**Chapter 10: Text Editors**

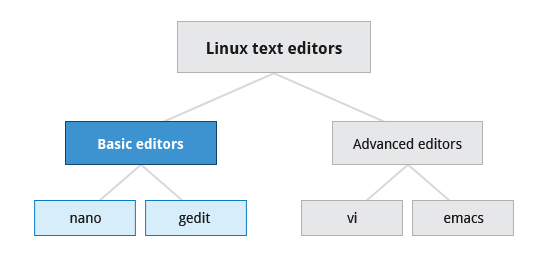
**Learning Objectives**

By the end of this chapter, you should be famililar with:

* How to create and edit files using the available Linux text editors.
* **nano**, a simple text-based editor.
* **gedit**, a simple graphical editor

**vi** and **emacs,**two advanced editors with both text-based and graphical interfaces.

**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**

[](https://courses.edx.org/c4x/LinuxFoundationX/LFS101x/asset/LFS01_ch10_screen05.jpg)

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.

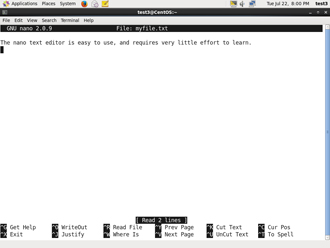
**nano and gedit**



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**.

**nano**

[](https://courses.edx.org/c4x/LinuxFoundationX/LFS101x/asset/LFS01_ch10_screen06.jpg)

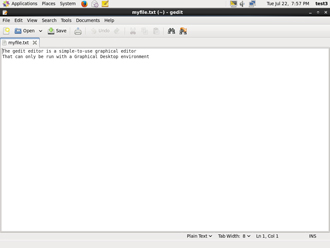
**nano**  is easy to use, and requires very little effort to learn. To open a file in **nano**, type nano <filename> and press **Enter**.  If the file doesn't exist, it will be created.

**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

**gedit**

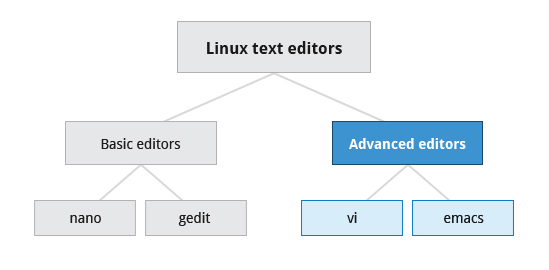
[](https://courses.edx.org/c4x/LinuxFoundationX/LFS101x/asset/LFS01_ch10_screen07.jpg)

**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**

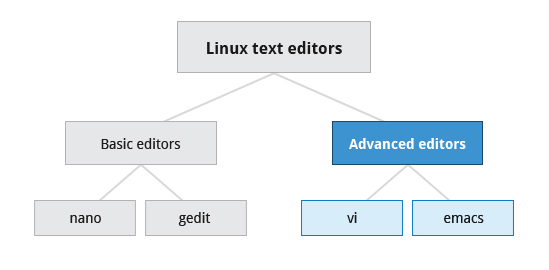


Developers and administrators experienced in working on UNIX-like systems almost always use one of the two venerable editing options; **vi** and **emacs**. Both are present or easily available on all distributions and are completely compatible with the versions available on other operating systems.

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.

**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.

**Introduction to vi**

[](https://courses.edx.org/c4x/LinuxFoundationX/LFS101x/asset/LFS01_ch10_screen09.jpg)

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”.

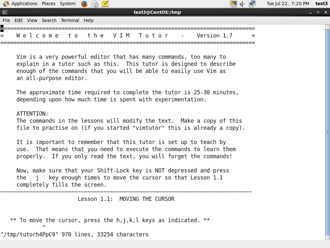
Even if you don’t want to use **vi,** it is good to gain some familiarity with it: it is a standard tool installed on virtually all Linux distributions. Indeed, there may be times where there is no other editor available on the system.

**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.

Click the image to view an enlarged version.

**vimtutor**

[](https://courses.edx.org/c4x/LinuxFoundationX/LFS101x/asset/LFS01_ch10_screen16.jpg)

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.

**Modes in vi**

|  |  |
| --- | --- |
| **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. |

**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.

**Working with Files in vi**

The table describes the most important commands used to start, exit, read, and write files in **vi**.

|  |  |
| --- | --- |
| **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<ret> | Read in **file2** and insert at current position |
| :w<ret> | Write to the file |
| :w myfile<ret> | Write out the file to **myfile** |
| :w! file2<ret> | Overwrite **file2** |
| :x<ret> or :wq<ret> | Exit **vi** and write out modified file |
| :q<ret> | Quit **vi** |
| :q!<ret> | Quit **vi** even though modifications have not been saved |

**Changing Cursor Positions in vi**

The table describes the most important keystrokes used when changing cursor position in **vi.**

|  |  |
| --- | --- |
| **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 |
| O | To move to beginning of line |
| $ | To move to end of line |
| w | To move to beginning of next word |
| :O <ret> or 1G | To move to beginning of file |
| :n <ret> or nG | To move to line n |
| :$ <ret> 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**.

|  |  |
| --- | --- |
| **Command** | **Usage** |
| /pattern<ret> | Search forward for pattern |
| ?pattern<ret> | 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 |

**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 (cut) the current line and put it in buffer |
| Nyy or yNy | Yank (cut) N lines and put it in buffer |
| p | Paste at the current position the yanked line or lines from the buffer. |

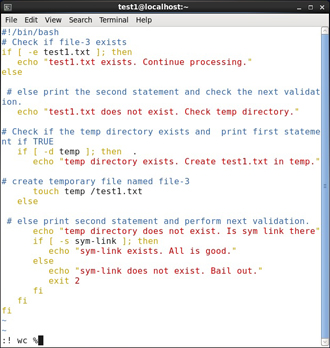
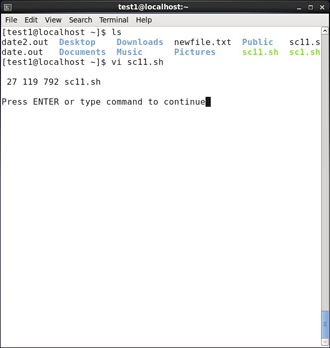
**Using External Commands**

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 %

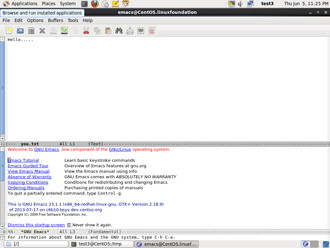
Typing this will run the wc (word count) command on the file; the character % represents the file currently being edited.

