5.10. Filesystems

5.10.1. What are filesystems?

A *filesystem* is the methods and data structures that an operating system uses to keep track of files on a disk or partition; that is, the way the files are organized on the disk. The word is also used to refer to a partition or disk that is used to store the files or the type of the filesystem. Thus, one might say ``I have two filesystems'' meaning one has two partitions on which one stores files, or that one is using the ``extended filesystem'', meaning the type of the filesystem.

The difference between a disk or partition and the filesystem it contains is important. A few programs (including, reasonably enough, programs that create filesystems) operate directly on the raw sectors of a disk or partition; if there is an existing file system there it will be destroyed or seriously corrupted. Most programs operate on a filesystem, and therefore won't work on a partition that doesn't contain one (or that contains one of the wrong type).

Before a partition or disk can be used as a filesystem, it needs to be initialized, and the bookkeeping data structures need to be written to the disk. This process is called *making a filesystem*.

Most UNIX filesystem types have a similar general structure, although the exact details vary quite a bit. The central concepts are *superblock*, *inode* , *data block*, *directory block* , and *indirection block*. The superblock contains information about the filesystem as a whole, such as its size (the exact information here depends on the filesystem). An inode contains all information about a file, except its name. The name is stored in the directory, together with the number of the inode. A directory entry consists of a filename and the number of the inode which represents the file. The inode contains the numbers of several data blocks, which are used to store the data in the file. There is space only for a few data block numbers in the inode, however, and if more are needed, more space for pointers to the data blocks is allocated dynamically. These dynamically allocated blocks are indirect blocks; the name indicates that in order to find the data block, one has to find its number in the indirect block first.

UNIX filesystems usually allow one to create a *hole* in a file (this is done with the `lseek()` system call; check the manual page), which means that the filesystem just pretends that at a particular place in the file there is just zero bytes, but no actual disk sectors are reserved for that place in the file (this means that the file will use a bit less disk space). This happens especially often for small binaries, Linux shared libraries, some databases, and a few other special cases. (Holes are implemented by storing a special value as the address of the data block in the indirect block or inode. This special address means that no data block is allocated for that part of the file, ergo, there is a hole in the file.)

5.10.2. Filesystems galore

Linux supports several types of filesystems. As of this writing the most important ones are:

minix

> The oldest, presumed to be the most reliable, but quite limited in features (some time stamps are missing, at most 30 character filenames) and restricted in capabilities (at most 64 MB per filesystem).

Dropbox Exhibit 1031 - Page 1
Dropbox, Inc. v. Entangled Media, LLC
IPR2024-00284 - U.S. Patent No. 8,296,338

xia

A modified version of the minix filesystem that lifts the limits on the filenames and filesystem sizes, but does not otherwise introduce new features. It is not very popular, but is reported to work very well.

ext3

The ext3 filesystem has all the features of the ext2 filesystem. The difference is, journaling has been added. This improves performance and recovery time in case of a system crash. This has become more popular than ext2.

ext2

The most featureful of the native Linux filesystems. It is designed to be easily upwards compatible, so that new versions of the filesystem code do not require re-making the existing filesystems.

ext

An older version of ext2 that wasn't upwards compatible. It is hardly ever used in new installations any more, and most people have converted to ext2.

reiserfs

A more robust filesystem. Journaling is used which makes data loss less likely. Journaling is a mechanism whereby a record is kept of transaction which are to be performed, or which have been performed. This allows the filesystem to reconstruct itself fairly easily after damage caused by, for example, improper shutdowns.

jfs

JFS is a journaled filesystem designed by IBM to to work in high performance environments>

xfs

XFS was originally designed by Silicon Graphics to work as a 64-bit journaled filesystem. XFS was also designed to maintain high performance with large files and filesystems.

In addition, support for several foreign filesystems exists, to make it easier to exchange files with other operating systems. These foreign filesystems work just like native ones, except that they may be lacking in some usual UNIX features, or have curious limitations, or other oddities.

msdos

Compatibility with MS-DOS (and OS/2 and Windows NT) FAT filesystems.

umsdos

Extends the msdos filesystem driver under Linux to get long filenames, owners, permissions, links, and device files. This allows a normal msdos filesystem to be used as if it were a Linux one, thus removing the need for a separate partition for Linux.

vfat

This is an extension of the FAT filesystem known as FAT32. It supports larger disk sizes than FAT. Most MS Windows disks are vfat.

