This time we'll talk about filesystems. We'll start out by looking at disk partitions, which are the traditional places to put filesystems. Then we'll take a look at “logical volumes”, which are an abstraction that moves us away from physical disk partitions.

Part 1: Partitions

A partition is just a section of a hard disk. We'll look at why we'd want to chop up a hard disk into partitions, but we'll start by looking at the structure of a hard disk.

Spinning Disk Geometry:

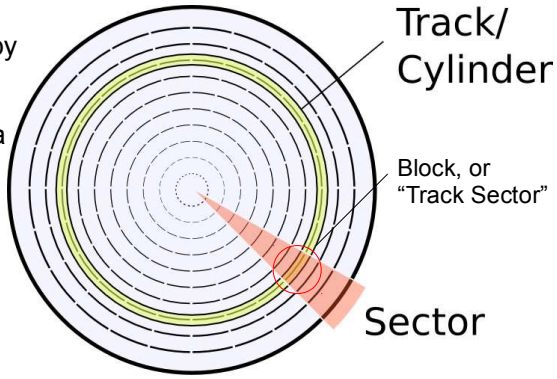
Disks are made of stacks of spinning platters, each surface of which is read by an independent "read head".

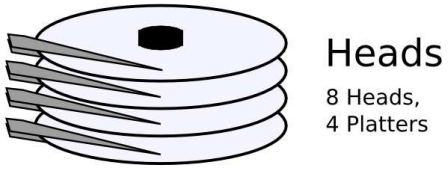
Originally, the position of a piece of data on a disk was given by the coordinates C,H and S, for "Cylinder", "Head" and "Sector".

The intersection of a cylinder with a platter surface is a "Track".

The intersection of a sector with a track is a "Block". Confusingly, the terms "Track Sector" or just "Sector" are also often used to refer to blocks.

Today, the CHS coordinates don't really refer to where the data is actually located on the disk. They're just abstractions. A more recent coordinate scheme, "Logical Block Addressing" (LBA) just numbers the blocks on the disk, starting with zero.





Each block is typically 512 bytes.

The CHS coordinate system began with floppy disks, where the (c,h,s) values really told you where to find the data. Some reasons CHS doesn't really tell you where the data is on a modern hard disk:

- As disks became smarter, they began transparently hiding bad blocks and substituting good blocks from a pool of spares.
- Modern disks actually increase the number of blocks per track for the outer cylinders, since there's more space in those tracks.
- These disks also try to optimize I/O performance, so they want to choose where to really put the data.
- You can have arrays of disks (e.g. RAID) that appear (to the operating system) to be one disk.
- The same addressing scheme can be applied to non-spinning devices, like solid-state disks.

If (c,h,s) is hard to grasp, realize that it's just equivalent to (r,z,θ). They're coordinates in a cylindrical coordinate system.

Partitions:

Sometimes, it's useful to split up a disk into smaller pieces, called “partitions”. Some motivations for this are:

• The operating system may not be able to use storage devices as large as the whole disk.

• You may want to install multiple operating systems.

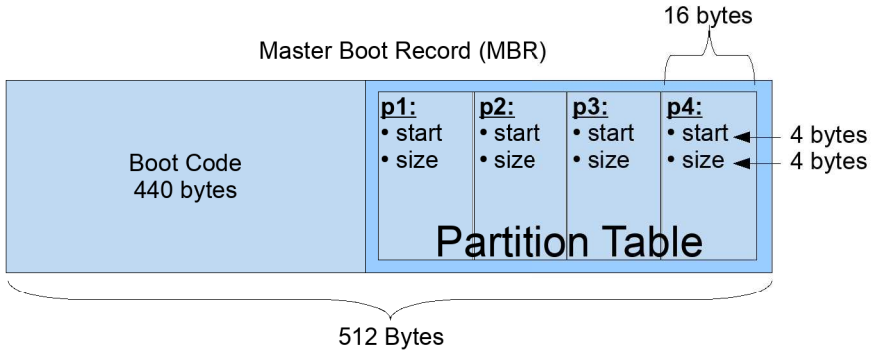
• You may want to designate one partition as swap space.

• You may want to prevent one part of your storage from filling up the whole disk.

One potential problem with having multiple partitions on a disk is that partitions are generally difficult to re-size after they are created.

MBR Partition Table:

The first block on a disk is the “Master Boot Record” (MBR). This block contains a boot program, used to boot an operating system, and a “partition table”. The partition table contains slots for up to four primary partitions. For each partition, the table specifies the starting position of the partition (in LBA coordinates) and the size of the partition (in blocks). Since the size is stored as a 4-byte value, the size of a partition is limited to about 2 Terabytes.



