

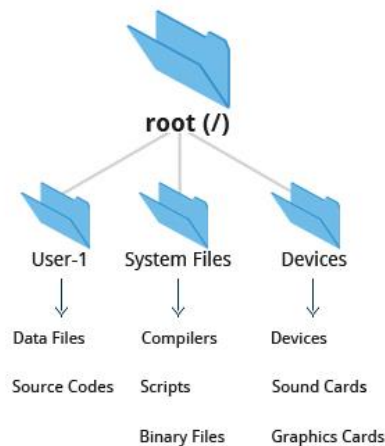
# Chapter 10

## Introduction to filesystems

In Linux (and all UNIX-like operating systems) it is often said “Everything is a file”, or at least it is treated as such. This means whether you are dealing with normal data files and documents, or with devices such as sound cards and printers, you interact with them through the same kind of Input/Output (I/O) operations. This simplifies things: you open a “file” and perform normal operations like reading the file and writing on it (which is one reason why text editors, which you will learn about in an upcoming section, are so important).

On many systems (including Linux), the filesystem is structured like a tree. The tree is usually portrayed as inverted, and starts at what is most often called the **root directory**, which marks the beginning of the hierarchical filesystem and is also sometimes referred to as the trunk, or simply denoted by **/**. The root directory is *not* the same as the root user. The hierarchical filesystem also contains other elements in the path (directory names), which are separated by forward slashes (**/**), as in **/usr/bin/emacs**, where the last element is the actual file name.

In this section, you will learn about some basic concepts, including the filesystem hierarchy, as well as about disk partitions.



## Filesystem varieties

Linux supports a number of native filesystem types, expressly created by Linux developers, such as:

- ext3
- ext4
- squashfs
- btrfs.

It also offers implementations of filesystems used on other alien operating systems, such as those from:

- Windows (ntfs, vfat)
- SGI (xfs)
- IBM (jfs)
- MacOS (hfs, hfs+).

Many older, legacy filesystems, such as FAT, are also supported.

It is often the case that more than one filesystem type is used on a machine, based on considerations such as the size of files, how often they are modified, what kind of hardware they sit on and what kind of access speed is needed, etc. The most advanced filesystem types in common use are the **journaling** varieties: ext4, xfs, btrfs, and jfs. These have many state-of-the-art features and high performance, and are very hard to corrupt accidentally.

## Linux Partitions

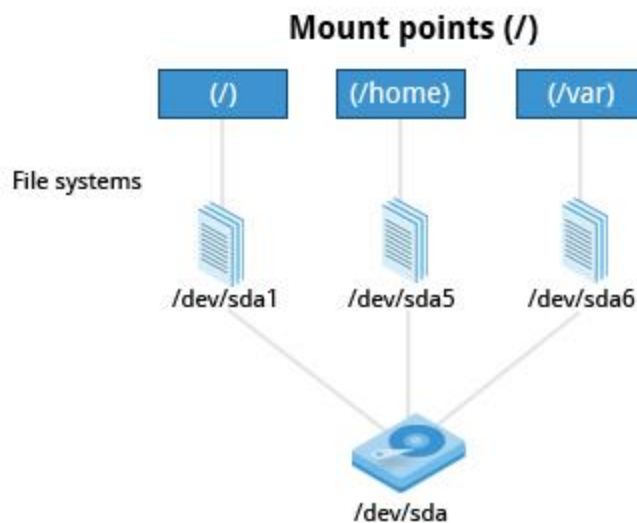
Each filesystem on a Linux system occupies a disk **partition**. Partitions help to organize the contents of disks according to the kind and use of the data contained. For example, important programs required to run the system are often kept on a separate partition (known as **root** or **/**) than the one that contains files owned by regular users of that system (**/home**). In addition, temporary files created and destroyed during the normal operation of Linux may be located on dedicated partitions. One advantage of this kind of isolation by type and variability is that when all available space on a particular partition is exhausted, the system may still operate normally.