Dropbox Exhibit 1031 - Page 2
Dropbox, Inc. v. Entangled Media, LLC
IPR2024-00284 - U.S. Patent No. 8,296,338

iso9660

The standard CD-ROM filesystem; the popular Rock Ridge extension to the CD-ROM standard that allows longer file names is supported automatically.

nfs

A networked filesystem that allows sharing a filesystem between many computers to allow easy access to the files from all of them.

smbfs

A networks filesystem which allows sharing of a filesystem with an MS Windows computer. It is compatible with the Windows file sharing protocols.

hpfs

The OS/2 filesystem.

sysv

SystemV/386, Coherent, and Xenix filesystems.

NTFS

The most advanced Microsoft journaled filesystem providing faster file access and stability over previous Microsoft filesystems.

The choice of filesystem to use depends on the situation. If compatibility or other reasons make one of the non-native filesystems necessary, then that one must be used. If one can choose freely, then it is probably wisest to use ext3, since it has all the features of ext2, and is a journaled filesystem. For more information on filesystems, see Section 5.10.6. You can also read the Filesystems HOWTO located at http://www.tldp.org/HOWTO/Filesystems-HOWTO.html

There is also the proc filesystem, usually accessible as the `/proc` directory, which is not really a filesystem at all, even though it looks like one. The proc filesystem makes it easy to access certain kernel data structures, such as the process list (hence the name). It makes these data structures look like a filesystem, and that filesystem can be manipulated with all the usual file tools. For example, to get a listing of all processes one might use the command

```
$ ls -l /proc
total 0
dr-xr-xr-x   4 root     root            0 Jan 31 20:37 1
dr-xr-xr-x   4 liw      users           0 Jan 31 20:37 63
dr-xr-xr-x   4 liw      users           0 Jan 31 20:37 94
dr-xr-xr-x   4 liw      users           0 Jan 31 20:37 95
dr-xr-xr-x   4 root     users           0 Jan 31 20:37 98
dr-xr-xr-x   4 liw      users           0 Jan 31 20:37 99
-r--r--r--   1 root     root            0 Jan 31 20:37 devices
-r--r--r--   1 root     root            0 Jan 31 20:37 dma
-r--r--r--   1 root     root            0 Jan 31 20:37 filesystems
-r--r--r--   1 root     root            0 Jan 31 20:37 interrupts
-r--------   1 root     root      8654848 Jan 31 20:37 kcore
-r--r--r--   1 root     root            0 Jan 31 11:50 kmsg
-r--r--r--   1 root     root            0 Jan 31 20:37 ksyms
-r--r--r--   1 root     root            0 Jan 31 11:51 loadavg
-r--r--r--   1 root     root            0 Jan 31 20:37 meminfo
-r--r--r--   1 root     root            0 Jan 31 20:37 modules
dr-xr-xr-x   2 root     root            0 Jan 31 20:37 net
dr-xr-xr-x   4 root     root            0 Jan 31 20:37 self
```


Dropbox Exhibit 1031 - Page 3
Dropbox, Inc. v. Entangled Media, LLC
IPR2024-00284 - U.S. Patent No. 8,296,338

```
-r--r--r--   1 root     root            0 Jan 31 20:37 stat
-r--r--r--   1 root     root            0 Jan 31 20:37 uptime
-r--r--r--   1 root     root            0 Jan 31 20:37
version
$
```

(There will be a few extra files that don't correspond to processes, though. The above example has been shortened.)

Note that even though it is called a filesystem, no part of the proc filesystem touches any disk. It exists only in the kernel's imagination. Whenever anyone tries to look at any part of the proc filesystem, the kernel makes it look as if the part existed somewhere, even though it doesn't. So, even though there is a multi-megabyte `/proc/kcore` file, it doesn't take any disk space.

5.10.3. Which filesystem should be used?

There is usually little point in using many different filesystems. Currently, ext3 is the most popular filesystem, because it is a journaled filesystem. Currently it is probably the wisest choice. Reiserfs is another popular choice because it to is journaled. Depending on the overhead for bookkeeping structures, speed, (perceived) reliability, compatibility, and various other reasons, it may be advisable to use another file system. This needs to be decided on a case-by-case basis.

