**cat**

**cat** is short for concatenate and is one of the most frequently used Linux command line utilities. It is often used to read and print files as well as for simply viewing file contents. To view a file, use the following command:

$ cat <filename>

For example, cat readme.txt will display the contents of readme.txt on the terminal. Often the main purpose of **cat,** however, is to combine (concatenate) multiple files together. You can perform the actions listed in the following table using **cat**:

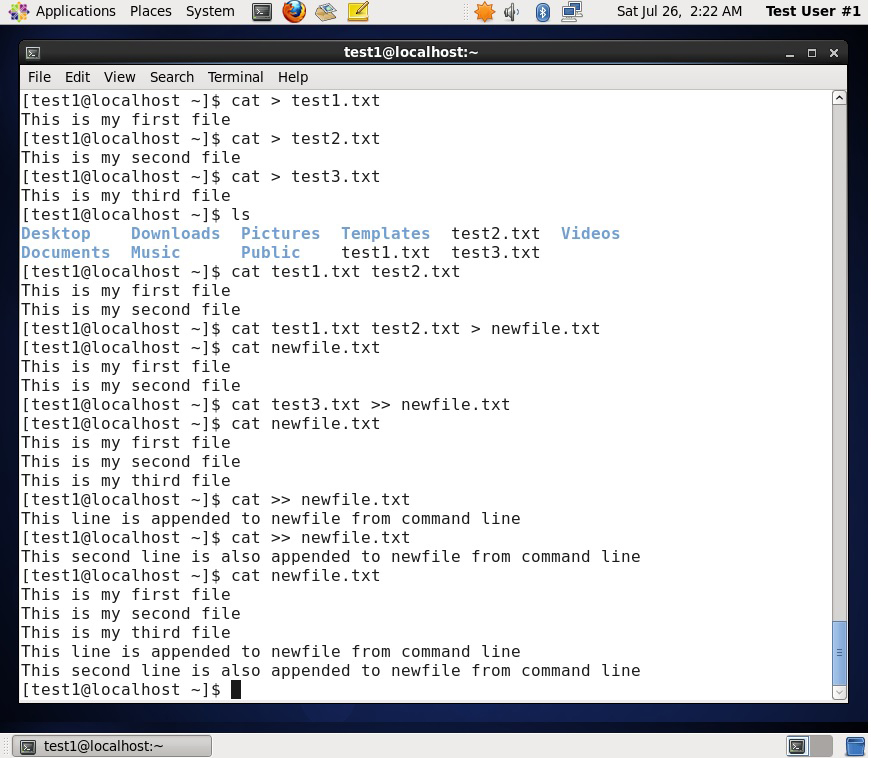
|  |  |
| --- | --- |
| **Command** | **Usage** |
| cat file1 file2 | Concatenate multiple files and display the output; i.e., the entire content of the first file is followed by that of the second file. |
| cat file1 file2 > newfile | Combine multiple files and save the output into a new file. |
| cat file >> existingfile | Append a file to the end of an existing file. |
| cat > file | Any subsequent lines typed will go into the file until CTRL-D is typed. |
| cat >> file | Any subsequent lines are appended to the file until CTRL-D is typed. |

The **tac** command (**cat** spelled backwards) prints the lines of a file in reverse order. (Each line remains the same but the order of lines is inverted.) The syntax of **tac** is exactly the same as for **cat** as in:

$ tac file

$ tac file1 file2 > newfile

**Using cat Interactively**



**cat** can be used to read from standard input (such as the terminal window) if no files are specified. You can use the > operator to create and add lines into a new file, and the >> operator to append lines (or files) to an existing file.

To create a new file, at the command prompt type cat > <filename> and press the **Enter** key.

This command creates a new file and waits for the user to edit/enter the text. After you finish typing the required text, press **CTRL-D** at the beginning of the next line to save and exit the editing.

Another way to create a file at the terminal is cat > <filename> << EOF. A new file is created and you can type the required input. To exit, enter EOF at the beginning of a line.

Note that EOF is case sensitive. (One can also use another word, such as STOP.)

**Note: The next few screens cover the demonstration and Try-It-Yourself activity. You can view a demonstration and practice the procedure through the Try-It-Yourself activity.**

Click the image to view an enlarged version.

**echo**

**echo** simply displays (echoes) text. It is used simply as in:

$ echo string

**echo** can be used to display a string on **standard output** (i.e., the terminal) or to place in a new file (using the > operator) or append to an already existing file (using the >> operator).

The –e option along with the following switches is used to enable special character sequences, such as the **newline** character or horizontal **tab**.

* \n represents newline
* \t represents horizontal tab

**echo** is particularly useful for viewing the values of environment variables (built-in shell variables). For example, echo $USERNAME will print the name of the user who has logged into the current terminal.

The following table lists **echo** commands and their usage:

|  |  |
| --- | --- |
| **Command** | **Usage** |
| echo string > newfile | The specified string is placed in a new file. |
| echo string >> existingfile | The specified string is appended to the end of an already existing file. |
| echo $variable | The contents of the specified environment variable are displayed. |

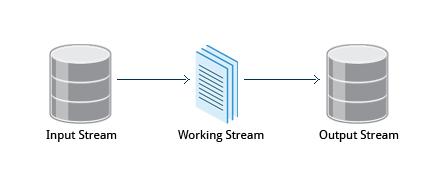
**Introduction to sed and awk**



It is very common to create and then repeatedly edit and/or extract contents from a file. Let’s learn how to use **sed** and **awk** to easily perform such operations.

Note that many Linux users and administrators will write scripts using more comprehensive language utilities such as **python** and **perl**, rather than use **sed** and **awk** (and some other utilities we'll discuss later.) Using such utilities is certainly fine in most circumstances; one should always feel free to use the tools one is experienced with. However, the utilities that are described here are much lighter; i.e., they use fewer system resources, and execute faster. There are times (such as during booting the system) where a lot of time would be wasted using the more complicated tools, and the system may not even be able to run them. So the simpler tools will always be needed.

**sed**



**sed** is a powerful text processing tool and is one of the oldest earliest and most popular UNIX utilities. It is used to modify the contents of a file, usually placing the contents into a new file. Its name is an abbreviation for **stream editor**.

**sed** can filter text as well as perform substitutions in data streams, working like a churn-mill.

Data from an input source/file (or stream) is taken and moved to a working space. The entire list of operations/modifications is applied over the data in the working space and the final contents are moved to the standard output space (or stream).

