from the local host. An example is shown below:

localhost:[~]> ssh user@remote-host "ls test"

online-backup.dat

oracle-storage.bat

unix-dedicated-server.txt

The ssh command connects to the remote host, runs the ls command, prints the output on the local host terminal and exits the connection from remote host.  
  
Let see whether the ls command actually displayed the correct result or not by connecting to the remote host.

localhost:[~]> ssh user@remote-host

user@remotehost password:

remotehost:[~]> cd test

remotehost:[~/test]> ls

online-backup.dat

oracle-storage.bat

unix-dedicated-server.txt

4. Version of the SSH command  
  
We can find the version of SSH installed on the unix system using the -V option to the ssh. This is shown below:

> ssh -V

OpenSSH\_4.3p2, OpenSSL 0.9.8e-fips-rhel5 01 Jul 2008

5. Debugging the SSH Client  
  
When we are not able to connect to the remote host, it is good to debug and find the exact error messages that causing the issue. Use the -v option for debugging the ssh client.

ssh -v user@remote-host

OpenSSH\_4.3p2, OpenSSL 0.9.8e-fips-rhel5 01 Jul 2008

debug1: Reading configuration data /etc/ssh/ssh\_config

debug1: Applying options for \*

debug1: Connecting to remote-host [172.22.200.140] port 22.

debug1: Connection established.

debug1: identity file /home/user/.ssh/identity type -1

debug1: identity file /home/user/.ssh/id\_rsa type -1

debug1: identity file /home/user/.ssh/id\_dsa type 2

debug1: loaded 3 keys

..........

..........

6. Copying files between remote host and local host.  
  
We can use the scp command to copy the files securely between the local host and remote host using the ssh authentication.  
  
To copy the file from local host to remote hosts /var/tmp/ directory, run the below scp command.

scp filename user@remote-host:/var/tmp/

To copy the file from remote hosts /usr/local/bin/ directory to local hosts current directory, run the below scp command.

scp user@remote-host:/usr/local/bin/add.sh .

# Sed Command in Unix and Linux Examples

Sed is a Stream Editor used for modifying the files in unix (or linux). Whenever you want to make changes to the file automatically, sed comes in handy to do this. Most people never learn its power; they just simply use sed to replace text. You can do many things apart from replacing text with sed. Here I will describe the features of sed with examples.  
  
Consider the below text file as an input.

>cat file.txt

unix is great os. unix is opensource. unix is free os.

learn operating system.

unixlinux which one you choose.

### Sed Command Examples

**1.** Replacing or substituting string  
  
Sed command is mostly used to replace the text in a file. The below simple sed command replaces the word "unix" with "linux" in the file.

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

linux is great os. unix is opensource. unix is free os.

learn operating system.

linuxlinux which one you choose.

Here the "s" specifies the substitution operation. The "/" are delimiters. The "unix" is the search pattern and the "linux" is the replacement string.  
  
By default, the sed command replaces the first occurrence of the pattern in each line and it won't replace the second, third...occurrence in the line.  
  
**2.** Replacing the nth occurrence of a pattern in a line.  
  
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

unix is great os. linux is opensource. unix is free os.

learn operating system.

unixlinux which one you choose.

**3.** Replacing all the occurrence of the pattern in a line.  
  
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

linux is great os. linux is opensource. linux is free os.

learn operating system.

linuxlinux which one you choose.

**4.** Replacing from nth occurrence to all occurrences in a line.  
  
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

unix is great os. unix is opensource. linux is free os.

learn operating system.

unixlinux which one you choose.

**5.** Changing the slash (/) delimiter  
  
You can use any delimiter other than the slash. As an example if you want to change the web url to another url as

>sed 's/http:\/\//www/' file.txt

In this case the url consists the delimiter character which we used. In that case you have to escape the slash with backslash character, otherwise the substitution won't work.  
  
Using too many backslashes makes the sed command look awkward. In this case we can change the delimiter to another character as shown in the below example.

>sed 's\_http://\_www\_' file.txt

>sed 's|http://|www|' file.txt

**6.** Using & as the matched string  
  
There might be cases where you want to search for the pattern and replace that pattern by adding some extra characters to it. In such cases & comes in handy. The & represents the matched string.