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.

[](https://courses.edx.org/c4x/LinuxFoundationX/LFS101x/asset/LFS01_ch10_screen25a.jpg)[](https://courses.edx.org/c4x/LinuxFoundationX/LFS101x/asset/LFS01_ch10_screen25b.jpg)

Click the image to view an enlarged version.

**Introduction to emacs**

[](https://courses.edx.org/c4x/LinuxFoundationX/LFS101x/asset/LFS01_ch10_screen19.jpg)

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.

**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 left |
| CTRL-b | One character right |
| 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 |

**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 |

**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 | Mark the beginning of the selected region. The end will be at the cursor position |
| CTRL-w | Yank (cut) the current marked region and put it in buffer |
| CTRL-y | Paste at the current position the yanked line or lines from the buffer |

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

[commands for emacs](https://learningmate.s3-us-west-2.amazonaws.com/LFS01/Chapter10/labs/emacs.pdf)

**Summary**



You have completed this chapter. Let’s summarize the key concepts covered.

* + 1. **Text editors** (rather than word processing programs) are used quite often in Linux, for tasks such as for creating or modifying system configuration files, writing scripts, developing source code, etc.
    2. **nano** is an easy-to-use text-based editor that utilizes on-screen prompts.
    3. **gedit** is a graphical editor very similar to **Notepad** in **Windows**.
    4. The **vi** editor is available on all Linux systems and is very widely used. Graphical extension versions of **vi**are widely available as well.
    5. **emacs** is available on all Linux systems as a popular alternative to **vi**. **emacs** can support both a graphical user interface and a text mode interface.
    6. To access the **vi** tutorial, type **vimtutor**at a command line window.
    7. To access the **emacs**tutorial type **Ctl-h** and then **t**from within **emacs.**
    8. **vi** has three modes: **Command**, **Insert**, and **Line; emacs** has only one but requires use of special keys such as Control and Escape.

Both editors use various combinations of keystrokes to accomplish tasks; the learning curve to master these can be long but once mastered using either editor is extremely efficient.

**Chapter 11: Local Principles**

**Learning Objectives**



By the end of this chapter, you should:

* Have a good grasp of best practices and tools for making Linux systems as secure as possible.
* Understand the powers and dangers of using the **root** (**superuser**) account.
* Know how to use the **sudo** command to perform privileged operations while restricting enhanced powers as much as feasible.
* Be able to explain the importance of process isolation and hardware access.
* Know how to work with passwords, including how to set and change them.

Describe how to secure the boot process and hardware resources.

|  |  |  |
| --- | --- | --- |
| **Field Name** | **Details** | **Remarks** |
| Username | User login name | Should be between 1 and 32 characters long |
| Password | User password (or the character **x**if the password is stored in the /etc/shadow file) in encrypted format | Is never shown in Linux when it is being typed; this stops prying eyes |
| User ID (UID) | Every user must have a user id (UID) | * UID 0 is reserved for root user * UID's ranging from 1-99 are reserved for other predefined accounts * UID's ranging from 100-999 are reserved for system accounts and groups (except for RHEL, which reserves only up to 499) * Normal users have UID's of 1000 or greater, except on RHEL where they start at 500 |
| Group ID (GID) | The primary Group ID (GID); Group Identification Number stored in the /etc/group file | Will be covered in detail in the chapter on Processes |
| User Info | This field is optional and allows insertion of extra information about the user such as their name | For example: Rufus T. Firefly |
| Home Directory | The absolute path location of user's home directory | For example: /home/rtfirefly |
| Shell | The absolute location of a user's default shell | For example: /bin/bash |

**User Accounts**

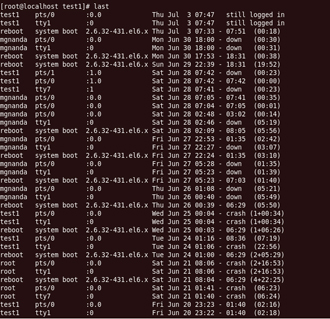
The Linux kernel allows properly authenticated users to access files and applications. While each user is identified by a unique integer (the user id or **UID),** a separate database associates a **username** with each UID. Upon account creation, new user information is added to the user database and the user's home directory must be created and populated with some essential files. Command line programs such as **useradd** and **userdel**as well asGUI tools are used for creating and removing accounts.

For each user, the following seven fields are maintained in the /etc/passwd file:

For each user, the following seven fields are maintained in the /etc/passwd file:

|  |  |  |
| --- | --- | --- |
| **Field Name** | **Details** | **Remarks** |
| Username | User login name | Should be between 1 and 32 characters long |
| Password | User password (or the character **x**if the password is stored in the /etc/shadow file) in encrypted format | Is never shown in Linux when it is being typed; this stops prying eyes |
| User ID (UID) | Every user must have a user id (UID) | * UID 0 is reserved for root user * UID's ranging from 1-99 are reserved for other predefined accounts * UID's ranging from 100-999 are reserved for system accounts and groups (except for RHEL, which reserves only up to 499) * Normal users have UID's of 1000 or greater, except on RHEL where they start at 500 |
| Group ID (GID) | The primary Group ID (GID); Group Identification Number stored in the /etc/group file | Will be covered in detail in the chapter on Processes |
| User Info | This field is optional and allows insertion of extra information about the user such as their name | For example: Rufus T. Firefly |
| Home Directory | The absolute path location of user's home directory | For example: /home/rtfirefly |
| Shell | The absolute location of a user's default shell | For example: /bin/bash |

**Types of Accounts**

[](https://courses.edx.org/c4x/LinuxFoundationX/LFS101x/asset/LFS01_ch018_screen4.jpg)

By default, Linux distinguishes between several account types in order to isolate processes and workloads. Linux has four types of accounts:

* root
* System
* Normal
* Network

For a safe working environment, it is advised to grant the minimum privileges possible and necessary to accounts, and remove inactive accounts. The **last** utility, which shows the last time each user logged into the system, can be used to help identify potentitally inactive accounts which are candidates for system removal.

Keep in mind that practices you use on multi-user business systems are more strict than practices you can use on personal desktop systems that only affect the casual user. This is especially true with security. We hope to show you practices applicable to enterprise servers that you can use on all systems, but understand that you may choose to relax these rules on your own personal system.