* [Previous](https://courses.edx.org/courses/LinuxFoundationX/LFS101x/2T2014/courseware/d5b9551efa684412abe71a1976a0f164/cc86413d4c514fc8aa21c3e352ca78d0/1#)
* [Next](https://courses.edx.org/courses/LinuxFoundationX/LFS101x/2T2014/courseware/d5b9551efa684412abe71a1976a0f164/cc86413d4c514fc8aa21c3e352ca78d0/1#)

**sed Command Syntax**

You can invoke **sed** using commands like those listed in the following table:

|  |  |
| --- | --- |
| **Command** | **Usage** |
| sed -e command <filename> | Specify editing commands at the command line, operate on file and put the output on standard out (e.g., the terminal) |
| sed -f scriptfile <filename> | Specify a scriptfile containing sed commands, operate on file and put output on standard out. |

The -e command option allows you to specify multiple editing commands simultaneously at the command lin

**sed Basic Operations**

Now that you know that you can perform multiple editing and filtering operations with **sed**, let’s explain some of them in more detail. The table explains some basic operations, where pattern is the current string and replace\_string is the new string:

|  |  |
| --- | --- |
| **Command** | **Usage** |
| sed s/pattern/replace\_string/ file | Substitute first string occurrence in a line |
| sed s/pattern/replace\_string/g file | Substitute all string occurrences in a line |
| sed 1,3s/pattern/replace\_string/g file | Substitute all string occurrences in a range of lines |
| sed -i s/pattern/replace\_string/g file | Save changes for string substitution in the same file |

You must use the -i option with care, because the action is not reversible. It is always safer to use **sed** without the –i option and then replace the file yourself, as shown in the following example:

$ sed s/pattern/replace\_string/g file > file2

The above command will replace all occurrences of pattern with replace\_string in file1 and move the contents to file2. The contents of file2 can be viewed with cat file2. If you approve you can then overwrite the original file with mv file2 file1.

Ex

ample: To convert 01/02/… to JAN/FEB/…

sed -e 's/01/JAN/' -e 's/02/FEB/' -e 's/03/MAR/' -e 's/04/APR/' -e 's/05/MAY/' \

-e 's/06/JUN/' -e 's/07/JUL/' -e 's/08/AUG/' -e 's/09/SEP/' -e 's/10/OCT/' \

-e 's/11/NOV/' -e 's/12/DEC/'

**awk**

**awk** is used to extract and then print specific contents of a file and is often used to construct reports. It was created at Bell Labs in the 1970s and derived its name from the last names of its authors: Alfred **A**ho, Peter **W**einberger, and Brian **K**ernighan.

**awk** has the following features:

* It is a powerful utility and interpreted programming language.
* It is used to manipulate data files, retrieving, and processing text.
* It works well with **fields** (containing a single piece of data, essentially a column) and **records** (a collection of fields, essentially a line in a file).

**awk** is invoked as shown in the following:

|  |  |
| --- | --- |
| **Command** | **Usage** |
| awk ‘command’ var=value file | Specify a command directly at the command line |
| awk -f scriptfile var=value file | Specify a file that contains the script to be executed along with f |

As with **sed**, short **awk** commands can be specified directly at the command line, but a more complex script can be saved in a file that you can specify using the -f option.

**awk Basic Operations**

The table explains the basic tasks that can be performed using **awk**.The input file is read one line at a time, and for each line, **awk** matches the given pattern in the given order and performs the requested action. The -F option allows you to specify a particular **field separator** character. For example, the /etc/passwd file uses : to separate the fields, so the -F: option is used with the /etc/passwd file.

The command/action in **awk** needs to be surrounded with apostrophes (or single-quote (')). awk can be used as follows:

|  |  |
| --- | --- |
| **Command** | **Usage** |
| awk '{ print $0 }' /etc/passwd | Print entire file |
| awk -F: '{ print $1 }' /etc/passwd | Print first field (column) of every line, separated by a space |
| awk -F: '{ print $1 $6 }' /etc/passwd | Print first and sixth field of every line |

**Note: The next screen covers the Try-It-Yourself activity through which you can practice the procedure.**

* [Previous](https://courses.edx.org/courses/LinuxFoundationX/LFS101x/2T2014/courseware/d5b9551efa684412abe71a1976a0f164/cc86413d4c514fc8aa21c3e352ca78d0/1#)
* [Next](https://courses.edx.org/courses/LinuxFoundationX/LFS101x/2T2014/courseware/d5b9551efa684412abe71a1976a0f164/cc86413d4c514fc8aa21c3e352ca78d0/1#)

**File Manipulation Utilities**

In managing your files you may need to perform many tasks, such as sorting data and

copying data from one location to another. Linux provides several file manipulation utilities that you can use while working with text files. In this section, you will learn about the following file manipulation programs:

* sort
* uniq
* paste
* join
* split

You will also learn about **regular expressions** and **search patterns**.

**sort**

**sort** is used to rearrange the lines of a text file either in ascending or descending order, according to a sort key. You can also sort by particular fields of a file. The default sort key is the order of the ASCII characters (i.e., essentially alphabetically).

**sort** can be used as follows:

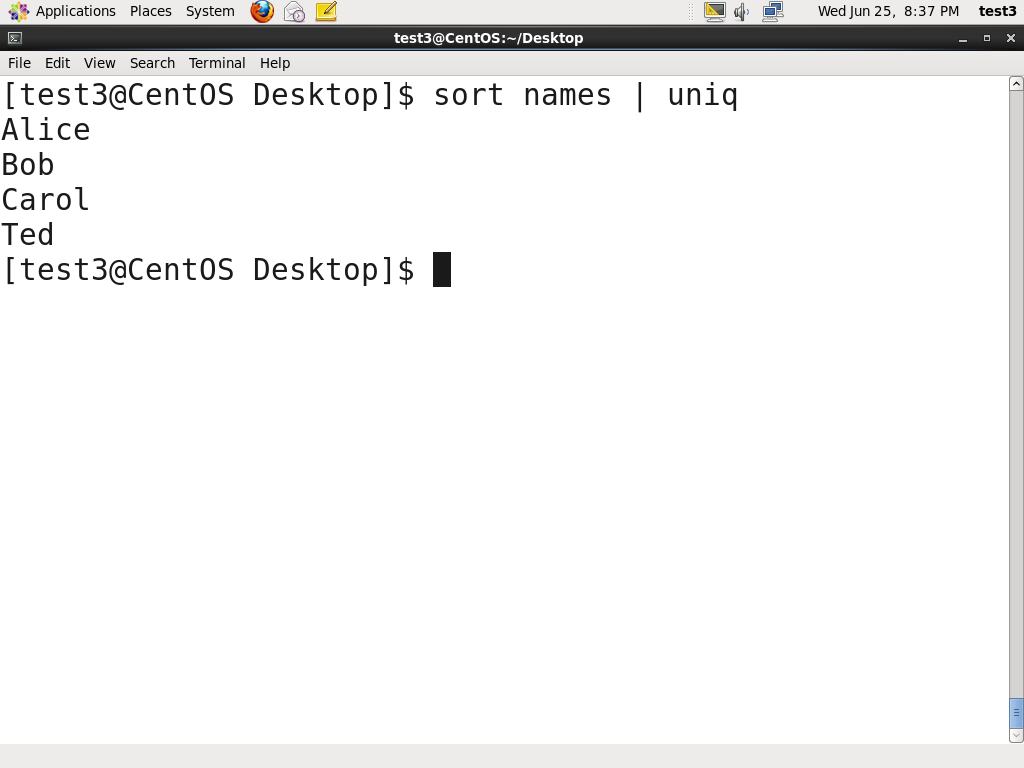
|  |  |
| --- | --- |
| **Syntax** | **Usage** |
| sort <filename> | Sort the lines in the specified file |
| cat file1 file2 | sort | Append the two files, then sort the lines and display the output on the terminal |
| sort -r <filename> | Sort the lines in reverse order |

When used with the -u option, **sort** checks for unique values after sorting the records (lines). It is equivalent to running **uniq** (which we shall discuss) on the output of **sort**.