The pictures shows the use of the **gparted** utility, which displays the partition layout on a system which has three operating systems on it: RHEL 8, CentOS 7, Ubuntu and Windows.

## Mount Points

Before you can start using a filesystem, you need to **mount** it on the filesystem tree at a mount point. This is simply a directory (which may or may not be empty) where the filesystem is to be grafted on. Sometimes, you may need to create the directory if it does not already exist.

**Warning:** If you mount a filesystem on a non-empty directory, the former contents of that directory are covered-up and not accessible until the filesystem is unmounted. Thus, mount points are usually empty directories.



## Mounting and Unmounting

The **mount** command is used to attach a filesystem (which can be local to the computer or on a network) somewhere within the filesystem tree. The basic arguments are the **device node** and mount point. For example,

```
$ sudo mount /dev/sda5 /home
```

will attach the filesystem contained in the disk partition associated with the **/dev/sda5** device node, into the filesystem tree at the **/home** mount point. There are other ways to specify the partition other than the device node, such as using the disk label or UUID.

To unmount the partition, the command would be:

**\$ sudo umount /home**

Note the command is **umount**, not unmount! Only a root user (logged in as root, or using **sudo**) has the privilege to run these commands, unless the system has been otherwise configured.

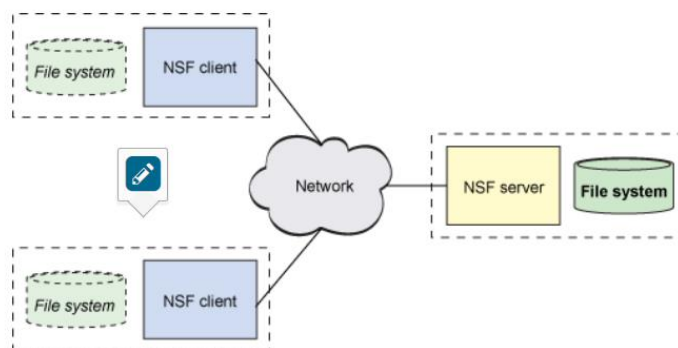
If you want it to be automatically available every time the system starts up, you need to edit **/etc/fstab** accordingly (the name is short for filesystem table). Looking at this file will show you the configuration of all pre-configured filesystems. **man fstab** will display how this file is used and how to configure it.

Executing **mount** without any arguments will show all presently mounted filesystems.

The command **df -Th** (disk-free) will display information about mounted filesystems, including the filesystem type, and usage statistics about currently used and available space.

## NFS and Network Filesystems

It is often necessary to share data across physical systems which may be either in the same location or anywhere that can be reached by the Internet. A network (also sometimes called distributed) filesystem may have all its data on one machine or have it spread out on more than one network node. A variety of different filesystems can be used locally on the individual machines; a network filesystem can be thought of as a grouping of lower level filesystems of varying types.



Many system administrators mount remote users' home directories on a server in order to give them access to the same files and configuration files across multiple client systems. This allows the users to log in to different computers, yet still have access to the same files and resources.

The most common such filesystem is named simply **NFS** (the **N**etwork **F**ile**s**ystem). It has a very long history and was first developed by Sun Microsystems. Another common implementation is **CIFS** (also termed **SAMBA**), which has Microsoft roots. We will restrict our attention in what follows to NFS.

## NFS on the server

We will now look in detail at how to use NFS on the server.

On the server machine, NFS uses **daemons** (built-in networking and service processes in Linux) and other system servers are started at the command line by typing:

```
$ sudo systemctl start nfs
```

The text file **/etc/exports** contains the directories and permissions that a host is willing to share with other systems over NFS. A very simple entry in this file may look like the following:

```
/projects *.example.com(rw)
```

This entry allows the directory **/projects** to be mounted using NFS with read and write (**rw**) permissions and shared with other hosts in the **example.com** domain. As we will detail in the next chapter, every file in Linux has three possible permissions: read (**r**), write (**w**) and execute (**x**).