**Understanding the root Account**

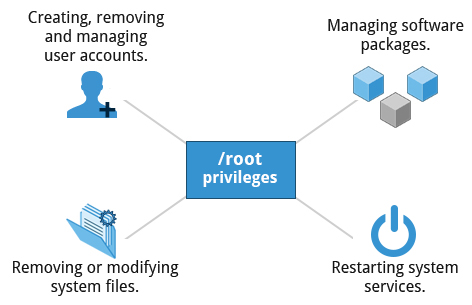


**root** is the most privileged account on a Linux/UNIX system. This account has the ability to carry out all facets of system administration, including adding accounts, changing user passwords, examining log files, installing software, etc. Utmost care must be taken when using this account. It has no security restrictions imposed upon it.

When you are signed in as, or acting as **root**, the shell prompt displays **'#**'  (if you are using **bash** and you haven’t customized the  prompt as we discuss elsewhere in this course). This convention is intended to serve as a warning to you of the absolute power of this account.

**Operations that Require root Privileges**

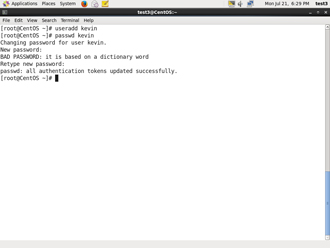
**root**privileges are required to perform operations such as:



* Creating, removing and managing user accounts.
* Managing software packages.
* Removing or modifying system files.
* Restarting system services.

Regular account users of Linux distributions may be allowed to install software packages, update some settings, and apply various kinds of changes to the system. However, **root** privilege is required for performing administration tasks such as restarting services, manually installing packages and managing parts of the filesystem that are outside the normal user’s directories.

**Creating a New User in Linux**

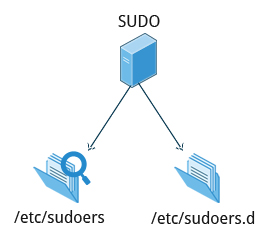
[](https://courses.edx.org/c4x/LinuxFoundationX/LFS101x/asset/LFS01_ch018_screen9.jpg)

To create a new user account:

* At the command prompt, as root type useradd <username> and press the **ENTER** key.
* To set the initial password, type passwd <username>  and press the **ENTER** key. The **New password**: prompt is displayed.
* Enter the password and press the **ENTER** key. To confirm the password, the prompt **Retype new password**: is displayed.
* Enter the password again and press the **ENTER** key. The message **passwd: all authentication tokens updated successfully**. is displayed.
* **Operations That Do Not Require root Privileges**
* A regular account user can perform some operations requiring special permissions; however, the system configuration must allow such abilities to be exercised.
* SUID (Set owner User ID upon execution—similar to the Windows "run as" feature) is a special kind of file permission given to a file. SUID provides temporary permissions to a user to run a program with the permissions of the file **owner**  (which may be root) instead of the permissions held by the user.
* The table provides examples of operations which do not require root privileges:

|  |  |
| --- | --- |
| **Operations that do not require Root privilege** | **Examples of this operation** |
| Running a network client | Sharing a file over the network |
| Using devices such as printers | Printing over the network |
| Operations on files that the user has proper permissions to access | Accessing files that you have access to or sharing data over the network |
| Running SUID-root applications | Executing programs such as passwd. |

**sudo Features**

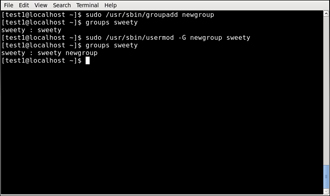


**sudo**has the ability to keep track of unsuccessful attempts at gaining root access. Users' authorization for using**sudo** is based on configuration information stored in the /etc/sudoers file and in the /etc/sudoers.d directory.

A message such as the following would appear in a system log file (usually /var/log/secure) when trying to execute **sudo bash** without successfully authenticating the user:

authentication failure; logname=op uid=0 euid=0 tty=/dev/pts/6 ruser=op rhost= user=op conversation failed auth could not identify password for [op] op : 1 incorrect password attempt ; TTY=pts/6 ; PWD=/var/log ; USER=root ; COMMAND=/bin/bash

**The sudoers File**

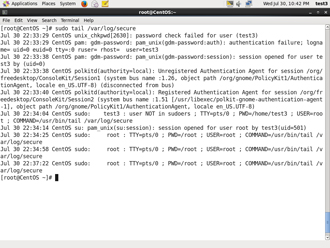
[](https://courses.edx.org/c4x/LinuxFoundationX/LFS101x/asset/LFS01_ch018_screen15.jpg)

Whenever **sudo**is invoked, a trigger will look at **/etc/sudoers** and the files in **/etc/sudoers.d** to determine if the user has the right to use **sudo** and what the scope of their privilege is. Unknown user requests and requests to do operations not allowed to the user even with **sudo**are reported. You can edit the **sudoers** file by using **visudo**, which ensures that only one person is editing the file at a time, has the proper permissions, and refuses to write out the file and exit if there is an error in the changes made.

The basic structure of an entry is: who where = (as\_whom) what

The file has a lot of documentation in it about how to customize. Most Linux distributions now prefer you add a file in the directory **/etc/sudoers.d** with a name the same as the user. This file contains the individual user's **sudo** configuration, and one should leave the master configuration file untouched except for changes that affect all users.

**Command Logging**

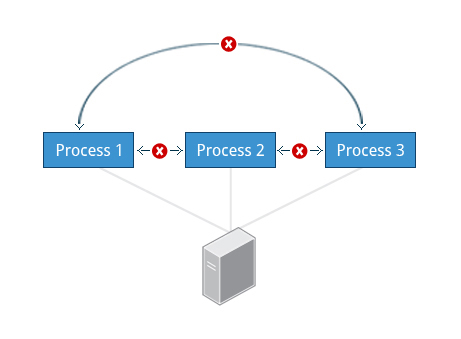
[](https://courses.edx.org/c4x/LinuxFoundationX/LFS101x/asset/LFS01_ch18_screen17a.jpg)

By default, **sudo** commands and any failures are logged in /var/log/auth.log under the **Debian** distribution family, and in /var/log/messages or /var/log/secure on other systems. This is an important safeguard to allow for tracking and accountability of **sudo** use. A typical entry of the message contains:

* Calling username
* Terminal info
* Working directory
* User account invoked
* Command with arguments

Running a command such as sudo whoami results in a log file entry such as: Dec 8 14:20:47 server1 sudo: op : TTY=pts/6 PWD=/var/log USER=root COMMAND=/usr/bin/whoami

**Process Isolation**



Linux is considered to be more secure than many other operating systems because processes are naturally **isolated** from each other. One process normally cannot access the resources of another process, even when that process is running with the same user privileges. Linux thus makes it difficult (though certainly not impossible) for viruses and security exploits to access and attack random resources on a system.