A filesystem that uses journaling is also called a journaled filesystem. A journaled filesystem maintains a log, or journal, of what has happened on a filesystem. In the event of a system crash, or if your 2 year old son hits the power button like mine loves to do, a journaled filesystem is designed to use the filesystem's logs to recreate unsaved and lost data. This makes data loss much less likely and will likely become a standard feature in Linux filesystems. However, do not get a false sense of security from this. Like everything else, errors can arise. Always make sure to back up your data in the event of an emergency.

See Section 5.10.6 for more details about the features of the different filesystem types.

5.10.4. Creating a filesystem

Filesystems are created, i.e., initialized, with the **mkfs** command. There is actually a separate program for each filesystem type. **mkfs** is just a front end that runs the appropriate program depending on the desired filesystem type. The type is selected with the `-t fstype` option.

The programs called by **mkfs** have slightly different command line interfaces. The common and most important options are summarized below; see the manual pages for more.

`-t fstype`

Select the type of the filesystem.

`-c`

Search for bad blocks and initialize the bad block list accordingly.

-l filename

Read the initial bad block list from the name file.

There are also many programs written to add specific options when creating a specific filesystem. For example **mkfs.ext3** adds a **-b** option to allow the administrator to specify what block size should be used. Be sure to find out if there is a specific program available for the filesystem type you want to use. For more information on determining what block size to use please see Section 5.10.5.

Dropbox Exhibit 1031 - Page 4
Dropbox, Inc. v. Entangled Media, LLC
IPR2024-00284 - U.S. Patent No. 8,296,338

To create an ext2 filesystem on a floppy, one would give the following commands:

```
$ fdformat -n /dev/fd0H1440
Double-sided, 80 tracks, 18 sec/track. Total capacity
1440 KB.
Formatting ... done
$ badblocks /dev/fd0H1440 1440 $>$
bad-blocks
$ mkfs.ext2 -l bad-blocks
/dev/fd0H1440
mke2fs 0.5a, 5-Apr-94 for EXT2 FS 0.5, 94/03/10
360 inodes, 1440 blocks
72 blocks (5.00%) reserved for the super user
First data block=1
Block size=1024 (log=0)
Fragment size=1024 (log=0)
1 block group
8192 blocks per group, 8192 fragments per group
360 inodes per group

Writing inode tables: done
Writing superblocks and filesystem accounting information:
done
$
```

First, the floppy was formatted (the `-n` option prevents validation, i.e., bad block checking). Then bad blocks were searched with **badblocks**, with the output redirected to a file, `bad-blocks`. Finally, the filesystem was created, with the bad block list initialized by whatever **badblocks** found.

The `-c` option could have been used with **mkfs** instead of **badblocks** and a separate file. The example below does that.

```
$ mkfs.ext2 -c
/dev/fd0H1440
mke2fs 0.5a, 5-Apr-94 for EXT2 FS 0.5, 94/03/10
360 inodes, 1440 blocks
72 blocks (5.00%) reserved for the super user
First data block=1
Block size=1024 (log=0)
Fragment size=1024 (log=0)
1 block group
8192 blocks per group, 8192 fragments per group
360 inodes per group

Checking for bad blocks (read-only test): done
Writing inode tables: done
Writing superblocks and filesystem accounting information:
done
$
```

The `-c` option is more convenient than a separate use of **badblocks**, but **badblocks** is necessary for checking after the filesystem has been created.

The process to prepare filesystems on hard disks or partitions is the same as for floppies, except that the formatting isn't needed.

5.10.5. Filesystem block size

The block size specifies size that the filesystem will use to read and write data. Larger block sizes will help improve disk I/O performance when using large files, such as databases. This happens because the disk can read or write data for a longer period of time before having to search for the next block.

Dropbox Exhibit 1031 - Page 5
Dropbox, Inc. v. Entangled Media, LLC
IPR2024-00284 - U.S. Patent No. 8,296,338

On the downside, if you are going to have a lot of smaller files on that filesystem, like the `/etc`, there the potential for a lot of wasted disk space.