>sed 's/unix/{&}/' file.txt

{unix} is great os. unix is opensource. unix is free os.

learn operating system.

{unix}linux which one you choose.

>sed 's/unix/{&&}/' file.txt

{unixunix} is great os. unix is opensource. unix is free os.

learn operating system.

{unixunix}linux which one you choose.

**7.** Using \1,\2 and so on to \9  
  
The first pair of parenthesis specified in the pattern represents the \1, the second represents the \2 and so on. The \1,\2 can be used in the replacement string to make changes to the source string. As an example, if you want to replace the word "unix" in a line with twice as the word like "unixunix" use the sed command as below.

>sed 's/\(unix\)/\1\1/' file.txt

unixunix is great os. unix is opensource. unix is free os.

learn operating system.

unixunixlinux which one you choose.

The parenthesis needs to be escaped with the backslash character. Another example is if you want to switch the words "unixlinux" as "linuxunix", the sed command is

>sed 's/\(unix\)\(linux\)/\2\1/' file.txt

unix is great os. unix is opensource. unix is free os.

learn operating system.

linuxunix which one you choose.

Another example is switching the first three characters in a line

>sed 's/^\(.\)\(.\)\(.\)/\3\2\1/' file.txt

inux is great os. unix is opensource. unix is free os.

aelrn operating system.

inuxlinux which one you choose.

**8.** Duplicating the replaced line with /p flag  
  
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

linux is great os. unix is opensource. unix is free os.

linux is great os. unix is opensource. unix is free os.

learn operating system.

linuxlinux which one you choose.

linuxlinux which one you choose.

**9.** 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

linux is great os. unix is opensource. unix is free os.

linuxlinux which one you choose.

If you use -n alone without /p, then the sed does not print anything.  
  
**10.** 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/'

linux is great system. unix is opensource. unix is free os.

learn operating system.

linuxlinux which one you chosysteme.

Sed provides -e option to run multiple sed commands in a single sed command. The above output can be achieved in a single sed command as shown below.

>sed -e 's/unix/linux/' -e 's/os/system/' file.txt

linux is great system. unix is opensource. unix is free os.

learn operating system.

linuxlinux which one you chosysteme.

**11.** 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

unix is great os. unix is opensource. unix is free os.

learn operating system.

linuxlinux which one you choose.

The above sed command replaces the string only on the third line.  
  
**12.** Replacing string on a range of lines.  
  
You can specify a range of line numbers to the sed command for replacing a string.

>sed '1,3 s/unix/linux/' file.txt

linux is great os. unix is opensource. unix is free os.

learn operating system.

linuxlinux which one you choose.

Here the sed command replaces the lines with range from 1 to 3. Another example is

>sed '2,$ s/unix/linux/' file.txt

linux is great os. unix is opensource. unix is free os.

learn operating system.

linuxlinux which one you choose.

Here $ indicates the last line in the file. So the sed command replaces the text from second line to last line in the file.  
  
**13.** Replace on a lines which matches a pattern.  
  
You can specify a pattern to the sed command to match in a line. If the pattern match occurs, then only the sed command looks for the string to be replaced and if it finds, then the sed command replaces the string.

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

unix is great os. unix is opensource. unix is free os.

learn operating system.

centoslinux which one you choose.

Here the sed command first looks for the lines which has the pattern "linux" and then replaces the word "unix" with "centos".  
  
**14.** Deleting lines.  
  
You can delete the lines a file by specifying the line number or a range or numbers.

>sed '2 d' file.txt

>sed '5,$ d' file.txt

**15.** Duplicating lines  
  
You can make the sed command to print each line of a file two times.

>sed 'p' file.txt

**16.** Sed as grep command  
  
You can make sed command to work as similar to grep command.

>grep 'unix' file.txt

>sed -n '/unix/ p' file.txt

Here the sed command looks for the pattern "unix" in each line of a file and prints those lines that has the pattern.  
  
You can also make the sed command to work as grep -v, just by using the reversing the sed with NOT (!).

>grep -v 'unix' file.txt

>sed -n '/unix/ !p' file.txt

The ! here inverts the pattern match.  
  