After modifying the **/etc/exports** file, you can type **exportfs -av** to notify Linux about the directories you are allowing to be remotely mounted using NFS. You can also restart NFS with **sudo systemctl restart nfs**, but this is heavier, as it halts NFS for a short while before starting it up again. To make sure the NFS service starts whenever the system is booted, issue **sudo systemctl enable nfs**. (**Note:** On RHEL/CentOS 8, the service is called **nfs-server**, not **nfs**).

## NFS on the client

On the client machine, if it is desired to have the remote filesystem mounted automatically upon system boot, **/etc/fstab** is modified to accomplish this. For example, an entry in the client's **/etc/fstab** might look like the following:

```
servername:/projects /mnt/nfs/projects nfs defaults 0 0
```

You can also mount the remote filesystem without a reboot or as a one-time mount by directly using the **mount** command:

```
$ sudo mount servername:/projects /mnt/nfs/projects
```

Remember, if **/etc/fstab** is not modified, this remote mount will not be present the next time the system is restarted. Furthermore, you may want to use the **nofail** option in **fstab** in case the NFS server is not live at boot.

## User Home Directories

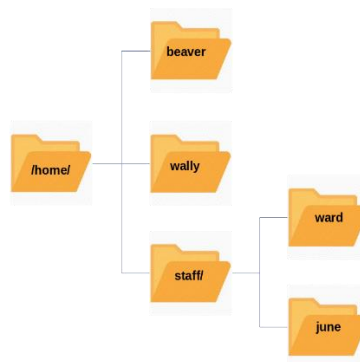
On the client machine, if it is desired to have the remote filesystem mounted automatically upon system boot, **/etc/fstab** is modified to accomplish this. For example, an entry in the client's **/etc/fstab** might look like the following:

```
servername:/projects /mnt/nfs/projects nfs defaults 0 0
```

You can also mount the remote filesystem without a reboot or as a one-time mount by directly using the **mount** command:

```
$ sudo mount servername:/projects /mnt/nfs/projects
```

Remember, if **/etc/fstab** is not modified, this remote mount will not be present the next time the system is restarted. Furthermore, you may want to use the **nofail** option in **fstab** in case the NFS server is not live at boot.



## The /bin and /sbin directories

The **/bin** directory contains executable binaries, essential commands used to boot the system or in single-user mode, and essential commands required by all system users, such as **cat**, **cp**, **ls**, **mv**, **ps**, and **rm**.

Likewise, the **/sbin** directory is intended for essential binaries related to system administration, such as **fsck** and **ip**. To view a list of these programs, type:

```
$ ls /bin /sbin
```

Commands that are not essential (theoretically) for the system to boot or operate in single-user mode are placed in the **/usr/bin** and **/usr/sbin** directories. Historically, this was done so **/usr** could be mounted as a separate filesystem that could be mounted at a later stage of system startup or even over a network. However, nowadays most find this distinction is obsolete. In fact, many distributions have been discovered to be unable to boot with this separation, as this modality had not been used or tested for a long time.

Thus, on some of the newest Linux distributions **/usr/bin** and **/bin** are actually just symbolically linked together, as are **/usr/sbin** and **/sbin**.

## The /proc filesystem

Commands that are not essential (theoretically) for the system to boot or operate in single-user mode are placed in the **/usr/bin** and **/usr/sbin** directories. Historically, this was done so **/usr** could be mounted as a separate filesystem that could be mounted at a later stage of system startup or even over a network. However, nowadays most find this distinction is obsolete. In fact, many distributions have been discovered to be unable to boot with this separation, as this modality had not been used or tested for a long time.

Thus, on some of the newest Linux distributions **/usr/bin** and **/bin** are actually just symbolically linked together, as are **/usr/sbin** and **/sbin**.