Click the image to view an enlarged version.

**uniq**

**uniq** is used to remove duplicate lines in a text file and is useful for simplifying text display. **uniq** requires that the duplicate entries to be removed are consecutive. Therefore one often runs **sort** first and then pipes the output into **uniq**;if **sort** is passed the -u option it can do all this in one step. In the example shown, the file is called names and was originally Ted, Bob, Alice, Bob, Carol, Alice.

To remove duplicate entries from some files, use the following command: sort file1 file2 | uniq > file3

**OR**

sort -u file1 file2 > file3

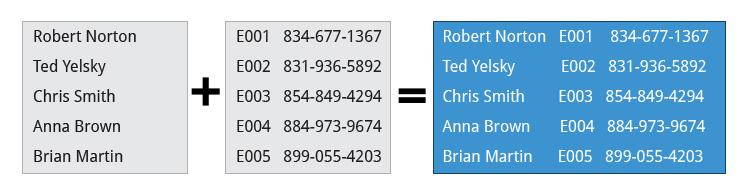
To count the number of duplicate entries, use the following command: uniq -c filename

**Note: The next screen covers the Try-It-Yourself activity through which you can practice the procedure.**

Click the image to view an enlarged version.

* [Previous](https://courses.edx.org/courses/LinuxFoundationX/LFS101x/2T2014/courseware/d5b9551efa684412abe71a1976a0f164/a7992d31d58943649d71f8d61e78c083/1#)
* [Next](https://courses.edx.org/courses/LinuxFoundationX/LFS101x/2T2014/courseware/d5b9551efa684412abe71a1976a0f164/a7992d31d58943649d71f8d61e78c083/1#)

**paste**



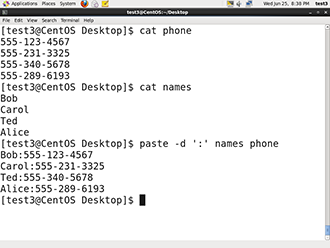
Suppose you have a file that contains the full name of all employees and another file that lists their phone numbers and Employee IDs. You want to create a new file that contains all the data listed in three columns: name, employee ID, and phone number. How can you do this effectively without investing too much time?

**paste** can be used to create a single file containing all three columns. The different columns are identified based on delimiters (spacing used to separate two fields). For example, delimiters can be a blank space, a tab, or an **Enter**. In the image provided, a single space is used as the delimiter in all files.

**paste** accepts the following options:

* -d delimiters, which specify a list of delimiters to be used instead of tabs for separating consecutive values on a single line. Each delimiter is used in turn; when the list has been exhausted, paste begins again at the first delimiter.
* -s, which causes **paste** to append the data in series rather than in parallel; that is, in a horizontal rather than vertical fashion.

**Using paste**



**paste** can be used to combine fields (such as name or phone number) from different files as well as combine lines from multiple files. For example, line one from file1 can be combined with line one of file2, line two from file1 can be combined with line two of file2, and so on.

To paste contents from two files one can do:

$ paste file1 file2

The syntax to use a different delimiter is as follows:

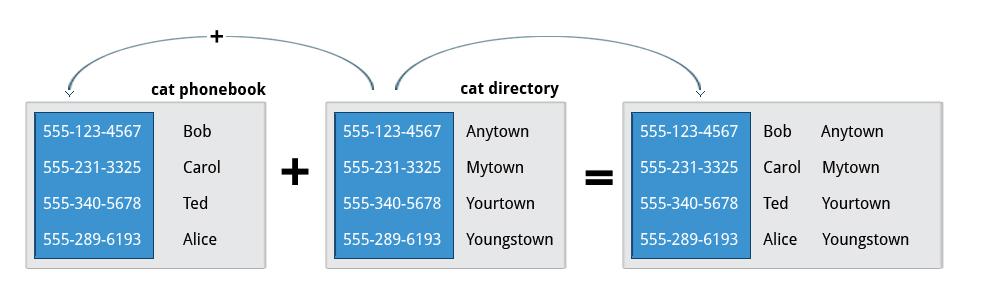
$ paste -d, file1 file2

Common delimiters are 'space', 'tab', '|', 'comma', etc.

Click the image to view an enlarged version.

* [Previous](https://courses.edx.org/courses/LinuxFoundationX/LFS101x/2T2014/courseware/d5b9551efa684412abe71a1976a0f164/a7992d31d58943649d71f8d61e78c083/1#)

**join**



Suppose you have two files with some similar columns. You have saved employees’ phone numbers in two files, one with their first name and the other with their last name. You want to combine the files without repeating the data of common columns. How do you achieve this?

The above task can be achieved using **join**, which is essentially an enhanced version of **paste**. It first checks whether the files share common fields, such as names or phone numbers, and then joins the lines in two files based on a common field.

* [Previous](https://courses.edx.org/courses/LinuxFoundationX/LFS101x/2T2014/courseware/d5b9551efa684412abe71a1976a0f164/a7992d31d58943649d71f8d61e78c083/1#)

**Using join**



To combine two files on a common field, at the command prompt type join file1 file2 and press the **Enter** key.