**17.** Add a line after a match.  
  
The sed command can add a new line after a pattern match is found. The "a" command to sed tells it to add a new line after a match is found.

>sed '/unix/ a "Add a new line"' file.txt

unix is great os. unix is opensource. unix is free os.

"Add a new line"

learn operating system.

unixlinux which one you choose.

"Add a new line"

**18.** Add a line before a match  
  
The sed command can add a new line before a pattern match is found. The "i" command to sed tells it to add a new line before a match is found.

>sed '/unix/ i "Add a new line"' file.txt

"Add a new line"

unix is great os. unix is opensource. unix is free os.

learn operating system.

"Add a new line"

unixlinux which one you choose.

**19.** Change a line  
  
The sed command can be used to replace an entire line with a new line. The "c" command to sed tells it to change the line.

>sed '/unix/ c "Change line"' file.txt

"Change line"

learn operating system.

"Change line"

**20.** Transform like tr command  
  
The sed command can be used to convert the lower case letters to upper case letters by using the transform "y" option.

>sed 'y/ul/UL/' file.txt

Unix is great os. Unix is opensoUrce. Unix is free os.

Learn operating system.

UnixLinUx which one yoU choose.

Here the sed command transforms the alphabets "ul" into their uppercase format "UL"

**. Display timestamp using HISTTIMEFORMAT**

Typically when you type history from command line, it displays the command# and the command. For auditing purpose, it may be beneficial to display the timepstamp along with the command as shown below.

# **export HISTTIMEFORMAT='%F %T '**

# **history | more**

1 2008-08-05 19:02:39 service network restart

2 2008-08-05 19:02:39 exit

3 2008-08-05 19:02:39 id

4 2008-08-05 19:02:39 cat /etc/redhat-release

**2. Search the history using Control+R**

I strongly believe, this may be your most frequently used feature of history. When you’ve already executed a very long command, you can simply search history using a keyword and re-execute the same command without having to type it fully. **Press Control+R and type the keyword**. In the following example, I searched for **red**, which displayed the previous command “**cat /etc/redhat-release**” in the history that contained the word red.

# [Press **Ctrl+R** from the command prompt,

which will display the reverse-i-search prompt]

(reverse-i-search)`**red**': cat /etc/redhat-release

[Note: Press **enter when you see your command**,

which will execute the command from the history]

# cat /etc/redhat-release

Fedora release 9 (Sulphur)

Sometimes you want to edit a command from history before executing it. For e.g. you can search for **httpd**, which will display **service httpd stop** from the command history, select this command and **change the stop to start** and re-execute it again as shown below.

# [Press **Ctrl+R** from the command prompt,

which will display the reverse-i-search prompt]

(reverse-i-search)`**httpd**': service httpd stop

[Note: Press **either left arrow or right arrow** key when you see your

command, which will display the command for you to edit, before executing it]

# service httpd start

**3. Repeat previous command quickly using 4 different methods**

Sometime you may end up repeating the previous commands for various reasons. Following are the 4 different ways to repeat the last executed command.

1. Use the **up arrow** to view the previous command and press enter to execute it.
2. Type **!!** and press enter from the command line
3. Type **!-1** and press enter from the command line.
4. Press **Control+P** will display the previous command, press enter to execute it

**4. Execute a specific command from history**

In the following example, If you want to repeat the command #4, you can do **!4** as shown below.

# **history | more**

1 service network restart

2 exit

3 id

4 cat /etc/redhat-release

# **!4**

cat /etc/redhat-release

Fedora release 9 (Sulphur)

**5. Execute previous command that starts with a specific word**

Type ! followed by the starting few letters of the command that you would like to re-execute. In the following example, typing !ps and enter, executed the previous command starting with ps, which is ‘ps aux | grep yp’.

# **!ps**

ps aux | grep yp

root 16947 0.0 0.1 36516 1264 ? Sl 13:10 0:00 ypbind

root 17503 0.0 0.0 4124 740 pts/0 S+ 19:19 0:00 grep yp

**6. Control the total number of lines in the history using HISTSIZE**

Append the following two lines to the .bash\_profile and relogin to the bash shell again to see the change. In this example, only 450 command will be stored in the bash history.