## The /dev directory

The **/dev** directory contains **device nodes**, a type of pseudo-file used by most hardware and software devices, except for network devices. This directory is:

- Empty on the disk partition when it is not mounted
- Contains entries which are created by the **udev** system, which creates and manages device nodes on Linux, creating them dynamically when devices are found. The **/dev** directory contains items such as:

1. **/dev/sda1** (first partition on the first hard disk)

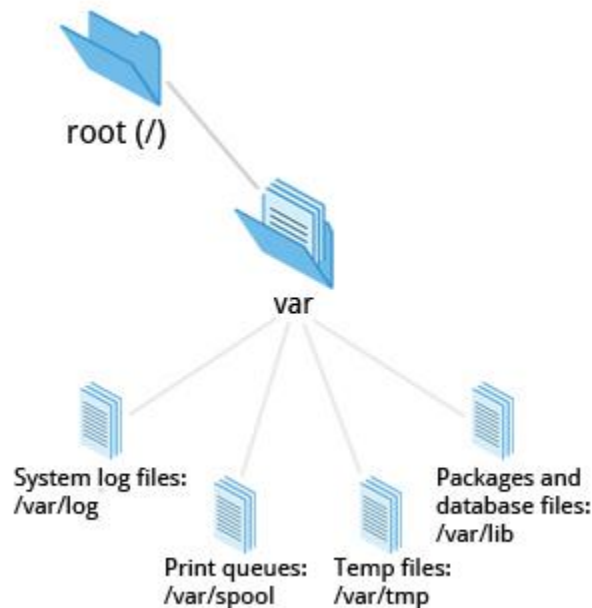
2. **/dev/lp1** (second printer)
3. **/dev/random** (a source of random numbers).

## The /var directory

The **/var** directory contains files that are expected to change in size and content as the system is running (var stands for variable), such as the entries in the following directories:

- System log files: **/var/log**
- Packages and database files: **/var/lib**
- Print queues: **/var/spool**
- Temporary files: **/var/tmp**.

The **/var** directory may be put on its own filesystem so that growth of the files can be accommodated and any exploding file sizes do not fatally affect the system. Network services directories such as **/var/ftp** (the FTP service) and **/var/www** (the HTTP web service) are also found under **/var**.





## The **/etc** directory

The **/etc** directory is the home for system configuration files. It contains no binary programs, although there are some executable scripts. For example, **/etc/resolv.conf** tells the system where to go on the network to obtain host name to IP address mappings (DNS). Files like **passwd**, **shadow** and **group** for managing user accounts are found in the **/etc** directory. While some distributions have historically had their own extensive infrastructure under **/etc** (for example, Red Hat and SUSE have used **/etc/sysconfig**), with the advent of **systemd** there is much more uniformity among distributions today.

Note that **/etc** is for system-wide configuration files and only the superuser can modify files there. User-specific configuration files are always found under their home directory.

## The **/boot** directory

The **/boot** directory contains the few essential files needed to boot the system. For every alternative kernel installed on the system there are four files:

1. **vmlinuz**  
The compressed Linux kernel, required for booting.
2. **initramfs**  
The initial ram filesystem, required for booting, sometimes called initrd, not initramfs.
3. **config**  
The kernel configuration file, only used for debugging and bookkeeping.
4. **System.map**  
Kernel symbol table, only used for debugging.

Each of these files has a kernel version appended to its name.

The Grand Unified Bootloader (GRUB) files such as **/boot/grub/grub.conf** or **/boot/grub2/grub2.cfg** are also found under the **/boot** directory.