For example, the common field (i.e., it contains the same values) among the phonebook and directory files is the phone number, as shown by the output of the following **cat** commands:

$ cat phonebook

555-123-4567 Bob

555-231-3325 Carol

555-340-5678 Ted

555-289-6193 Alice

$ cat directory

555-123-4567 Anytown

555-231-3325 Mytown

555-340-5678 Yourtown

555-289-6193 Youngstown

The result of **join**ing these two file is as shown in the output of the following command:

$ join phonebook directory

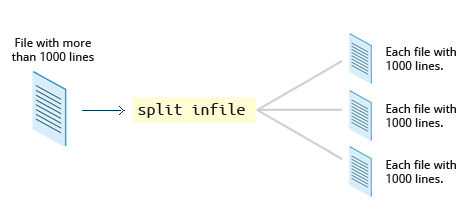
555-123-4567 Bob Anytown

555-231-3325 Carol Mytown

555-340-5678 Ted Yourtown

555-289-6193 Alice Youngstown

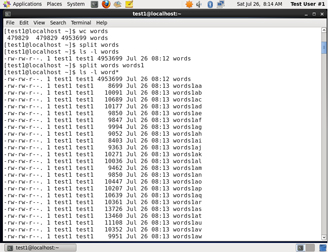
**split**



**split** is used to break up (or split) a file into equal-sized segments for easier viewing and manipulation, and is generally used only on relatively large files. By default **split** breaks up a file into 1,000-line segments. The original file remains unchanged, and set of new files with the same name plus an added prefix is created. By default, the **x** prefix is added. To split a file into segments, use the command split infile.

To split a file into segments using a different prefix, use the command split infile <Prefix>.

**Using split**

To demonstrate the use of **split,** we'll apply it to an american-english dictionary file of over 99,000 lines:

$ wc -l american-english

99171 american-english

where we have used the **wc** program (soon to be discussed) to report on the number of lines in the file. Then typing:

$ split american-english dictionary

will split the american-english file into equal-sized segments named 'dictionary'.

$ ls -l dictionary\*

-rw-rw-r 1 me me 8552 Mar 23 20:19 dictionaryab

-rw-rw-r 1 me me 8653 Mar 23 20:19 dictionaryaa

. . .

**Regular Expressions and Search Patterns**

**Regular expressions** are text strings used for matching a specific **pattern**, or to search for a specific location, such as the start or end of a line or a word. Regular expressions can contain both normal characters or so-called **metacharacters**, such as \* and $.

Many text editors and utilities such as **vi**, **sed**, **awk**, **find** and **grep** work extensively with regular expressions. Some of the popular computer languages that use regular expressions include **Perl**, **Python** and **Ruby**. It can get rather complicated and there are whole books written about regular expressions; we'll only skim the surface here.

These regular expressions are different from the wildcards (or "metacharacters") used in filename matching in command shells such as **bash** (which were covered in the earlier Chapter on Command Line Operations)**.** The table lists search patterns and their usage.

|  |  |
| --- | --- |
| **Search Patterns** | **Usage** |
| .(dot) | Match any single character |
| a|z | Match a or z |
| $ | Match end of string |
| \* | Match preceding item 0 or more times |

* [Previous](https://courses.edx.org/courses/LinuxFoundationX/LFS101x/2T2014/courseware/d5b9551efa684412abe71a1976a0f164/a7992d31d58943649d71f8d61e78c083/1#)
* [Next](https://courses.edx.org/courses/LinuxFoundationX/LFS101x/2T2014/courseware/d5b9551efa684412abe71a1976a0f164/a7992d31d58943649d71f8d61e78c083/1#)

**Using Regular Expressions and Search Patterns**

For example, Consider the following sentence:

**the quick brown fox jumped over the lazy dog**

Some of the patterns that can be applied to this sentence are as follows:

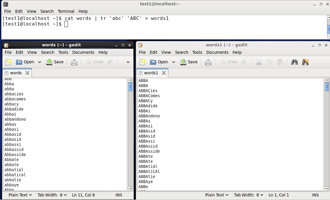
|  |  |
| --- | --- |
| **Command** | **Usage** |
| a.. | matches azy |
| b.|j. | matches both br and ju |
| ..$ | matches og |
| l.\* | matches lazy dog |
| l.\*y | matches lazy |
| the.\* | matches the whole sentence |

**grep**

**grep** is extensively used as a primary text searching tool. It scans files for specified patterns and can be used with regular expressions as well as simple strings as shown in the table.

|  |  |
| --- | --- |
| **Command** | **Usage** |
| grep [pattern] <filename> | Search for a pattern in a file and print all matching lines |
| grep -v [pattern] <filename> | Print all lines that do **not** match the pattern |
| grep [0-9] <filename> | Print the lines that contain the numbers 0 through 9 |
| grep -C 3 [pattern] <filename> | Print context of lines (specified number of lines above and below the pattern) for matching the pattern. Here the number of lines is specified as 3. |

**tr**

In this section, you will learn about some additional text utilities that you can use for performing various actions on your Linux files, such as changing the case of letters or determining the count of words, lines, and characters in a file.

The **tr** utilityis used to **translate** specified characters into other characters or to delete them. The general syntax is as follows:

$ tr [options] set1 [set2]

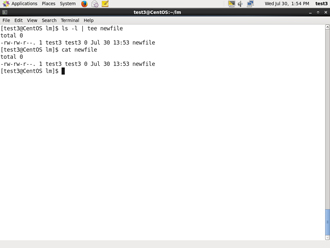
The items in the square brackets are optional. **tr** requires at least one argument and accepts a maximum of two. The first, designated set1 in the example, lists the characters in the text to be replaced or removed. The second, set2, lists the characters that are to be substituted for the characters listed in the first argument. Sometimes these sets need to be surrounded by apostrophes (or single-quotes (')) in order to have the shell ignore that they mean something special to the shell. It is usually safe (and may be required) to use the single-quotes around each of the sets as you will see in the examples below.

For example, suppose you have a file named city containing several lines of text in mixed case. To translate all lower case characters to upper case, at the command prompt type cat city | tr a-z A-Z and press the **Enter** key.

|  |  |
| --- | --- |
| **Command** | **Usage** |
| $ tr abcdefghijklmnopqrstuvwxyz ABCDEFGHIJKLMNOPQRSTUVWXYZ | Convert lower case to upper case |
| $ tr '{}' '()' < inputfile > outputfile | Translate braces into parenthesis |
| $ echo "This is for testing" | tr [:space:] '\t' | Translate white-space to tabs |
| $ echo "This is for testing" | tr -s [:space:] | Squeeze repetition of characters using -s |
| $ echo "the geek stuff" | tr -d 't' | Delete specified characters using -d option |
| $ echo "my username is 432234" | tr -cd [:digit:] | Complement the sets using -c option |
| $ tr -cd [:print:] < file.txt | Remove all non-printable character from a file |
| $ tr -s '\n' ' ' < file.txt | Join all the lines in a file into a single line |

Click the image to view an enlarged version.

**tee**



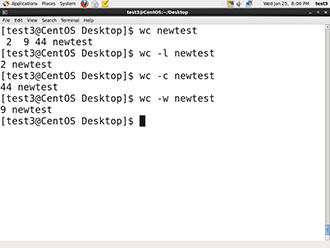
**tee** takes the output from any command, and while sending it to standard output, it also saves it to a file. In other words, it "tees**"** the output stream from the command: one stream is displayed on the standard output and the other is saved to a file.

For example, to list the contents of a directory on the screen and save the output to a file, at the command prompt type ls -l | tee newfile and press the **Enter** key.