# **vi ~/.bash\_profile**

HISTSIZE=450

HISTFILESIZE=450

**7. Change the history file name using HISTFILE**

By default, history is stored in **~/.bash\_history** file. Add the following line to the .bash\_profile and relogin to the bash shell, to store the history command in .commandline\_warrior file instead of .bash\_history file. I’m yet to figure out a practical use for this. I can see this getting used when you want to track commands executed from different terminals using different history file name.

# **vi ~/.bash\_profile**

HISTFILE=/root/.commandline\_warrior

If you have a good reason to change the name of the history file, please share it with me, as I’m interested in finding out how you are using this feature.

**8. Eliminate the continuous repeated entry from history using HISTCONTROL**

In the following example pwd was typed three times, when you do history, you can see all the 3 continuous occurrences of it. To eliminate duplicates, set HISTCONTROL to ignoredups as shown below.

# **pwd**

# **pwd**

# **pwd**

# **history | tail -4**

44 pwd

45 pwd

46 pwd [Note that there are **three pwd** commands in history, after

executing pwd 3 times as shown above]

47 history | tail -4

# **export HISTCONTROL=ignoredups**

# **pwd**

# **pwd**

# **pwd**

# **history | tail -3**

56 export HISTCONTROL=ignoredups

57 pwd [Note that there is only **one pwd** command in the history, even after

executing pwd 3 times as shown above]

58 history | tail -4

**9. Erase duplicates across the whole history using HISTCONTROL**

The ignoredups shown above removes duplicates only if they are consecutive commands. To eliminate duplicates across the whole history, set the HISTCONTROL to erasedups as shown below.

# **export HISTCONTROL=erasedups**

# **pwd**

# **service httpd stop**

# **history | tail -3**

38 pwd

39 service httpd stop

40 history | tail -3

# **ls -ltr**

# **service httpd stop**

# **history | tail -6**

35 export HISTCONTROL=erasedups

36 pwd

37 history | tail -3

38 ls -ltr

39 service httpd stop

[Note that the previous service httpd stop after pwd got erased]

40 history | tail -6

**10. Force history not to remember a particular command using HISTCONTROL**

When you execute a command, you can instruct history to ignore the command by setting HISTCONTROL to ignorespace AND typing a space in front of the command as shown below. I can see lot of junior sysadmins getting excited about this, as they can hide a command from the history. It is good to understand how ignorespace works. But, as a best practice, don’t hide purposefully anything from history.

# **export HISTCONTROL=ignorespace**

# **ls -ltr**

# **pwd**

#  **service httpd stop** [Note that there is a space at the beginning of service,

to ignore this command from history]

# **history | tail -3**

67 ls -ltr

68 pwd

69 history | tail -3

### 11. Clear all the previous history using option -c

Sometime you may want to clear all the previous history, but want to keep the history moving forward.

# **history -c**

### 12. Subtitute words from history commands

When you are searching through history, you may want to execute a different command but use the same parameter from the command that you’ve just searched.

In the example below, the **!!:$** next to the vi command gets the argument from the previous command to the current command.

# **ls anaconda-ks.cfg**

anaconda-ks.cfg

# **vi !!:$**

vi anaconda-ks.cfg

In the example below, the **!^** next to the vi command gets the first argument from the previous command (i.e cp command) to the current command (i.e vi command).

# **cp anaconda-ks.cfg anaconda-ks.cfg.bak**

anaconda-ks.cfg

# **vi !^**

vi anaconda-ks.cfg

### 13. Substitute a specific argument for a specific command.

In the example below, **!cp:2** searches for the previous command in history that starts with cp and takes the second argument of cp and substitutes it for the ls -l command as shown below.

# **cp ~/longname.txt /really/a/very/long/path/long-filename.txt**

# **ls -l !cp:2**

ls -l /really/a/very/long/path/long-filename.txt

In the example below, **!cp:$** searches for the previous command in history that starts with cp and takes the last argument (in this case, which is also the second argument as shown above) of cp and substitutes it for the ls -l command as shown below.

# **ls -l !cp:$**

ls -l /really/a/very/long/path/long-filename.txt

### 14. Disable the usage of history using HISTSIZE

If you want to disable history all together and don’t want bash shell to remember the commands you’ve typed, set the HISTSIZE to 0 as shown below.