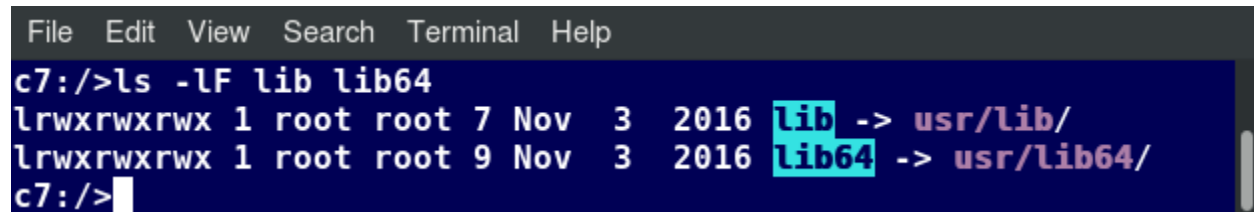
The screenshot shows an example listing of the **/boot** directory, taken from a RHEL 7 system that has multiple installed kernels, including both distribution-supplied and custom-compiled ones. Names will vary and things will tend to look somewhat different on a different distribution.

## The /lib and /lib64 directories

**/lib** contains libraries (common code shared by applications and needed for them to run) for the essential programs in **/bin** and **/sbin**. These library filenames either start with **ld** or **lib**. For example, **/lib/libncurses.so.5.9**.

Most of these are what is known as dynamically loaded libraries (also known as shared libraries or Shared Objects (SO)). On some Linux distributions there exists a **/lib64** directory containing 64-bit libraries, while **/lib** contains 32-bit versions.

On recent Linux distributions, one finds:



```
File Edit View Search Terminal Help
c7: /> ls -lF lib lib64
lrwxrwxrwx 1 root root 7 Nov  3  2016 lib -> usr/lib/
lrwxrwxrwx 1 root root 9 Nov  3  2016 lib64 -> usr/lib64/
c7: />
```

## The /lib and /lib64 Directories

i.e just like for **/bin** and **/sbin**, the directories just point to those under **/usr**.

Kernel modules (kernel code, often device drivers, that can be loaded and unloaded without re-starting the system) are located in **/lib/modules/<kernel-version-number>**.

## Removable media: the /media, /run and /mnt Directories

One often uses removable media, such as USB drives, CDs and DVDs. To make the material accessible through the regular filesystem, it has to be mounted at a convenient location. Most Linux systems are configured so any removable media are automatically mounted when the system notices something has been plugged in.

While historically this was done under the **/media** directory, modern Linux distributions place these mount points under the **/run** directory. For example, a USB pen drive with a label **myusbdrive** for a user name **student** would be mounted at **/run/media/student/myusbdrive**.

The **/mnt** directory has been used since the early days of UNIX for temporarily mounting filesystems. These can be those on removable media, but more often might

be network filesystems , which are not normally mounted. Or these can be temporary partitions, or so-called **loopback** filesystems, which are files which pretend to be partitions.

#### Additional Directories under /:

Directory Name	Usage
<b>/opt</b>	Optional application software packages
<b>/sys</b>	Virtual pseudo-filesystem giving information about the system and the hardware Can be used to alter system parameters and for debugging purposes
<b>/srv</b>	Site-specific data served up by the system Seldom used
<b>/tmp</b>	Temporary files; on some distributions erased across a reboot and/or may actually be a ramdisk in memory
<b>/usr</b>	Multi-user applications, utilities and data

#### The /usr directory tree

The **/usr** directory tree contains theoretically non-essential programs and scripts (in the sense that they should not be needed to initially boot the system) and has at least the following sub-directories:

Directory Name	Usage
<code>/usr/include</code>	Header files used to compile applications
<code>/usr/lib</code>	Libraries for programs in <code>/usr/bin</code> and <code>/usr/sbin</code>
<code>/usr/lib64</code>	64-bit libraries for 64-bit programs in <code>/usr/bin</code> and <code>/usr/sbin</code>
<code>/usr/sbin</code>	Non-essential system binaries, such as system daemons
<code>/usr/share</code>	Shared data used by applications, generally architecture-independent
<code>/usr/src</code>	Source code, usually for the Linux kernel
<code>/usr/local</code>	Data and programs specific to the local machine. Subdirectories include <code>bin</code> , <code>sbin</code> , <code>lib</code> , <code>share</code> , <code>include</code> , etc.
<code>/usr/bin</code>	This is the primary directory of executable commands on the system

## Comparing files with diff

Now that you know about the filesystem and its structure, let's learn how to manage files and directories.