Additional security mechanisms that have been recently introduced in order to make risks even smaller are:

* **Control Groups (cgroups)**: Allows system administrators to group processes and associate finite resources to each cgroup.
* **Linux Containers (LXC)**: Makes it possible to run multiple isolated Linux systems (containers) on a single system by relying on **cgroups**.

**Virtualization**: Hardware is emulated in such a way that not only processes can be isolated, but entire systems are run simultaneously as isolated and insulated guests (virtual machines) on one physical host.

**Hardware Device Access**

[](https://courses.edx.org/c4x/LinuxFoundationX/LFS101x/asset/LFS01_ch018_screen19.jpg)

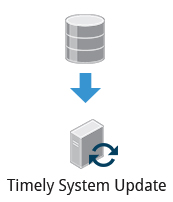
Linux limits user access to non-networking hardware devices in a manner that is extremely similar to regular file access. Applications interact by engaging the filesystem layer (which is independent of the actual device or hardware the file resides on). This layer will then opens a **device special file** (often called a **device node**) under the **/dev** directory that corresponds to the device being accessed. Each device special file has standard owner, group and world permission fields. Security is naturally enforced just as it is when standard files are accessed.

Hard disks, for example, are represented as **/dev/sd\***. While a root user can read and write to the disk in a **raw** fashion (for example, by doing something like:

 $ echo hello world > /dev/sda1

the standard permissions as shown in the figure make it impossible for regular users to do so.  Writing to a device in this fashion can easily obliterate the filesystem stored on it in a way that cannot be repaired without great effort, if at all.  The normal reading and writing of files on the hard disk by applications is done at a higher level through the filesystem, and never through direct access to the device node.

**Keeping Current**

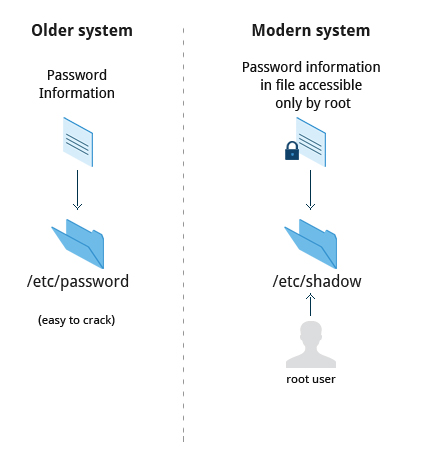


When security problems in either the Linux kernel or applications and libraries are discovered, Linux distributions have a good record of reacting quickly and pushing out fixes to all systems by updating their software repositories and sending notifications to update immediately. The same thing is true with bug fixes and performance improvements that are not security related.

However, it is well known that many systems do not get updated frequently enough and problems which have already been cured are allowed to remain on computers for a long time; this is particularly true with proprietary operating systems where users are either uninformed or distrustful of the vendor's patching policy as sometimes updates can cause new problems and break existing operations. Many of the most successful attack vectors come from exploiting security holes for which fixes are already known but not universally deployed.

So the best practice is to take advantage of your Linux distribution's mechanism for automatic updates and never postpone them. It is extremely rare that such an update will cause new problems.

**How Passwords are Stored**



The system verifies authenticity and identity using user credentials. Originally, encrypted passwords were stored in the /etc/passwd file, which was readable by everyone. This made it rather easy for passwords to be cracked. On modern systems, passwords are actually stored in an encrypted format in a secondary file named **/etc/shadow**. Only those with **root access** can modify/read this file.

**Password Encryption**

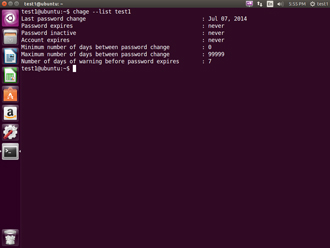
[](https://courses.edx.org/c4x/LinuxFoundationX/LFS101x/asset/LFS01_ch018_screen23.jpg)

Protecting passwords has become a crucial element of security. Most Linux distributions rely on a modern password encryption algorithm called **SHA-512** (Secure Hashing Algorithm 512 bits), developed by the U.S. National Security Agency (NSA) to encrypt passwords.

The **SHA-512** algorithm is widely used for security applications and protocols. These security applications and protocols include TLS, SSL, PHP, SSH, S/MIME and IPSec. **SHA-512** is one of the most tested hashing algorithms.

For example, if you wish to experiment with **SHA-512** encoding, the word “test” can be encoded using the program **sha512sum**to produce the **SHA-512** form (see graphic):

**Good Password Practices**

[](https://courses.edx.org/c4x/LinuxFoundationX/LFS101x/asset/LFS01_ch018_screen24.jpg)

IT professionals follow several good practices for securing the data and the password of every user.

* **Password aging** is a method to ensure that users get prompts that remind them to create a new password after a specific period. This can ensure that passwords, if cracked, will only be usable for a limited amount of time. This feature is implemented using **chage,** which configures the password expiry information for a user.
* Another method is to force users to set strong passwords using **Pluggable Authentication Modules (PAM)**. **PAM** can be configured to automatically verify that a password created or modified using the **passwd**utility is sufficiently strong. **PAM** configuration is implemented using a library called **pam\_cracklib.so**, which can also be replaced by **pam\_passwdqc.so** for more options.
* One can also install password cracking programs, such as **Jack The Ripper**, to secure the password file and detect weak password entries. It is recommended that written authorization be obtained before installing such tools on any system that you do not own.
* **Requiring Boot Loader Passwords**
* [](https://courses.edx.org/c4x/LinuxFoundationX/LFS101x/asset/LFS01_ch018_screen29.jpg)
* You can secure the boot process with a secure password to prevent someone from bypassing the user authentication step. For systems using the **GRUB** boot loader, for the older **GRUB**version 1*,*you can invoke **grub-md5-crypt** which will prompt you for a password and then and then encrypt as shown on the adjoining screen.
* You then must edit /boot/grub/grub.conf by adding the following line below the timeout entry:
* password --md5 $1$Wnvo.1$qz781HRVG4jUnJXmdSCZ30
* You can also force passwords for only certain boot choices rather than all.
* For the now more common **GRUB**version 2 things are more complicated, and you have more flexibility and can do things like use user-specific passwords, which can be their normal login password.  Also you never edit the configuration file, /boot/grub/grub.cfg, directly, rather you edit system configuration files in /etc/grub.d and then run **update-grub**. One explanation of this can be found at <https://help.ubuntu.com/community/Grub2/Passwords>.