# **export HISTSIZE=0**

# **history**

# [Note that history did not display anything]

### 15. Ignore specific commands from the history using HISTIGNORE

Sometimes you may not want to clutter your history with basic commands such as pwd and ls. Use HISTIGNORE to specify all the commands that you want to ignore from the history. Please note that adding ls to the HISTIGNORE ignores only ls and not ls -l. So, you have to provide the exact command that you would like to ignore from the history.

# **export HISTIGNORE="pwd:ls:ls -ltr:"**

# **pwd**

# **ls**

# **ls -ltr**

# **service httpd stop**

# **history | tail -3**

79 export HISTIGNORE="pwd:ls:ls -ltr:"

80 service httpd stop

81 history

[Note that history did not record pwd, ls and ls -ltr]

# Sed Command in Unix and Linux Examples

Sed is a Stream Editor used for modifying the files in unix (or linux). Whenever you want to make changes to the file automatically, sed comes in handy to do this. Most people never learn its power; they just simply use sed to replace text. You can do many things apart from replacing text with sed. Here I will describe the features of sed with examples.  
  
Consider the below text file as an input.

>cat file.txt

unix is great os. unix is opensource. unix is free os.

learn operating system.

unixlinux which one you choose.

### Sed Command Examples

**1.** Replacing or substituting string  
  
Sed command is mostly used to replace the text in a file. The below simple sed command replaces the word "unix" with "linux" in the file.

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

linux is great os. unix is opensource. unix is free os.

learn operating system.

linuxlinux which one you choose.

Here the "s" specifies the substitution operation. The "/" are delimiters. The "unix" is the search pattern and the "linux" is the replacement string.  
  
By default, the sed command replaces the first occurrence of the pattern in each line and it won't replace the second, third...occurrence in the line.  
  
**2.** Replacing the nth occurrence of a pattern in a line.  
  
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

unix is great os. linux is opensource. unix is free os.

learn operating system.

unixlinux which one you choose.

**3.** Replacing all the occurrence of the pattern in a line.  
  
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

linux is great os. linux is opensource. linux is free os.

learn operating system.

linuxlinux which one you choose.

**4.** Replacing from nth occurrence to all occurrences in a line.  
  
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

unix is great os. unix is opensource. linux is free os.

learn operating system.

unixlinux which one you choose.

**5.** Changing the slash (/) delimiter  
  
You can use any delimiter other than the slash. As an example if you want to change the web url to another url as

>sed 's/http:\/\//www/' file.txt

In this case the url consists the delimiter character which we used. In that case you have to escape the slash with backslash character, otherwise the substitution won't work.  
  
Using too many backslashes makes the sed command look awkward. In this case we can change the delimiter to another character as shown in the below example.

>sed 's\_http://\_www\_' file.txt

>sed 's|http://|www|' file.txt

**6.** Using & as the matched string  
  
There might be cases where you want to search for the pattern and replace that pattern by adding some extra characters to it. In such cases & comes in handy. The & represents the matched string.

>sed 's/unix/{&}/' file.txt

{unix} is great os. unix is opensource. unix is free os.

learn operating system.

{unix}linux which one you choose.

>sed 's/unix/{&&}/' file.txt

{unixunix} is great os. unix is opensource. unix is free os.

learn operating system.

{unixunix}linux which one you choose.

**7.** Using \1,\2 and so on to \9  
  
The first pair of parenthesis specified in the pattern represents the \1, the second represents the \2 and so on. The \1,\2 can be used in the replacement string to make changes to the source string. As an example, if you want to replace the word "unix" in a line with twice as the word like "unixunix" use the sed command as below.

>sed 's/\(unix\)/\1\1/' file.txt

unixunix is great os. unix is opensource. unix is free os.

learn operating system.

unixunixlinux which one you choose.

The parenthesis needs to be escaped with the backslash character. Another example is if you want to switch the words "unixlinux" as "linuxunix", the sed command is

>sed 's/\(unix\)\(linux\)/\2\1/' file.txt

unix is great os. unix is opensource. unix is free os.

learn operating system.

linuxunix which one you choose.