**diff** is used to compare files and directories. This often-used utility program has many useful options (see: **man diff**) including:

diff Option	Usage
<code>-c</code>	Provides a listing of differences that include three lines of context before and after the lines differing in content
<code>-r</code>	Used to recursively compare subdirectories, as well as the current directory
<code>-i</code>	Ignore the case of letters
<code>-w</code>	Ignore differences in spaces and tabs (white space)
<code>-q</code>	Be quiet: only report if files are different without listing the differences

To compare two files, at the command prompt, type **diff [options] <filename1> <filename2>**. **diff** is meant to be used for text files; for binary files, one can use **cmp**.

## Using diff3 and patch

You can compare three files at once using **diff3**, which uses one file as the reference basis for the other two. For example, suppose you and a co-worker both have made modifications to the same file working at the same time independently. **diff3** can show

the differences based on the common file you both started with. The syntax for **diff3** is as follows:

```
$ diff3 MY-FILE COMMON-FILE YOUR-FILE
```

The graphic shows the use of **diff3**.

Many modifications to source code and configuration files are distributed utilizing patches, which are applied, not surprisingly, with the **patch** program. A patch file contains the deltas (changes) required to update an older version of a file to the new one. The patch files are actually produced by running **diff** with the correct options, as in:

```
$ diff -Nur originalfile newfile > patchfile
```

Distributing just the patch is more concise and efficient than distributing the entire file. For example, if only one line needs to change in a file that contains 1000 lines, the patch file will be just a few lines long.

To apply a patch, you can just do either of the two methods below:

```
$ patch -p1 < patchfile  
$ patch originalfile patchfile
```

The first usage is more common, as it is often used to apply changes to an entire directory tree, rather than just one file, as in the second example. To understand the use of the **-p1** option and many others, see the man page for **patch**.

## Using the file utility

In Linux, a file's extension often does not categorize it the way it might in other operating systems. One cannot assume that a file named **file.txt** is a text file and not an executable program. In Linux, a filename is generally more meaningful to the user of the system than the system itself. In fact, most applications directly examine a file's contents to see what kind of object it is rather than relying on an extension. This is very different from the way Windows handles filenames, where a filename ending with **.exe**, for example, represents an executable binary file.

The real nature of a file can be ascertained by using the **file** utility. For the file names given as arguments, it examines the contents and certain characteristics to determine whether the files are plain text, shared libraries, executable programs, scripts, or something else.

## Backing up data

There are many ways you can back up data or even your entire system. Basic ways to do so include the use of simple copying with **cp** and use of the more robust **rsync**.

Both can be used to synchronize entire directory trees. However, **rsync** is more efficient, because it checks if the file being copied already exists. If the file exists and there is no change in size or modification time, **rsync** will avoid an unnecessary copy and save time. Furthermore, because **rsync** copies only the parts of files that have actually changed, it can be very fast.

**cp** can only copy files to and from destinations on the local machine (unless you are copying to or from a filesystem mounted using NFS), but **rsync** can also be used to copy files from one machine to another. Locations are designated in the **target:path** form, where **target** can be in the form of **someone@host**. The **someone@** part is optional and used if the remote user is different from the local user.

**rsync** is very efficient when recursively copying one directory tree to another, because only the differences are transmitted over the network. One often synchronizes the destination directory tree with the origin, using the **-r** option to recursively walk down the directory tree copying all files and directories below the one listed as the source.

## Using rsync

**rsync** is a very powerful utility. For example, a very useful way to back up a project directory might be to use the following command:

```
$ rsync -r project-X archive-machine:archives/project-X
```

Note that **rsync** can be very destructive! Accidental misuse can do a lot of harm to data and programs, by inadvertently copying changes to where they are not wanted. Take care to specify the correct options and paths. It is highly recommended that you first test your **rsync** command using the **-dry-run** option to ensure that it provides the results that you want.