Typing cat newfile will then display the output of ls –l.

Click the image to view an enlarged version.

**wc**

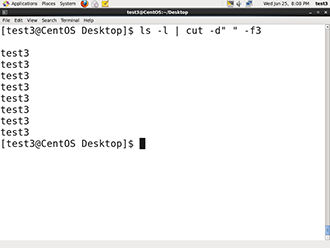
**wc** (word count) counts the number of lines, words, and characters in a file or list of files. Options are given in the table below.

By default all three of these options are active.

For example, to print the number of lines contained in a file, at the command prompt type wc -l filename and press the **Enter** key.

|  |  |
| --- | --- |
| **Option** | **Description** |
| –l | display the number of lines. |
| -c | display the number of bytes. |
| -w | display the number of words. |

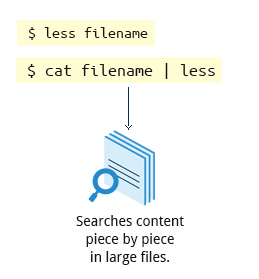
**cut**



**cut** is used for manipulating column-based files and is designed to extract specific columns. The default column separator is the **tab** character. A different delimiter can be given as a command option.

For example, to display the third column delimited by a blank space, at the command prompt type ls -l | cut -d" " -f3 and press the **Enter** key.

**Working with Large Files**



System administrators need to work with configuration files, text files, documentation files, and log files. Some of these files may be large or become quite large as they accumulate data with time. These files will require both viewing and administrative updating. In this section, you will learn how to manage such large files.

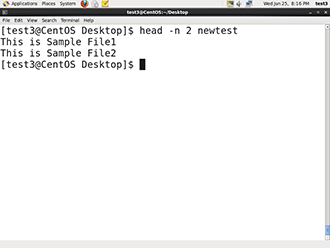
For example, a banking system might maintain one simple large log file to record details of all of one day's ATM transactions. Due to a security attack or a malfunction, the administrator might be forced to check for some data by navigating within the file. In such cases, directly opening the file in an editor will cause issues, due to high memory utilization, as an editor will usually try to read the whole file into memory first. However, one can use **less** to view the contents of such a large file, scrolling up and down page by page without the system having to place the entire file in memory before starting. This is much faster than using a text editor.

Viewing the file can be done by typing either of the two following commands:

$ less <filename>

$ cat <filename> | less

**head**

**head** reads the first few lines of each named file (10 by default) and displays it on standard output. You can give a different number of lines in an option.

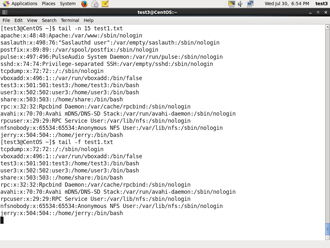
For example, If you want to print the first 5 lines from atmtrans.txt, use the following command:

$ head –n 5 atmtrans.txt

(You can also just say head -5 atmtrans.txt.)

Click the image to view an enlarged version.

**tail**

**tail** prints the last few lines of each named file and displays it on standard output. By default, it displays the last 10 lines. You can give a different number of lines as an option. **tail** is especially useful when you are troubleshooting any issue using log files as you probably want to see the most recent lines of output.

For example, to display the last 15 lines of atmtrans.txt, use the following command:

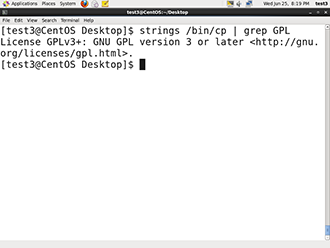
$ tail -n 15 atmtrans.txt

(You can also just say tail -15 atmtrans.txt.) To continually monitor new output in a growing log file:

$ tail -f atmtrans.txt

This command will continuously display any new lines of output in **atmtrans.txt** as soon as they appear. Thus it enables you to monitor any current activity that is being reported and recorded.

**strings**



**strings** is used to extract all printable character strings found in the file or files given as arguments. It is useful in locating human readable content embedded in binary files: for text files one can just use **grep**.

For example, to search for the string **my\_string** in a spreadsheet:

$ strings book1.xls | grep my\_string

**The z Command Family**

When working with compressed files many standard commands can not be used directly. For many commonly-used file and text manipulation programs there is also a version especially designed to work directly with compressed files. These associated utilities have the letter **z** prefixed to their name. For example, we have utility programs such as **zcat**, **zless**, **zdiff,** and **zgrep**.

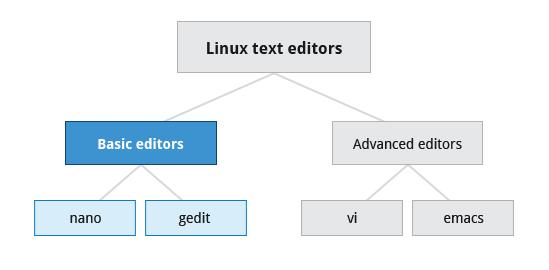
Here is a table listing some z family commands:

|  |  |
| --- | --- |
| **Command** | **Description** |
| $ zcat compressed-file.txt.gz | To view a compressed file |
| $ zless <filename>.gz  or  $ zmore <filename>.gz | To page through a compressed file |
| $ zgrep -i less test-file.txt.gz | To search inside a compressed file |
| $ zdiff filename1.txt.gz  filename2.txt.gz | To compare two compressed files |

Note that if you run **zless** on an uncompressed file, it will still work and ignore the decompression stage. There are also equivalent utility programs for other compression methods besides **gzip**; i.e, we have **bzcat** and **bzless** associated with **bzip2,** and **xzcat** and **xzless** associated with **xz**.

**Note: The next two screens cover the Try-It-Yourself activities through which you can practice the procedures.**

**Overview of Text Editors in Linux**



At some point you will need to manually edit **text files**. You might be composing an email off-line, writing a script to be used for **bash** or other command interpreters, altering a system or application configuration file, or developing source code for a programming language such as **C** or **Java**.

Linux Administrators quite often sidestep the text editors, by using graphical utilities for creating and modifying system configuration files. However, this can be far more laborious than directly using a text editor. Note that word processing applications such as **Notepad** or the applications that are part of office suites are not really basic text editors because they add a lot of extra (usually invisible) formatting information that will probably render system administration configuration files unusable for their intended purpose. So using text editors really is essential in Linux.

By now you have certainly realized Linux is packed with choices; when it comes to text editors, there are many choices ranging from quite simple to very complex, including:

**- nano**

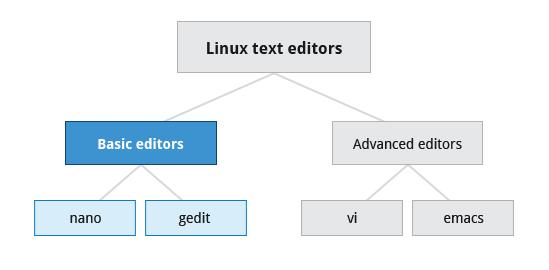
**- gedit**

**- vi**

**- emacs**

In this section, we will learn about **nano** and **gedit**; editors which are relatively simple and easy to learn. Before we start, let's take a look at some cases where an editor is not needed.