Another example is switching the first three characters in a line

>sed 's/^\(.\)\(.\)\(.\)/\3\2\1/' file.txt

inux is great os. unix is opensource. unix is free os.

aelrn operating system.

inuxlinux which one you choose.

**8.** Duplicating the replaced line with /p flag  
  
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

linux is great os. unix is opensource. unix is free os.

linux is great os. unix is opensource. unix is free os.

learn operating system.

linuxlinux which one you choose.

linuxlinux which one you choose.

**9.** 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

linux is great os. unix is opensource. unix is free os.

linuxlinux which one you choose.

If you use -n alone without /p, then the sed does not print anything.  
  
**10.** 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/'

linux is great system. unix is opensource. unix is free os.

learn operating system.

linuxlinux which one you chosysteme.

Sed provides -e option to run multiple sed commands in a single sed command. The above output can be achieved in a single sed command as shown below.

>sed -e 's/unix/linux/' -e 's/os/system/' file.txt

linux is great system. unix is opensource. unix is free os.

learn operating system.

linuxlinux which one you chosysteme.

**11.** 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

unix is great os. unix is opensource. unix is free os.

learn operating system.

linuxlinux which one you choose.

The above sed command replaces the string only on the third line.  
  
**12.** Replacing string on a range of lines.  
  
You can specify a range of line numbers to the sed command for replacing a string.

>sed '1,3 s/unix/linux/' file.txt

linux is great os. unix is opensource. unix is free os.

learn operating system.

linuxlinux which one you choose.

Here the sed command replaces the lines with range from 1 to 3. Another example is

>sed '2,$ s/unix/linux/' file.txt

linux is great os. unix is opensource. unix is free os.

learn operating system.

linuxlinux which one you choose.

Here $ indicates the last line in the file. So the sed command replaces the text from second line to last line in the file.  
  
**13.** Replace on a lines which matches a pattern.  
  
You can specify a pattern to the sed command to match in a line. If the pattern match occurs, then only the sed command looks for the string to be replaced and if it finds, then the sed command replaces the string.

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

unix is great os. unix is opensource. unix is free os.

learn operating system.

centoslinux which one you choose.

Here the sed command first looks for the lines which has the pattern "linux" and then replaces the word "unix" with "centos".  
  
**14.** Deleting lines.  
  
You can delete the lines a file by specifying the line number or a range or numbers.

>sed '2 d' file.txt

>sed '5,$ d' file.txt

**15.** Duplicating lines  
  
You can make the sed command to print each line of a file two times.

>sed 'p' file.txt

**16.** Sed as grep command  
  
You can make sed command to work as similar to grep command.

>grep 'unix' file.txt

>sed -n '/unix/ p' file.txt

Here the sed command looks for the pattern "unix" in each line of a file and prints those lines that has the pattern.  
  
You can also make the sed command to work as grep -v, just by using the reversing the sed with NOT (!).

>grep -v 'unix' file.txt

>sed -n '/unix/ !p' file.txt

The ! here inverts the pattern match.  
  
**17.** Add a line after a match.  
  
The sed command can add a new line after a pattern match is found. The "a" command to sed tells it to add a new line after a match is found.

>sed '/unix/ a "Add a new line"' file.txt

unix is great os. unix is opensource. unix is free os.

"Add a new line"

learn operating system.

unixlinux which one you choose.

"Add a new line"

**18.** Add a line before a match  
  
The sed command can add a new line before a pattern match is found. The "i" command to sed tells it to add a new line before a match is found.

>sed '/unix/ i "Add a new line"' file.txt

"Add a new line"

unix is great os. unix is opensource. unix is free os.

learn operating system.

"Add a new line"

unixlinux which one you choose.

**19.** Change a line  
  
The sed command can be used to replace an entire line with a new line. The "c" command to sed tells it to change the line.

>sed '/unix/ c "Change line"' file.txt

"Change line"

learn operating system.

"Change line"

**20.** Transform like tr command  
  
The sed command can be used to convert the lower case letters to upper case letters by using the transform "y" option.

>sed 'y/ul/UL/' file.txt

Unix is great os. Unix is opensoUrce. Unix is free os.

Learn operating system.

UnixLinUx which one yoU choose.