To use **rsync** at the command prompt, type **rsync sourcefile destinationfile**, where either file can be on the local machine or on a networked machine; The contents of **sourcefile** will be copied to **destinationfile**.

A good combination of options is shown in:

```
$ rsync --progress -avrxH --delete sourcedir destdir
```

## Compressing Data

File data is often compressed to save disk space and reduce the time it takes to transmit files over networks.

Linux uses a number of methods to perform this compression, including:

Command	Usage
<b>gzip</b>	The most frequently used Linux compression utility
<b>bzip2</b>	Produces files significantly smaller than those produced by <b>gzip</b>
<b>xz</b>	The most space-efficient compression utility used in Linux
<b>zip</b>	Is often required to examine and decompress archives from other operating systems

These techniques vary in the efficiency of the compression (how much space is saved) and in how long they take to compress; generally, the more efficient techniques take longer. Decompression time does not vary as much across different methods.

In addition, the **tar** utility is often used to group files in an archive and then compress the whole archive at once.

### Compressing data using gzip

**gzip** is the most often used Linux compression utility. It compresses very well and is very fast. The following table provides some usage examples:

Command	Usage
<code>gzip *</code>	Compresses all files in the current directory; each file is compressed and renamed with a <code>.gz</code> extension
<code>gzip -r projectX</code>	Compresses all files in the <code>projectX</code> directory, along with all files in all of the directories under <code>projectX</code>
<code>gunzip foo</code>	De-compresses <code>foo</code> found in the file <code>foo.gz</code> . Under the hood, the <code>gunzip</code> command is actually the same as <code>gzip -d</code>

### Compressing data using bzip2

**bzip2** has a syntax that is similar to `gzip` but it uses a different compression algorithm and produces significantly smaller files, at the price of taking a longer time to do its work. Thus, it is more likely to be used to compress larger files.

Examples of common usage are also similar to **gzip**:

Command	Usage
<code>bzip2 *</code>	Compresses all of the files in the current directory and replaces each file with a file renamed with a <code>.bz2</code> extension
<code>bunzip2 *.bz2</code>	Decompresses all of the files with an extension of <code>.bz2</code> in the current directory. Under the hood, <code>bunzip2</code> is the same as calling <code>bzip2 -d</code>

### Compressing data using xz

**xz** is the most space efficient compression utility used in Linux and is used to [store archives of the Linux kernel](#). Once again, it trades a slower compression speed for an even higher compression ratio.



Command	Usage
<code>xz *</code>	Compresses all of the files in the current directory and replaces each file with one with a <code>.xz</code> extension
<code>xz foo</code>	Compresses <code>foo</code> into <code>foo.xz</code> using the default compression level (-6), and removes <code>foo</code> if compression succeeds
<code>xz -dk bar.xz</code>	Decompresses <code>bar.xz</code> into <code>bar</code> and does not remove <code>bar.xz</code> even if decompression is successful
<code>xz -dcf a.txt b.txt.xz &gt; abcd.txt</code>	Decompresses a mix of compressed and uncompressed files to standard output, using a single command
<code>xz -d *.xz</code>	Decompresses the files compressed using <code>xz</code>

Compressed files are stored with a `.xz` extension.

### Handling files using zip

The `zip` program is not often used to compress files in Linux, but is often required to examine and decompress archives from other operating systems. It is only used in Linux when you get a zipped file from a Windows user. It is a legacy program.

Command	Usage
<code>zip backup *</code>	Compresses all files in the current directory and places them in the <code>backup.zip</code>
<code>zip -r backup.zip ~</code>	Archives your login directory (~) and all files and directories under it in <code>backup.zip</code>
<code>unzip backup.zip</code>	Extracts all files in <code>backup.zip</code> and places them in the current directory

### Archiving and compressing data using tar

Historically, **tar** stood for "tape archive" and was used to archive files to a magnetic tape. It allows you to create or extract files from an archive file, often called a **tarball**. At the same time, you can optionally compress while creating the archive, and decompress while extracting its contents.