For example, if you set your block size to 4096, or 4K, and you create a file that is 256 bytes in size, it will still consume 4K of space on your harddrive. For one file that may seem trivial, but when your filesystem contains hundreds or thousands of files, this can add up.

Block size can also effect the maximum supported file size on some filesystems. This is because many modern filesystem are limited not by block size or file size, but by the number of blocks. Therefore you would be using a "block size * max # of blocks = max block size" formula.

5.10.6. Filesystem comparison

Table 5-1. Comparing Filesystem Features

FS Name	Year Introduced	Original OS	Max File Size	Max FS Size	Journaling
FAT16	1983	MSDOS V2	4GB	16MB to 8GB	N
FAT32	1997	Windows 95	4GB	8GB to 2TB	N
HPFS	1988	OS/2	4GB	2TB	N
NTFS	1993	Windows NT	16EB	16EB	Y
HFS+	1998	Mac OS	8EB	?	N
UFS2	2002	FreeBSD	512GB to 32PB	1YB	N
ext2	1993	Linux	16GB to 2TB4	2TB to 32TB	N
ext3	1999	Linux	16GB to 2TB4	2TB to 32TB	Y
ReiserFS3	2001	Linux	8TB8	16TB	Y
ReiserFS4	2005	Linux	?	?	Y
XFS	1994	IRIX	9EB	9EB	Y
JFS	?	AIX	8EB	512TB to 4PB	Y
VxFS	1991	SVR4.0	16EB	?	Y
ZFS	2004	Solaris 10	1YB	16EB	N

Legend

Table 5-2. Sizes

Kilobyte - KB	1024 Bytes
Megabyte - MB	1024 KBs
Gigabyte - GB	1024 MBs
Terabyte - TB	1024 GBs
Petabyte - PB	1024 TBs
Exabyte - EB	1024 PBs
Zettabyte - ZB	1024 EBs
Yottabyte - YB	1024 ZBs

It should be noted that Exabytes, Zettabytes, and Yottabytes are rarely encountered, if ever. There is a current estimate that the worlds printed material is equal to 5 Exabytes. Therefore, some of these filesystem limitations

Dropbox Exhibit 1031 - Page 6
Dropbox, Inc. v. Entangled Media, LLC
IPR2024-00284 - U.S. Patent No. 8,296,338

are considered by many as theoretical. However, the filesystem software has been written with these capabilities.

For more detailed information you can visit http://en.wikipedia.org/wiki/Comparison_of_file_systems.

5.10.7. Mounting and unmounting

Before one can use a filesystem, it has to be *mounted*. The operating system then does various bookkeeping things to make sure that everything works. Since all files in UNIX are in a single directory tree, the mount operation will make it look like the contents of the new filesystem are the contents of an existing subdirectory in some already mounted filesystem.

For example, Figure 5-3 shows three separate filesystems, each with their own root directory. When the last two filesystems are mounted below `/home` and `/usr`, respectively, on the first filesystem, we can get a single directory tree, as in Figure 5-4.

Figure 5-3. Three separate filesystems.

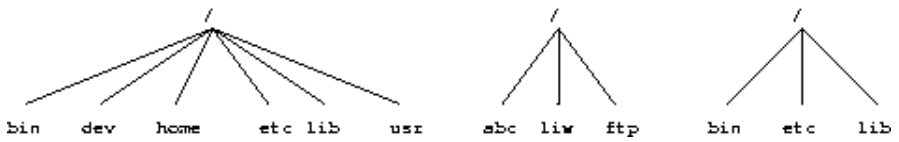
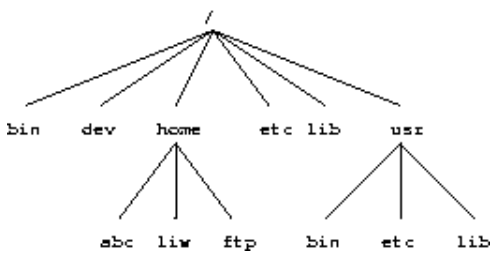


Figure 5-4. `/home` and `/usr` have been mounted.