**Hardware Vulnerability**



When hardware is physically accessible, security can be compromised by:

* Key logging: Recording the real time activity of a computer user including the keys they press. The captured data can either be stored locally or transmitted to remote machines
* Network sniffing: Capturing and viewing the network packet level data on your network
* Booting with a live or rescue disk
* Remounting and modifying disk content

Your IT security policy should start with requirements on how to properly secure physical access to servers and workstations. Physical access to a system makes it possible for attackers to easily leverage several attack vectors, in a way that makes all operating system level recommendations irrelevant.

The guidelines of security are:

* Lock down workstations and servers
* Protect your network links such that it cannot be accessed by people you do not trust
* Protect your keyboards where passwords are entered to ensure the keyboards cannot be tampered with
* Ensure a password protects the BIOS in such a way that the system cannot be booted with a live or rescue DVD or USB key

For single user computers and those in a home environment some of the above features (like preventing booting from removable media) can be excessive, and you can avoid implementing them. However, if sensitive information is on your system that requires careful protection, either it shouldn't be there or it should be better protected by following the above guidelines.

**Summary (1 of 2)**



You have completed this chapter. Let’s summarize the key concepts covered:

* The **root** account has authority over the entire system.
* **root** privileges may be required for tasks, such as restarting services, manually installing packages and managing parts of the filesystem that are outside your home directory.
* In order to perform any privileged operations such as system-wide changes, you need to use either **su** or **sudo.**
* Calls to **sudo**trigger a lookup in the**/etc/sudoers** file, or in the **/etc/sudoers.d** directory which first validates that the calling user is allowed to use**sudo**and that it is being used within permitted scope
* One of the most powerful features of **sudo**is its ability to log unsuccessful attempts at gaining root access.  By default **sudo** commands and failures are logged in **/var/log/auth.log** under the **Debian** family and **/var/log/messages** in other distribution families.

**Summary (2 of 2)**



* One process cannot access another process’ resources, even when that process is running with the same user privileges.
* Using the user credentials, the system verifies the authenticity and identity.
* The SHA-512 algorithm is typically used to encode passwords. They can be encrypted but not decrypted.
* Pluggable Authentication Modules (**PAM**) can be configured to automatically verify that passwords created or modified using the **passwd** utility are strong enough (what is considered strong enough can also be configured).
* Your IT security policy should start with requirements on how to properly secure physical access to servers and workstations.

Keeping your systems updated is an important step in avoiding security attacks.

**Chapter 12: Network Operations**

**Learning Objectives**



By the end of this chapter, you should be able to:

* Explain many basic networking concepts including types of networks and addressing issues.
* Know how to configure network interfaces and use basic networking utilties, such as **ifconfig**, **ip**, **ping**, **route** & **traceroute**.
* Use graphical and non-graphical browsers, such as **Lynx**, **w3m**, **Firefox**, **Chrome** and **Epiphany**.
* Transfer files to and from clients and servers using both graphical and text mode applications, such as **Filezilla**, **ftp**, **sftp**, **curl** and **wget**.

**Introduction to Networking**



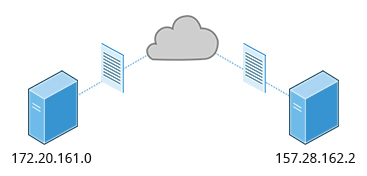
A network is a group of computers and computing devices connected together through communication channels, such as cables or wireless media. The computers connected over a network may be located in the same geographical area or spread across the world.

A network is used to:

* Allow the connected devices to communicate with each other.
* Enable multiple users to share devices over the network, such as printers and scanners.
* Share and manage information across computers easily.

Most organizations have both an internal network and an Internet connection for users to communicate with machines and people outside the organization. The Internet is the largest network in the world and is often called "the network of networks".

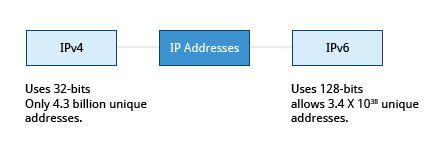
**IP Addresses**



Devices attached to a network must have at least one unique network address identifier known as the IP (**Internet Prococol)**address. The address is essential for routing **packets** of information through the network.

Exchanging information across the network requires using streams of bite-sized packets, each of which contains a piece of the information going from one machine to another. These packets contain **data buffers** together with **headers** which contain information about where the packet is going to and coming from, and where it fits in the sequence of packets that constitute the stream. Networking protocols and software are rather complicated due to the diversity of machines and operating systems they must deal with, as well as the fact that even very old standards must be supported.

**IPv4 and IPv6**



There are two different types of IP addresses available: **IPv4** (version 4) and **IPv6** (version 6). **IPv4** is older and by far the more widely used, while **IPv6** is newer and is designed to get past the limitations of the older standard and furnish many more possible addresses.

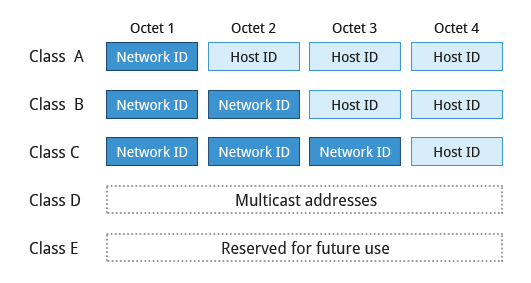
**IPv4** uses 32-bits for addresses; there are ***only*** 4.3 billion unique addresses available.  Furthermore, many addresses are allotted and reserved but not actually used. **IPv4** is becoming inadequate because the number of devices available on the global network has significantly increased over the past years.

**IPv6** uses 128-bits for addresses; this allows for 3.4 X 1038 unique addresses. If you have a larger network of computers and want to add more, you may want to move to **IPv6**, because it provides more unique addresses. However, it is difficult to move to **IPv6** as the two protocols do not inter-operate. Due to this, migrating equipment and addresses to **IPv6** requires significant effort and hasn't been as fast as was originally intended.