**Overview of Text Editors in Linux**



At some point you will need to manually edit **text files**. You might be composing an email off-line, writing a script to be used for **bash** or other command interpreters, altering a system or application configuration file, or developing source code for a programming language such as **C** or **Java**.

Linux Administrators quite often sidestep the text editors, by using graphical utilities for creating and modifying system configuration files. However, this can be far more laborious than directly using a text editor. Note that word processing applications such as **Notepad** or the applications that are part of office suites are not really basic text editors because they add a lot of extra (usually invisible) formatting information that will probably render system administration configuration files unusable for their intended purpose. So using text editors really is essential in Linux.

By now you have certainly realized Linux is packed with choices; when it comes to text editors, there are many choices ranging from quite simple to very complex, including:

**- nano**

**- gedit**

**- vi**

**- emacs**

In this section, we will learn about **nano** and **gedit**; editors which are relatively simple and easy to learn. Before we start, let's take a look at some cases where an editor is not needed.

**Creating Files Without Using an Editor**



Sometimes you may want to create a short file and don't want to bother invoking a full text editor. In addition, doing so can be quite useful when used from within scripts, even when creating longer files. You'll no doubt find yourself using this method when you start on the later chapters that cover **bash** scripting!

If you want to create a file without using an editor there are two standard ways to create one from the command line and fill it with content.

The first is to use **echo** repeatedly:

$ echo line one > myfile

$ echo line two >> myfile

$ echo line three >> myfile

Earlier we learned that a single greater-than sign (>) will send the output of a command to a file. Two greater-than signs (>>) will **append** new output to an existing file.

The second way is to use **cat** combined with redirection:

$ cat << EOF > myfile

> line one

> line two

> line three

> EOF

$

Both the above techniques produce a file with the following lines in it:

line one

line two

line three

and are extremely useful when employed by scripts.

**nano and gedit**

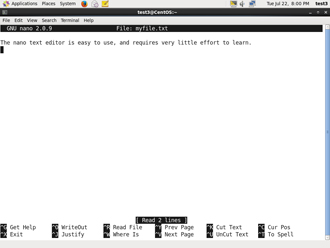


There are some text editors that are pretty obvious; they require no particular experience to learn and are actually quite capable if not robust. One particularly easy one to use is the text-terminal based editor **nano**. Just invoke **nano** by giving a file name as an argument. All the help you need is displayed at the bottom of the screen, and you should be able to proceed without any problem.

As a graphical editor, **gedit** is part of the **GNOME** desktop system (**kwrite** is associated with **KDE)**. The **gedit** and **kwrite** editors are very easy to use and are extremely capable. They are also very configurable. They look a lot like **Notepad** in **Windows**. Other variants such as **kedit** and **kate** are also supported by **KDE**.

* [Previous](https://courses.edx.org/courses/LinuxFoundationX/LFS101x/2T2014/courseware/faa5967a9e09455499a9e57e3d9232a1/2d2e9075cac249e9be5b813d4d9a705a/1#)
* [Next](https://courses.edx.org/courses/LinuxFoundationX/LFS101x/2T2014/courseware/faa5967a9e09455499a9e57e3d9232a1/2d2e9075cac249e9be5b813d4d9a705a/1#)

**nano**



**nano** is easy to use, and requires very little effort to learn. To open a file in **nano**, type nano <filename> and press **Enter**. If the file doesn't exist, it will be created.

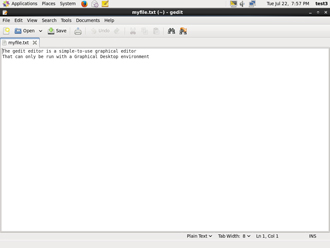
**nano** provides a two line “shortcut bar” at the bottom of the screen that lists the available commands. Some of these commands are:

* CTRL-G: Display the help screen
* CTRL-O: Write to a file
* CTRL-X: Exit a file
* CTRL-R: Insert contents from another file to the current buffer
* CTRL-C: Cancels previous commands

Click the image to view an enlarged version.

* [Previous](https://courses.edx.org/courses/LinuxFoundationX/LFS101x/2T2014/courseware/faa5967a9e09455499a9e57e3d9232a1/2d2e9075cac249e9be5b813d4d9a705a/1#)
* [Next](https://courses.edx.org/courses/LinuxFoundationX/LFS101x/2T2014/courseware/faa5967a9e09455499a9e57e3d9232a1/2d2e9075cac249e9be5b813d4d9a705a/1#)

**gedit**

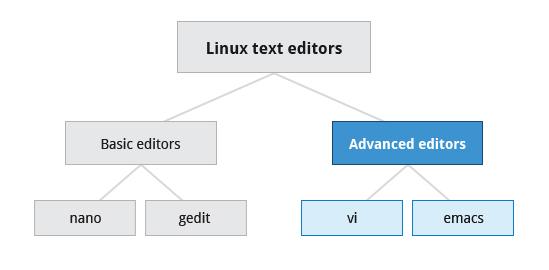


**gedit** (pronounced 'g-edit') is a simple-to-use graphical editor that can only be run within a Graphical Desktop environment. It is visually quite similar to the **Notepad** text editor in **Windows**, but is actually far more capable and very configurable and has a wealth of plugins available to extend its capabilities further.

To open a new file in **gedit**, find the program in your desktop's menu system, or from the command line type gedit <filename>. If the file doesn't exist it will be created.

Using **gedit** is pretty straight-forward and doesn't require much training. Its interface is composed of quite familiar elements.

**vi and emacs**



Developers and administrators experienced in working on UNIX-like systems almost always use one of the two venerable editing options; **vi** and **emacs**. Both are present or easily available on all distributions and are completely compatible with the versions available on other operating systems.

Both **vi** and **emacs** have a basic purely text-based form that can run in a non-graphical environment. They also have one or more **X**-based graphical forms with extended capabilities; these may be friendlier for a less experienced user. While **vi** and **emacs** can have significantly steep learning curves for new users, they are extremely efficient when one has learned how to use them.

You need to be aware that fights among seasoned users over which editor is better can be quite intense and are often described as a holy war.