The mounts could be done as in the following example:

```
$ mount /dev/hda2 /home
$ mount /dev/hda3 /usr
$
```

The **mount** command takes two arguments. The first one is the device file corresponding to the disk or partition containing the filesystem. The second one is the directory below which it will be mounted. After these commands the contents of the two filesystems look just like the contents of the `/home` and `/usr` directories, respectively. One would then say that `/dev/hda2` *is mounted on* `/home`", and similarly for `/usr`. To look at either filesystem, one would look at the contents of the directory on which it has been mounted, just as if it were any other directory. Note the difference between the device file, `/dev/hda2`, and the mounted-on directory, `/home`. The device file gives access to the raw contents of the disk, the mounted-on directory gives access to the files on the disk. The mounted-on directory is called the *mount point*.

Linux supports many filesystem types. **mount** tries to guess the type of the filesystem. You can also use the `-t fstype` option to specify the type directly; this is sometimes necessary, since the heuristics **mount** uses do not always work. For example, to mount an MS-DOS floppy, you could use the following command:

```
        $ mount -t msdos /dev/fd0
        /floppy
        $
```


Dropbox Exhibit 1031 - Page 7
Dropbox, Inc. v. Entangled Media, LLC
IPR2024-00284 - U.S. Patent No. 8,296,338

The mounted-on directory need not be empty, although it must exist. Any files in it, however, will be inaccessible by name while the filesystem is mounted. (Any files that have already been opened will still be accessible. Files that have hard links from other directories can be accessed using those names.) There is no harm done with this, and it can even be useful. For instance, some people like to have `/tmp` and `/var/tmp` synonymous, and make `/tmp` be a symbolic link to `/var/tmp`. When the system is booted, before the `/var` filesystem is mounted, a `/var/tmp` directory residing on the root filesystem is used instead. When `/var` is mounted, it will make the `/var/tmp` directory on the root filesystem inaccessible. If `/var/tmp` didn't exist on the root filesystem, it would be impossible to use temporary files before mounting `/var`.

If you don't intend to write anything to the filesystem, use the `-r` switch for **mount** to do a *read-only mount*. This will make the kernel stop any attempts at writing to the filesystem, and will also stop the kernel from updating file access times in the inodes. Read-only mounts are necessary for unwritable media, e.g., CD-ROMs.

The alert reader has already noticed a slight logistical problem. How is the first filesystem (called the *root filesystem*, because it contains the root directory) mounted, since it obviously can't be mounted on another filesystem? Well, the answer is that it is done by magic. The root filesystem is magically mounted at boot time, and one can rely on it to always be mounted. If the root filesystem can't be mounted, the system does not boot. The name of the filesystem that is magically mounted as root is either compiled into the kernel, or set using LILO or **rdev**.

For more information, see the kernel source or the Kernel Hackers' Guide.

The root filesystem is usually first mounted read-only. The startup scripts will then run **fsck** to verify its validity, and if there are no problems, they will *re-mount* it so that writes will also be allowed. **fsck** must not be run on a mounted filesystem, since any changes to the filesystem while **fsck** is running *will* cause trouble. Since the root filesystem is mounted read-only while it is being checked, **fsck** can fix any problems without worry, since the remount operation will flush any metadata that the filesystem keeps in memory.

On many systems there are other filesystems that should also be mounted automatically at boot time. These are specified in the `/etc/fstab` file; see the fstab man page for details on the format. The details of exactly when the extra filesystems are mounted depend on many factors, and can be configured by each administrator if need be; see Chapter 8.

When a filesystem no longer needs to be mounted, it can be unmounted with **umount**. **umount** takes one argument: either the device file or the mount point. For example, to unmount the directories of the previous example, one could use the commands

```
$ umount /dev/hda2
$ umount /usr
$
```

See the man page for further instructions on how to use the command. It is imperative that you always unmount a mounted floppy. *Don't just pop the floppy out of the drive!* Because of disk caching, the data is not necessarily written to the floppy until you unmount it, so removing the floppy from the drive too early might cause the contents to become garbled. If you only read from the floppy, this is not very likely, but if you write, even accidentally, the result may be catastrophic.

Mounting and unmounting requires super user privileges, i.e., only root can do it. The reason for this is that if any user can mount a floppy on any directory, then it is rather easy to create a floppy with, say, a Trojan horse disguised as `/bin/sh`, or any other often used program. However, it is often necessary to allow users to use floppies, and there are several ways to do this:

- Give the users the root password. This is obviously bad security, but is the easiest solution. It works well if there is no need for security anyway, which is the case on many non-networked, personal systems.