**Decoding IPv4 Addresses**

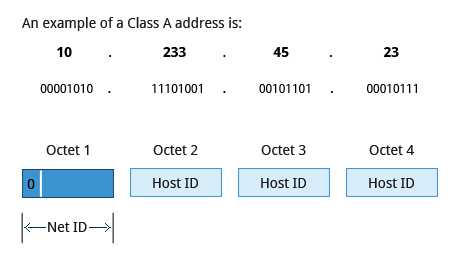
A 32-bit IPv4 address is divided into four 8-bit sections called [octets](http://en.wikipedia.org/wiki/Octet_(computing)).

Example: IP address →            172  .          16  .          31  .         46 Bit format →     10101100.00010000.00011111.00101110



Network address are divided into five classes: A, B, C, D, and E. Classes A, B, and C are classified into two parts: **Network addresses (Net ID)** and **Host address (Host ID)**. The Net ID is used to identify the network, while the Host ID is used to identify a host in the network. Class D is used for special multicast applications (information is broadcast to multiple computers simultaneously) and Class E is reserved for future use. In this section you will learn about classes A, B, and C.

**Class A Network Addresses**

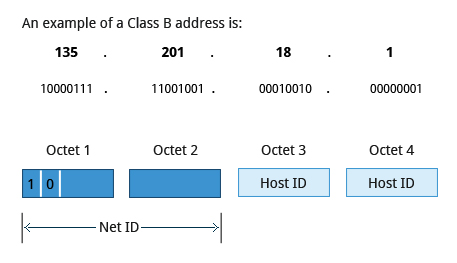


Class A addresses use the first octet of an IP address as their Net ID and use the other three octets as the **Host ID.** The first bit of the first octet is always set to zero. So you can use only 7-bits for unique network numbers. As a result, there are a maximum of 127 Class A networks available. Not surprisingly, this was only feasible when there were very few unique networks with large numbers of hosts. As the use of the Internet expanded, Classes B and C were added in order to accomodate the growing demand for independent networks.

Each Class A network can have up to 16.7 million unique hosts on its network. The range of host address is from 1.0.0.0 to 127.255.255.255.

**Note: The value of an octet, or 8-bits, can range from 0 to 255.**

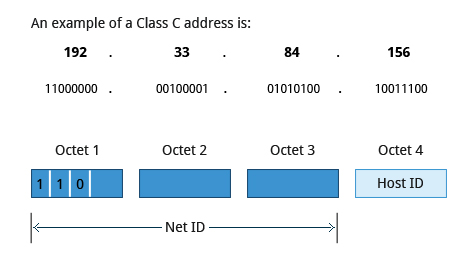
**Class B Network Addresses**



Class B addresses use the first two octets of the IP address as their Net ID and the last two octets as the Host ID. The first two bits of the first octet are always set to binary 10, so there are a maximum of 16,384 (14-bits) Class B networks. The first octet of a Class B address has values from 128 to 191. The introduction of Class B networks expanded the number of networks but it soon became clear that a further level would be needed.

Each Class B network can support a maximum of 65,536 unique hosts on its network. The range of host address is from 128.0.0.0 to 191.255.255.255.

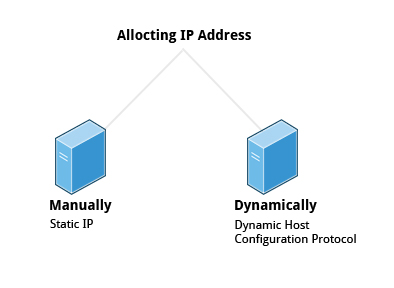
**Class C Network Addresses**



Class C addresses use the first three octets of the IP address as their Net ID and the last octet as their Host ID. The first three bits of the first octet are set to binary 110, so almost 2.1 million (21-bits) Class C networks are available. The first octet of a Class C address has values from 192 to 223. These are most common for smaller networks which don't have many unique hosts.

Each Class C network can support up to 256 (8-bits) unique hosts. The range of host address is from 192.0.0.0 to 223.255.255.255.

**IP Address Allocation**



Typically, a range of IP addresses are requested from your Internet Service Provider (ISP) by your organization's network administrator. Often your choice of which class of IP address you are given depends on the size of your network and expected growth needs.

You can assign IP addresses to computers over a network manually or dynamically. When you assign IP addresses manually, you add**static** (never changing) addresses to the network. When you assign IP addresses dynamically (they can change everytime you reboot or even more often), the **Dynamic Host Configuration Protocol (DHCP)** is used to assign IP addresses.

**Manually Allocating an IP Address**

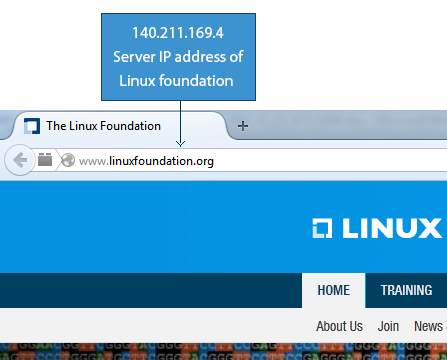
Before an IP address can be allocated manually, one must identify the size of the network by determining the host range; this determines which network class (A, B, or C) can be used. The **ipcalc**program can be used to ascertain the host range.

**Note: The version of ipcalc supplied in the Fedora family of distributions does not behave as described below, it is really a different program.**

Assume that you have a Class C network. The first three octets of the IP address are 192.168.0. As it uses 3 octets (i.e. 24 bits) for the network mask, the shorthand for this type of address is 192.168.0.0/24. To determine the host range of the address you can use for this new host, at the command prompt, type: ipcalc 192.168.0.0/24 and press **Enter**.

From the result, you can check the **HostMin** and **HostMax** values to manually assign a static address available from 1 to 254 (192.168.0.1 to 192.168.0.254).

**Name Resolution**



**Name Resolution** is used to convert numerical IP address values into a human-readable format known as the **hostname**. For example, 140.211.169.4 is the numerical IP address that refers to the **linuxfoundation.org** hostname. Hostnames are easier to remember.

Given an IP address, you can obtain its corresponding hostname. Accessing the machine over the network becomes easier when you can type the hostname instead of the IP address.

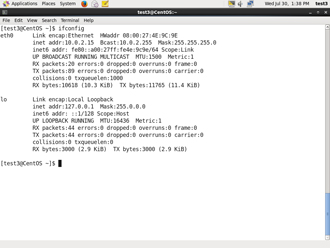
You can view your system’s hostname simply by typing **hostname**with no argument.

**Note: If you give an argument, the system will try to change its hostname to match it, however, only root users can do that.**

The special hostname **localhost** is associated with the IP address 127.0.0.1**,** and describes the machine you are currently on (which normally has additional network-related IP addresses).