* [Previous](https://courses.edx.org/courses/LinuxFoundationX/LFS101x/2T2014/courseware/faa5967a9e09455499a9e57e3d9232a1/ea8bf65ed93c433ea3b85beb12f775a6/1#)
* [Next](https://courses.edx.org/courses/LinuxFoundationX/LFS101x/2T2014/courseware/faa5967a9e09455499a9e57e3d9232a1/ea8bf65ed93c433ea3b85beb12f775a6/1#)

**Introduction to vi**



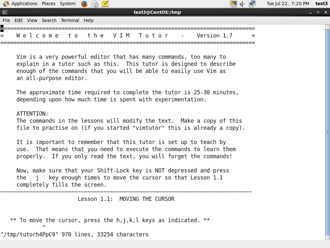
Usually the actual program installed on your system is **vim** which stands for **vi Improved**, and is aliased to the name **vi**. The name is pronounced as “vee-eye”.

Even if you don’t want to use **vi,** it is good to gain some familiarity with it: it is a standard tool installed on virtually all Linux distributions. Indeed, there may be times where there is no other editor available on the system.

**GNOME** extends **vi** with a very graphical interface known as **gvim** and **KDE** offers **kvim**. Either of these may be easier to use at first.

When using **vi**, all commands are entered through the keyboard; you don’t need to keep moving your hands to use a pointer device such as a mouse or touchpad, unless you want to do so when using one of the graphical versions of the editor.

**vimtutor**



Typing vimtutor launches a short but very comprehensive tutorial for those who want to learn their first **vi** commands. This tutorial is a good place to start learning **vi**. Even though it provides only an introduction and just seven lessons, it has enough material to make you a very proficient **vi** user because it covers a large number of commands. After learning these basic ones, you can look up new tricks to incorporate into your list of **vi** commands because there are always more optimal ways to do things in **vi** with less typing.

Click the image to view an enlarged version.

**Modes in vi**

**vi** provides three **modes** as described in the table below. It is vital to not lose track of which mode you are in. Many keystrokes and commands behave quite differently in different modes.

|  |  |
| --- | --- |
| **Mode** | **Feature** |
| **Command** | * By default, **vi** starts inCommand mode. * Each key is an editor command. * Keyboard strokes are interpreted as commands that can modify file contents. |
| **Insert** | * Type i to switch to Insert mode from Command mode. * Insert mode is used to enter (insert) text into a file. * Insert mode is indicated by an “**? INSERT ?**” indicator at the bottom of the screen. * Press Esc to exit Insert mode and return to Command mode. |
| **Line** | * Type: to switch to the Line mode from Command mode. Each key is an external command, including operations such as writing the file contents to disk or exiting. * Uses line editing commands inherited from older line editors. Most of these commands are actually no longer used. Some line editing commands are very powerful. * Press Esc to exit Line mode and return to Command mode. |

The table describes the most important commands used to start, exit, read, and write files in **vi**. The **ENTER** key needs to be pressed after all of these commands.

|  |  |
| --- | --- |
| **Command** | **Usage** |
| vi myfile | Start the **vi** editor and edit the **myfile** file |
| vi -r myfile | Start **vi** and edit **myfile** in recovery mode from a system crash |
| :r file2 | Read in **file2** and insert at current position |
| :w | Write to the file |
| :w myfile | Write out the file to **myfile** |
| :w! file2 | Overwrite **file2** |
| :x or :wq | Exit **vi** and write out modified file |
| :q | Quit **vi** |
| :q! | Quit **vi** even though modifications have not been saved |

* [Previous](https://courses.edx.org/courses/LinuxFoundationX/LFS101x/2T2014/courseware/faa5967a9e09455499a9e57e3d9232a1/ea8bf65ed93c433ea3b85beb12f775a6/1#)
* [Next](https://courses.edx.org/courses/LinuxFoundationX/LFS101x/2T2014/courseware/faa5967a9e09455499a9e57e3d9232a1/ea8bf65ed93c433ea3b85beb12f775a6/1#)

**Changing Cursor Positions in vi**

The table describes the most important keystrokes used when changing cursor position in **vi.** Line mode commands (those following colon (:)) require the **ENTER** key to be pressed after the command is typed.

|  |  |
| --- | --- |
| **Key** | **Usage** |
| arrow keys | To move up, down, left and right |
| j or <ret> | To move one line down |
| k | To move one line up |
| h or Backspace | To move one character left |
| l or Space | To move one character right |
| 0 | To move to beginning of line |
| $ | To move to end of line |
| w | To move to beginning of next word |
| :0 or 1G | To move to beginning of file |
| :n or nG | To move to line n |
| :$ or G | To move to last line in file |
| CTRL-F or Page Down | To move forward one page |
| CTRL-B or Page Up | To move backward one page |
| ^l | To refresh and center screen |

**Searching for Text in vi**

The table describes the most important commands used when searching for text in **vi**. The **ENTER** key should be pressed after typing the search pattern.

|  |  |
| --- | --- |
| **Command** | **Usage** |
| /pattern | Search forward for pattern |
| ?pattern | Search backward for pattern |

The table describes the most important keystrokes used when searching for text in **vi**.

|  |  |
| --- | --- |
| **Key** | **Usage** |
| n | Move to next occurrence of search pattern |
| N | Move to previous occurrence of search pattern |

* [Previous](https://courses.edx.org/courses/LinuxFoundationX/LFS101x/2T2014/courseware/faa5967a9e09455499a9e57e3d9232a1/ea8bf65ed93c433ea3b85beb12f775a6/1#)
* [Next](https://courses.edx.org/courses/LinuxFoundationX/LFS101x/2T2014/courseware/faa5967a9e09455499a9e57e3d9232a1/ea8bf65ed93c433ea3b85beb12f775a6/1#)

**Working with Text in vi**

The table describes the most important keystrokes used when changing, adding, and deleting text in **vi.**

Click the link to download a consolidated PDF file with commands for **vi.**

[commands for vi](https://learningmate.s3-us-west-2.amazonaws.com/LFS01/Chapter10/labs/VI%20Editor.pdf)

|  |  |
| --- | --- |
| **Key** | **Usage** |
| a | Append text after cursor; stop upon Escape key |
| A | Append text at end of current line; stop upon Escape key |
| i | Insert text before cursor; stop upon Escape key |
| I | Insert text at beginning of current line; stop upon Escape key |
| o | Start a new line below current line, insert text there; stop upon Escape key |
| O | Start a new line above current line, insert text there; stop upon Escape key |
| r | Replace character at current position |
| R | Replace text starting with current position; stop upon Escape key |
| x | Delete character at current position |
| Nx | Delete N characters, starting at current position |
| dw | Delete the word at the current position |
| D | Delete the rest of the current line |
| dd | Delete the current line |
| Ndd or dNd | Delete N lines |
| u | Undo the previous operation |
| yy | Yank (copy) the current line and put it in buffer |
| Nyy or yNy | Yank (copy) N lines and put it in buffer |
| p | Paste at the current position the yanked line or lines from the buffer. |

**Using External Commands**

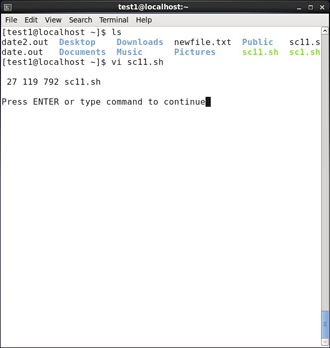
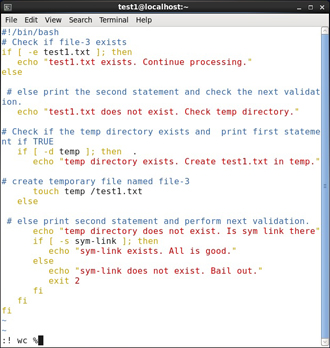
Typing :sh command opens an external command shell. When you exit the shell, you will resume your **vi** editing session.