Dropbox Exhibit 1031 - Page 8
Dropbox, Inc. v. Entangled Media, LLC
IPR2024-00284 - U.S. Patent No. 8,296,338

- Use a program such as **sudo** to allow users to use mount. This is still bad security, but doesn't directly give super user privileges to everyone. It requires several seconds of hard thinking on the users' behalf. Furthermore **sudo** can be configured to only allow users to execute certain commands. See the sudo(8), sudoers(5), and visudo(8) manual pages.

- Make the users use **mtools**, a package for manipulating MS-DOS filesystems, without mounting them. This works well if MS-DOS floppies are all that is needed, but is rather awkward otherwise.

- List the floppy devices and their allowable mount points together with the suitable options in `/etc/fstab`.

The last alternative can be implemented by adding a line like the following to the `/etc/fstab` file:

```
/dev/fd0            /floppy      msdos   user,noauto      0     0
```

The columns are: device file to mount, directory to mount on, filesystem type, options, backup frequency (used by **dump**), and **fsck** pass number (to specify the order in which filesystems should be checked upon boot; 0 means no check).

The `noauto` option stops this mount to be done automatically when the system is started (i.e., it stops **mount -a** from mounting it). The `user` option allows any user to mount the filesystem, and, because of security reasons, disallows execution of programs (normal or setuid) and interpretation of device files from the mounted filesystem. After this, any user can mount a floppy with an msdos filesystem with the following command:

```
$ mount /floppy
$
```

The floppy can (and needs to, of course) be unmounted with the corresponding **umount** command.

If you want to provide access to several types of floppies, you need to give several mount points. The settings can be different for each mount point. For example, to give access to both MS-DOS and ext2 floppies, you could have the following to lines in `/etc/fstab`:

```
/dev/fd0    /mnt/dosfloppy    msdos   user,noauto  0  0
/dev/fd0    /mnt/ext2floppy   ext2    user,noauto  0  0
```

The alternative is to just add one line similar to the following:

```
/dev/fd0    /mnt/floppy    auto   user,noauto  0  0
```

The "auto" option in the filesystem type column allows the mount command to query the filesystem and try to determine what type it is itself. This option won't work on all filesystem types, but works fine on the more common ones.

For MS-DOS filesystems (not just floppies), you probably want to restrict access to it by using the `uid`, `gid`, and `umask` filesystem options, described in detail on the **mount** manual page. If you aren't careful, mounting an MS-DOS filesystem gives everyone at least read access to the files in it, which is not a good idea.

5.10.8. Filesystem Security

TO BE ADDED

This section will describe mount options and how to use them in `/etc/fstab` to provide additional system security.

5.10.9. Checking filesystem integrity with fsck

Dropbox Exhibit 1031 - Page 9
Dropbox, Inc. v. Entangled Media, LLC
IPR2024-00284 - U.S. Patent No. 8,296,338

Filesystems are complex creatures, and as such, they tend to be somewhat error-prone. A filesystem's correctness and validity can be checked using the **fsck** command. It can be instructed to repair any minor problems it finds, and to alert the user if there any unrepairable problems. Fortunately, the code to implement filesystems is debugged quite effectively, so there are seldom any problems at all, and they are usually caused by power failures, failing hardware, or operator errors; for example, by not shutting down the system properly.

Most systems are setup to run **fsck** automatically at boot time, so that any errors are detected (and hopefully corrected) before the system is used. Use of a corrupted filesystem tends to make things worse: if the data structures are messed up, using the filesystem will probably mess them up even more, resulting in more data loss. However, **fsck** can take a while to run on big filesystems, and since errors almost never occur if the system has been shut down properly, a couple of tricks are used to avoid doing the checks in such cases. The first is that if the file `/etc/fastboot` exists, no checks are made. The second is that the ext2 filesystem has a special marker in its superblock that tells whether the filesystem was unmounted properly after the previous mount. This allows **e2fsck** (the version of **fsck** for the ext2 filesystem) to avoid checking the filesystem if the flag indicates that the unmount was done (the assumption being that a proper unmount indicates no problems). Whether the `/etc/fastboot` trick works on your system depends on your startup scripts, but the ext2 trick works every time you use **e2fsck**. It has to be explicitly bypassed with an option to **e2fsck** to be avoided. (See the **e2fsck** man page for details on how.)