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

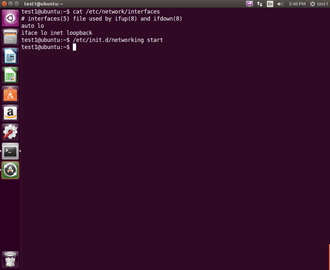
**Network Interfaces**

[](https://courses.edx.org/c4x/LinuxFoundationX/LFS101x/asset/LFS01_ch11_screen15.jpg)

Network interfaces are a connection channel between a device and a network. Physically, network interfaces can proceed through a **network interface card** (**NIC**) or can be more abstractly implemented as software. You can have multiple network interfaces operating at once. Specific interfaces can be brought up (activated) or brought down (de-activated) at any time.

A list of currently active network interfaces is reported by the **ifconfig** utility which you may have to run as the superuser, or at least, give the full path, i.e., /sbin/ifconfig, on some distributions.

**Network Configuration Files**

[](https://courses.edx.org/c4x/LinuxFoundationX/LFS101x/asset/LFS01_ch11_screen16.jpg)

Network configuration files are essential to ensure that interfaces function correctly.

For **Debian** family configuration, the basic network configuration file is /etc/network/interfaces. You can type /etc/init.d/networking start to start the networking configuration.

For **Fedora** family system configuration, the routing and host information is contained in /etc/sysconfig/network. The network interface configuration script is located at /etc/sysconfig/network-scripts/ifcfg-eth0.

For **SUSE** family system configuration, the routing and host information and network interface configuration scripts are contained in the /etc/sysconfig/network directory.

You can type /etc/init.d/network start to start the networking configuration for **Fedora** and **SUSE** families.

**Network Configuration Commands**

[](https://courses.edx.org/c4x/LinuxFoundationX/LFS101x/asset/LFS01_ch11_screen17.jpg)

To view the IP address: $ /sbin/ip addr show

To view the routing information: $ /sbin/ip route show

**ip** is a very powerful program that can do many things. Older (and more specific) utilties such as **ifconfig** and **route**are often used to accomplish similar tasks. A look at the relevant **man pages** can tell you much more about these utilties.

**Network Configuration Commands**

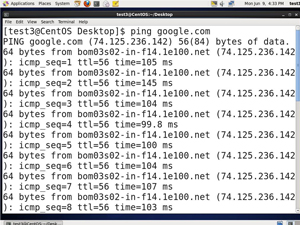
[](https://courses.edx.org/c4x/LinuxFoundationX/LFS101x/asset/LFS01_ch11_screen17.jpg)

To view the IP address: $ /sbin/ip addr show

To view the routing information: $ /sbin/ip route show

**ip** is a very powerful program that can do many things. Older (and more specific) utilties such as **ifconfig** and **route**are often used to accomplish similar tasks. A look at the relevant **man pages** can tell you much more about these utilties.

**ping**

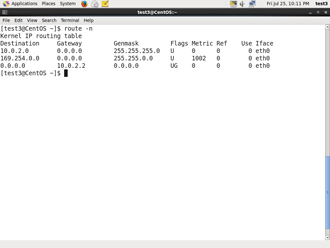
[](https://courses.edx.org/c4x/LinuxFoundationX/LFS101x/asset/LFS01_ch11_screen18.jpg)

**ping** is used to check whether or not a machine attached to the network can receive and send data; i.e., it confirms that the remote host is online and is responding.

To check the status of the remote host, at the command prompt, type ping <hostname>.

**ping**is frequently used for network testing and management; however, its usage can increase network load unacceptably. Hence, you can abort the execution of **ping** by typing **CTRL-C**, or by using the **-c** option, which limits the number of packets that **ping** will send before it quits. When execution stops, a summary is displayed.

**route**

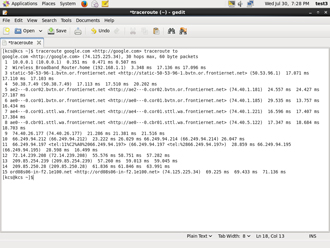
[](https://courses.edx.org/c4x/LinuxFoundationX/LFS101x/asset/LFS01_ch11_screen19.jpg)

A network requires the connection of many nodes. Data moves from source to destination by passing through a series of routers and potentially across multiple networks. Servers maintain **routing tables**containing the addresses of each node in the network. The **IP Routing protocols** enable routers to build up a forwarding table that correlates final destinations with the next **hop** addresses.

**route**is used to view or change the IP routing table. You may want to change the IP routing table to add, delete or modify specific (static ) routes to specific hosts or networks. The table explains some commands that can be used to manage IP routing.

|  |  |
| --- | --- |
| **Task** | **Command** |
| Show current routing table | $ route –n |
| Add static route | $ route add -net address |
| Delete static route | $ route del -net address |

**traceroute**

[](https://courses.edx.org/c4x/LinuxFoundationX/LFS101x/asset/LFS01_ch11_screen20.jpg)

**traceroute** is used to inspect the route which the data packet takes to reach the destination host which makes it quite useful for troubleshooting network delays and errors. By using **traceroute** you can isolate connectivity issues between **hops**, which helps resolve them faster.

To print the route taken by the packet to reach the network host, at the command prompt, type traceroute <domain>.

Click the image to view an enlarged version.

**More Networking Tools**

Now, let’s learn about some additional networking tools. Networking tools are very useful for monitoring and debugging network problems, such as network connectivity and network traffic.

|  |  |
| --- | --- |
| **Networking Tools** | **Description** |
| ethtool | Queries network interfaces and can also set various parameters such as the speed. |
| netstat | Displays all active connections and routing tables. Useful for monitoring performance and troubleshooting. |
| nmap | Scans open **ports** on a network. Important for security analysis |
| tcpdump | Dumps network traffic for analysis. |
| iptraf | Monitors network traffic in text mode. |

**Graphical and Non-Graphical Browsers**

**Browsers** are used to retrieve, transmit, and explore information resources, usually on the **World Wide Web**. Linux users commonly use both graphical and non-graphical browser applications.