Typing :!executes a command from within **vi**. The command follows the exclamation point. This technique best suited for non-interactive commands such as:

:! wc %

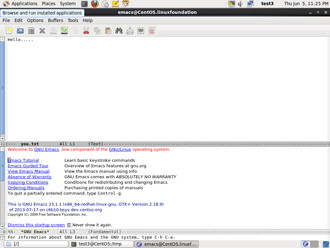
Typing this will run the wc (word count) command on the file; the character % represents the file currently being edited.

The **fmt** command does simple formatting of text. If you are editing a file and want the file to look nice, you can run the file through **fmt**. One way to do this while editing is by using:%!fmt, which runs the entire file (the % part) through **fmt** and replaces the file with the results.



Click the image to view an enlarged version.

**Introduction to emacs**



The **emacs** editor is a popular competitor for **vi**. Unlike **vi**, it does not work with modes. **emacs** is highly customizable and includes a large number of features. It was initially designed for use on a console, but was soon adapted to work with a GUI as well. **emacs** has many other capabilities other than simple text editing; it can be used for email, debugging, etc.

Rather than having different modes for command and insert, like **vi**, **emacs** uses the **CTRL** and **Esc** keys for special commands.

Click the image to view an enlarged version.

**Working with emacs**

The table lists some of the most important key combinations that are used when starting, exiting, reading, and writing files in **emacs**.

|  |  |
| --- | --- |
| **Key** | **Usage** |
| emacs myfile | Start emacs and edit myfile |
| CTRL-x i | Insert prompted for file at current position |
| CTRL-x s | Save all files |
| CTRL-x CTRL-w | Write to the file giving a new name when prompted |
| CTRL-x CTRL-s | Saves the current file |
| CTRL-x CTRL-c | Exit after being prompted to save any modified files |

The **emacs** tutorial is a good place to start learning basic **emacs** commands. It is available any time when in **emacs** by simply typing CTRL-h (for help) and then the letter t for tutorial.

* [Previous](https://courses.edx.org/courses/LinuxFoundationX/LFS101x/2T2014/courseware/faa5967a9e09455499a9e57e3d9232a1/ea8bf65ed93c433ea3b85beb12f775a6/1#)
* [Next](https://courses.edx.org/courses/LinuxFoundationX/LFS101x/2T2014/courseware/faa5967a9e09455499a9e57e3d9232a1/ea8bf65ed93c433ea3b85beb12f775a6/1#)

**Changing Cursor Positions in emacs**

The table lists some of the keys and key combinations that are used for changing cursor positions in emacs.

|  |  |
| --- | --- |
| **Key** | **Usage** |
| arrow keys | Use the arrow keys for up, down, left and right |
| CTRL-n | One line down |
| CTRL-p | One line up |
| CTRL-f | One character forward/right |
| CTRL-b | One character back/left |
| CTRL-a | Move to beginning of line |
| CTRL-e | Move to end of line |
| Esc-f | Move to beginning of next word |
| Esc-b | Move back to beginning of preceding word |
| Esc-< | Move to beginning of file |
| Esc-x | Goto-line n move to line n |
| Esc-> | Move to end of file |
| CTRL-v or Page Down | Move forward one page |
| Esc-v or Page Up | Move backward one page |
| CTRL-l | Refresh and center screen |

* [Previous](https://courses.edx.org/courses/LinuxFoundationX/LFS101x/2T2014/courseware/faa5967a9e09455499a9e57e3d9232a1/ea8bf65ed93c433ea3b85beb12f775a6/1#)
* [Next](https://courses.edx.org/courses/LinuxFoundationX/LFS101x/2T2014/courseware/faa5967a9e09455499a9e57e3d9232a1/ea8bf65ed93c433ea3b85beb12f775a6/1#)

**Searching for Text in emacs**

The table lists the key combinations that are used for searching for text in **emacs**.

|  |  |
| --- | --- |
| **Key** | **Usage** |
| CTRL-s | Search forward for prompted pattern, or for next pattern |
| CTRL-r | Search backwards for prompted pattern, or for next pattern |

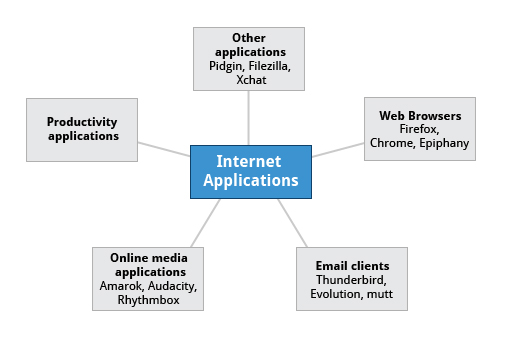
* [Previous](https://courses.edx.org/courses/LinuxFoundationX/LFS101x/2T2014/courseware/faa5967a9e09455499a9e57e3d9232a1/ea8bf65ed93c433ea3b85beb12f775a6/1#)
* [Next](https://courses.edx.org/courses/LinuxFoundationX/LFS101x/2T2014/courseware/faa5967a9e09455499a9e57e3d9232a1/ea8bf65ed93c433ea3b85beb12f775a6/1#)

**Working with Text in emacs**

The table lists some of the key combinations used for changing, adding, and deleting text in **emacs**:

|  |  |
| --- | --- |
| **Key** | **Usage** |
| CTRL-o | Insert a blank line |
| CTRL-d | Delete character at current position |
| CTRL-k | Delete the rest of the current line |
| CTRL-\_ | Undo the previous operation |
| CTRL- (space or CTRL-@) | Mark the beginning of the selected region. The end will be at the cursor position |
| CTRL-w | Delete the current marked text and write it to the buffer |
| CTRL-y | Insert at current cursor location whatever was most recently deleted |

**Internet Applications**



The Internet is a global network that allows users around the world to perform multiple tasks such as searching for data, communicating through emails and online shopping. Obviously, you need to use network-aware applications to take advantage of the Internet. These include:

* Web browsers
* Email clients
* Online media applications
* Other applications