The automatic checking only works for the filesystems that are mounted automatically at boot time. Use **fsck** manually to check other filesystems, e.g., floppies.

If **fsck** finds unrepairable problems, you need either in-depth knowledge of how filesystems work in general, and the type of the corrupt filesystem in particular, or good backups. The latter is easy (although sometimes tedious) to arrange, the former can sometimes be arranged via a friend, the Linux newsgroups and mailing lists, or some other source of support, if you don't have the know-how yourself. I'd like to tell you more about it, but my lack of education and experience in this regard hinders me. The **debugfs** program by Theodore Ts'o should be useful.

fsck must only be run on unmounted filesystems, never on mounted filesystems (with the exception of the read-only root during startup). This is because it accesses the raw disk, and can therefore modify the filesystem without the operating system realizing it. There *will* be trouble, if the operating system is confused.

5.10.10. Checking for disk errors with badblocks

It can be a good idea to periodically check for bad blocks. This is done with the **badblocks** command. It outputs a list of the numbers of all bad blocks it can find. This list can be fed to **fsck** to be recorded in the filesystem data structures so that the operating system won't try to use the bad blocks for storing data. The following example will show how this could be done.

```
$ badblocks /dev/fd0H1440 1440 >
bad-blocks
$ fsck -t ext2 -l bad-blocks
/dev/fd0H1440
Parallelizing fsck version 0.5a (5-Apr-94)
e2fsck 0.5a, 5-Apr-94 for EXT2 FS 0.5, 94/03/10
Pass 1: Checking inodes, blocks, and sizes
Pass 2: Checking directory structure
Pass 3: Checking directory connectivity
Pass 4: Check reference counts.
Pass 5: Checking group summary information.

/dev/fd0H1440: ***** FILE SYSTEM WAS MODIFIED *****
/dev/fd0H1440: 11/360 files, 63/1440 blocks
$
```

If badblocks reports a block that was already used, **e2fsck** will try to move the block to another place. If the block was really bad, not just marginal, the contents of the file may be corrupted.

Dropbox Exhibit 1031 - Page 10
Dropbox, Inc. v. Entangled Media, LLC
IPR2024-00284 - U.S. Patent No. 8,296,338

5.10.11. Fighting fragmentation?

When a file is written to disk, it can't always be written in consecutive blocks. A file that is not stored in consecutive blocks is *fragmented*. It takes longer to read a fragmented file, since the disk's read-write head will have to move more. It is desirable to avoid fragmentation, although it is less of a problem in a system with a good buffer cache with read-ahead.

Modern Linux filesystem keep fragmentation at a minimum by keeping all blocks in a file close together, even if they can't be stored in consecutive sectors. Some filesystems, like ext3, effectively allocate the free block that is nearest to other blocks in a file. Therefore it is not necessary to worry about fragmentation in a Linux system.

In the earlier days of the ext2 filesystem, there was a concern over file fragmentation that lead to the development of a defragmentation program called, defrag. A copy of it can still be downloaded at http://www.go.dlr.de/linux/src/defrag-0.73.tar.gz. However, it is HIGHLY recommended that you NOT use it. It was designed for and older version of ext2, and has not bee updated since 1998! I only mention it here for references purposes.

There are many MS-DOS defragmentation programs that move blocks around in the filesystem to remove fragmentation. For other filesystems, defragmentation must be done by backing up the filesystem, re-creating it, and restoring the files from backups. Backing up a filesystem before defragmenting is a good idea for all filesystems, since many things can go wrong during the defragmentation.

5.10.12. Other tools for all filesystems

Some other tools are also useful for managing filesystems. **df** shows the free disk space on one or more filesystems; **du** shows how much disk space a directory and all its files contain. These can be used to hunt down disk space wasters. Both have manual pages which detail the (many) options which can be used.

sync forces all unwritten blocks in the buffer cache (see Section 6.6) to be written to disk. It is seldom necessary to do this by hand; the daemon process **update** does this automatically. It can be useful in catastrophes, for example if **update** or its helper process **bdflush** dies, or if you must turn off power *now* and can't wait for **update** to run. Again, there are manual pages. The **man** is your very best friend in Linux. Its cousin **apropos** is also very useful when you don't know what the name of the command you want is.