The common graphical browsers used in Linux are:

* **Firefox**
* **Google Chrome**
* **Chromium**
* **Epiphany**
* **Opera**

Sometimes you either do not have a graphical environment to work in (or have reasons not to use it) but still need to access web resources. In such a case, you can use non-graphical browsers such as the following:

|  |  |
| --- | --- |
| **Non-Graphical Browsers** | **Description** |
| [lynx](http://lynx.browser.org/) | Configurable text-based web browser; the earliest such browser and still in use. |
| [links or elinks](http://elinks.or.cz/) | Based on**lynx.** It can display tables and frames. |
| [w3m](http://w3m.sourceforge.net/) | Newer text-based web browser with many features. |

**wget**

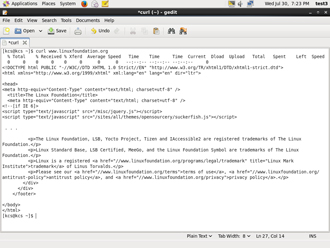
[](https://courses.edx.org/c4x/LinuxFoundationX/LFS101x/asset/LFS01_ch11_screen28.jpg)

Sometimes you need to download files and information but a browser is not the best choice, either because you want to download multiple files and/or directories, or you want to perform the action from a command line or a script. **wget** is a command line utility that can capably handle the following types of downloads:

* Large file downloads
* Recursive downloads, where a web page refers to other web pages and all are downloaded at once
* Password-required downloads
* Multiple file downloads

To download a webpage, you can simply type wget <url>, and then you can read the downloaded page as a local file using a graphical or non-graphical browser.

**curl**

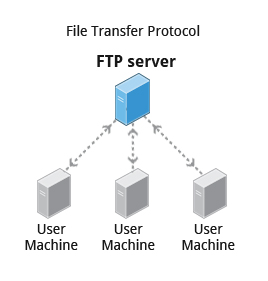
[](https://courses.edx.org/c4x/LinuxFoundationX/LFS101x/asset/LFS01_ch11_screen29.jpg)

Besides downloading you may want to obtain information about a URL, such as the source code being used. **curl** can be used from the command line or a script to read such information. **curl**also allows you to save the contents of a web page to a file as does **wget.**

You can read a URL using curl <URL>. For example, if you want to read [http://www.linuxfoundation.org](http://www.linuxfoundation.org/) , type curl [http://www.linuxfoundation.org](http://www.linuxfoundation.org/).

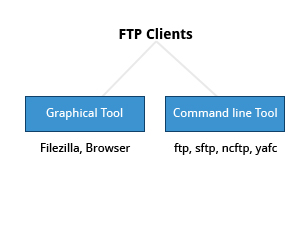
To get the contents of a web page and store it to a file, type curl -o saved.html [http://www.mysite.com](http://www.mysite.com/). The contents of the main index file at the website will be saved in saved.html.

**FTP (File Transfer Protocol)**



When you are connected to a network, you may need to transfer files from one machine to another. **File Transfer Protocol (FTP)** is a well-known and popular method for transferring files between computers using the Internet. This method is built on a **client-server** model. FTP can be used within a browser or with standalone client programs.

**FTP Clients**



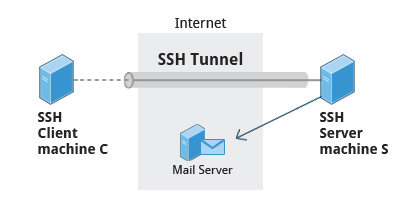
FTP clients enable you to transfer files with remote computers using the FTP protocol. These clients can be either graphical or command line tools. **Filezilla**, for example, allows use of the drag-and-drop approach to transfer files between hosts. All web browsers support FTP, all you have to do is give a URL like : ftp://ftp.kernel.org where the usual http:// becomes ftp://.

Some command line FTP clients are:

* **ftp**
* **sftp**
* **ncftp**
* **yafc** (Yet Another FTP Client)

**sftp** is a very secure mode of connection, which uses the **Secure Shell** (**ssh)** protocol, which we will discuss shortly. **sftp** encrypts its data and thus sensitive information is transmitted more securely. However, it does not work with so-called **anonymous FTP** (guest user credentials). Both **ncftp** and **yafc** are also powerful FTP clients which work on a wide variety of operating systems including **Windows** and **Linux**.

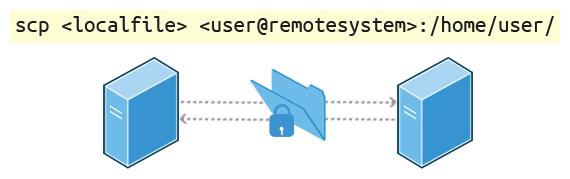
**SSH: Executing Commands Remotely**



**Secure Shell (SSH)** is a cryptographic network protocol used for secure data communication. It is also used for remote services and other secure services between two devices on the network and is very useful for administering systems which are not easily available to physically work on but to which you have remote access.

To run my\_command on a remote system via SSH, at the command prompt, type, ssh <remotesystem> my\_command and press **Enter**. **ssh** then prompts you for the remote password. You can also configure **ssh** to securely allow your remote access without typing a password each time.

**Copying Files Securely with scp**



We can also move files securely using **Secure Copy** (**scp**) between two networked hosts. **scp** uses the SSH protocol for transferring data.

To copy a local file to a remote system, at the command prompt, type scp <localfile> <user@remotesystem>:/home/user/ and press **Enter**.

You will receive a prompt for the remote password. You can also configure **scp** so that it does not prompt for a password for each transfer.

**Summary (1 of 2)**



You have completed this chapter. Let’s summarize the key concepts covered:

* 1. The **IP** (Internet Protocol) **address** is a unique logical network address that is assigned to a device on a network.
  2. **IPv4** uses 32-bits for addresses and **IPv6** uses 128-bits for addresses.
  3. Every IP address contains both a network and a host address field.
  4. There are five classes of network addresses available: A, B, C, D & E.
  5. **DNS** (Domain Name System) is used for converting Internet domain and host names to IP addresses.
  6. The **ifconfig** program is used to display current active network interfaces.
  7. The commands ip addr show and ip route show can be used to view IP address and routing information.

You can use**ping**to check if the remote host is alive and responding.

**Summary (2 of 2)**



* You can use the **route** utility program to manage IP routing.
* You can monitor and debug network problems using networking tools.
* **Firefox**, **Google Chrome**, **Chromium**, and **Epiphany** are the main graphical browsers used in Linux.
* Non-graphical or text browsers used in Linux are **Lynx**, **Links**, and **w3m**.
* You can use **wget** to download webpages.
* You can use **curl** to obtain information about URL's.
* **FTP**(File Transfer Protocol) is used to transfer files over a network.
* **ftp**, **sftp**, **ncftp**, and **yafc** are command line FTP clients used in Linux.

You can use **ssh**to run commands on remote systems.