Here are some examples of the use of **tar**:

Command	Usage
<code>tar xvf mydir.tar</code>	Extract all the files in <b>mydir.tar</b> into the <b>mydir</b> directory
<code>tar zcvf mydir.tar.gz mydir</code>	Create the archive and compress with <b>gzip</b>
<code>tar jcvf mydir.tar.bz2 mydir</code>	Create the archive and compress with <b>bz2</b>
<code>tar Jcvf mydir.tar.xz mydir</code>	Create the archive and compress with <b>xz</b>
<code>tar xvf mydir.tar.gz</code>	Extract all the files in <b>mydir.tar.gz</b> into the <b>mydir</b> directory <b>Note:</b> You do <b>not</b> have to tell <b>tar</b> it is in <b>gzip</b> format

You can separate out the archiving and compression stages, as in:

```
$ tar cvf mydir.tar mydir ; gzip mydir.tar
$ gunzip mydir.tar.gz ; tar xvf mydir.tar
```

but this is slower and wastes space by creating an unneeded intermediary **.tar** file.

### Relative compression times and sizes

To demonstrate the relative efficiency of **gzip**, **bzip2**, and **xz**, the following screenshot shows the results of compressing a purely text file directory tree (the **include** directory from the kernel source) using the three methods.

This shows that as compression factors go up, CPU time does as well (i.e. producing smaller archives takes longer).

## Disk to Disk Copying (dd)

The **dd** program is very useful for making copies of raw disk space. For example, to back up your Master Boot Record (MBR) (the first 512-byte sector on the disk that contains a table describing the partitions on that disk), you might type:

```
dd if=/dev/sda of=sda.mbr bs=512 count=1
```

**WARNING!**

Typing:

```
dd if=/dev/sda of=/dev/sdb
```

to make a copy of one disk onto another, will delete everything that previously existed on the second disk.

An exact copy of the first disk device is created on the second disk device.

**Do not experiment with this command as written above, as it can erase a hard disk!**

Exactly what the name **dd** stands for is an often-argued item. The words data definition is the most popular theory and has roots in early IBM history. Often, people joke that it means disk destroyer and other variants such as delete data!



## Disk-to-Disk Copying (dd)

## Summary

- The filesystem tree starts at what is often called the root directory (or trunk, or **/**).
- The Filesystem Hierarchy Standard (FHS) provides Linux developers and system administrators a standard directory structure for the filesystem.
- Partitions help to segregate files according to usage, ownership, and type.
- Filesystems can be mounted anywhere on the main filesystem tree at a mount point. Automatic filesystem mounting can be set up by editing **/etc/fstab**.
- NFS (Network File System) is a useful method for sharing files and data through the network systems.
- Filesystems like **/proc** are called pseudo filesystems because they exist only in memory.
- **/root** (slash-root) is the home directory for the root user.
- **/var** may be put in its own filesystem so that growth can be contained and not fatally affect the system.
- **/boot** contains the basic files needed to boot the system.
- **patch** is a very useful tool in Linux. Many modifications to source code and configuration files are distributed with patch files, as they contain the deltas or changes to go from an old version of a file to the new version of a file.
- File extensions in Linux do not necessarily mean that a file is of a certain type.
- **cp** is used to copy files on the local machine, while **rsync** can also be used to copy files from one machine to another, as well as synchronize contents.
- **gzip**, **bzip2**, **xz** and **zip** are used to compress files.
- **tar** allows you to create or extract files from an archive file, often called a tarball. You can optionally compress while creating the archive, and decompress while extracting its contents.
- **dd** can be used to make large exact copies, even of entire disk partitions, efficiently.