5.10.13. Other tools for the ext2/ext3 filesystem

In addition to the filesystem creator (**mke2fs**) and checker (**e2fsck**) accessible directly or via the filesystem type independent front ends, the ext2 filesystem has some additional tools that can be useful.

tune2fs adjusts filesystem parameters. Some of the more interesting parameters are:

- A maximal mount count. **e2fsck** enforces a check when filesystem has been mounted too many times, even if the clean flag is set. For a system that is used for developing or testing the system, it might be a good idea to reduce this limit.
- A maximal time between checks. **e2fsck** can also enforce a maximal time between two checks, even if the clean flag is set, and the filesystem hasn't been mounted very often. This can be disabled, however.
- Number of blocks reserved for root. Ext2 reserves some blocks for root so that if the filesystem fills up, it is still possible to do system administration without having to delete anything. The reserved amount is by default 5 percent, which on most disks isn't enough to be wasteful. However, for floppies there is no point in reserving any blocks.

Dropbox Exhibit 1031 - Page 11
Dropbox, Inc. v. Entangled Media, LLC
IPR2024-00284 - U.S. Patent No. 8,296,338

See the **tune2fs** manual page for more information.

dumpe2fs shows information about an ext2 or ext3 filesystem, mostly from the superblock. Below is a sample output. Some of the information in the output is technical and requires understanding of how the filesystem works, but much of it is readily understandable even for lay-admins.

```
# dumpe2fs
dumpe2fs 1.32 (09-Nov-2002)
Filesystem volume name:   /
Last mounted on:          not available
Filesystem UUID:          51603f82-68f3-4ae7-a755-b777ff9dc739
Filesystem magic number:  0xEF53
Filesystem revision #:    1 (dynamic)
Filesystem features:      has_journal filetype needs_recovery sparse_super
Default mount options:    (none)
Filesystem state:         clean
Errors behavior:          Continue
Filesystem OS type:       Linux
Inode count:              3482976
Block count:              6960153
Reserved block count:     348007
Free blocks:              3873525
Free inodes:              3136573
First block:              0
Block size:               4096
Fragment size:            4096
Blocks per group:         32768
Fragments per group:      32768
Inodes per group:         16352
Inode blocks per group:   511
Filesystem created:       Tue Aug 26 08:11:55 2003
Last mount time:          Mon Dec 22 08:23:12 2003
Last write time:          Mon Dec 22 08:23:12 2003
Mount count:              3
Maximum mount count:      -1
Last checked:             Mon Nov  3 11:27:38 2003
Check interval:           0 (none)
Reserved blocks uid:      0 (user root)
Reserved blocks gid:      0 (group root)
First inode:              11
Inode size:               128
Journal UUID:             none
Journal inode:            8
Journal device:           0x0000
First orphan inode:       655612


Group 0: (Blocks 0-32767)
  Primary superblock at 0, Group descriptors at 1-2
  Block bitmap at 3 (+3), Inode bitmap at 4 (+4)
  Block bitmap at 3 (+3), Inode bitmap at 4 (+4)
  Inode table at 5-515 (+5)
  3734 free blocks, 16338 free inodes, 2 directories
```

debugfs is a filesystem debugger. It allows direct access to the filesystem data structures stored on disk and can thus be used to repair a disk that is so broken that **fsck** can't fix it automatically. It has also been known to be used to recover deleted files. However, **debugfs** very much requires that you understand what you're doing; a failure to understand can destroy all your data.

dump and **restore** can be used to back up an ext2 filesystem. They are ext2 specific versions of the traditional UNIX backup tools. See Section 12.1 for more information on backups.

Dropbox Exhibit 1031 - Page 12
Dropbox, Inc. v. Entangled Media, LLC
IPR2024-00284 - U.S. Patent No. 8,296,338

Prev Home Next
Partitions Up Disks without filesystems

Dropbox Exhibit 1031 - Page 13
Dropbox, Inc. v. Entangled Media, LLC
IPR2024-00284 - U.S. Patent No. 8,296,338