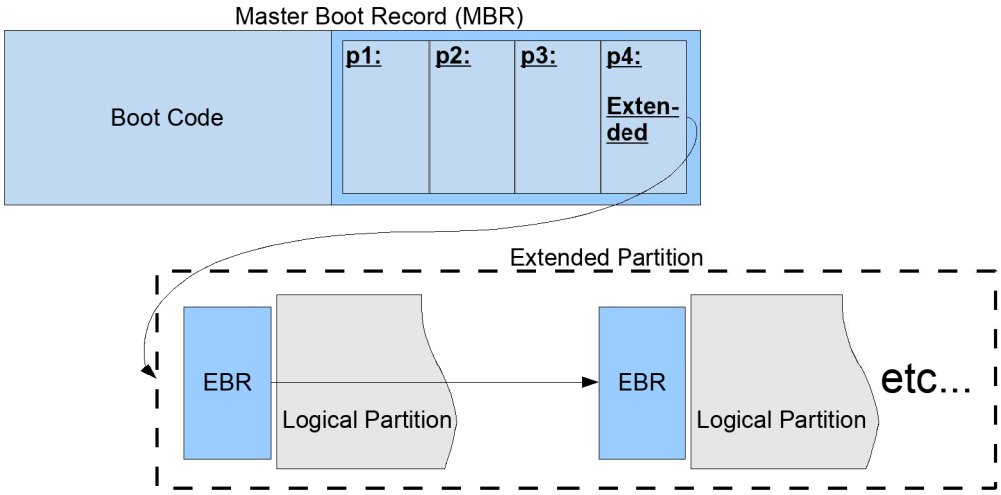
Each disk will have at least one partition. Note that you can only have up to four primary partitions. We'll talk about how to get around the 2 Terabyte size limit later.

The MBR also contains a few other values, like a disk signature, but you can see by adding up the numbers that the boot code and the parition table make up the bulk of the MBR.

In LBA coordinates, the MBR is LBA=0.

Extended Partitions:

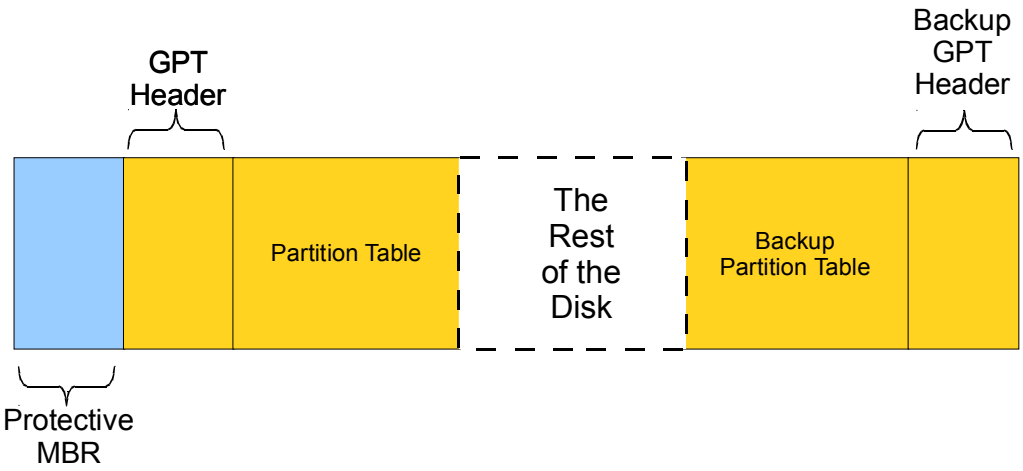
Any one (but only one) of the primary partitions can be an "extended partition". At the start of an extended partition is an "Extended Boot Record" (EBR). This describes a "logical partition" within the extended partition. If there's more than one logical partition, the EBR will also point to the next logical partition in the chain. There's no fixed limit on the number of logical partitions.



In practice, you seldom see disks with more than half a dozen partitions. These days, the typical desktop Linux computer's disk has only two or three.

GUID Partition Tables:

For disks larger than 2 TB, we need to use a different kind of partition table. A “GUID Partition Table” (GPT) can accommodate much larger disks. GPT uses 8 bytes to store addresses, so the maximum theoretical size of partitions is about 9.4 Zettabytes (9.4 billion TB). That should hold us for the near future.



The successor to MBR partition tables are GUID partition tables. They have no fixed number of partitions, and can accommodate much larger disks.

At the front of the disk there's still an MBR. In this case, the MBR's own partition table just says there's one big partition on the disk, of type "GPT", causing non-GPT-aware operating systems to ignore it.

The GPT header has information about the size of each entry in the parition table, and how many entries there are (128 is a typical default value).

Notice that the GPT header and the partition table are duplicated at the end of the disk.

UEFI and GUID Partition Tables:

The successor to the PC BIOS is called “Unified Extensible Firmware Interface” (UEFI). Rather than using boot code stored in the MBR, UEFI uses bootloaders stored in a special partition on the disk.

MS Windows and Apple OS X require UEFI when using GPT partition tables, but Linux allows you to use either UEFI or the traditional MBR boot code with GPT partitions.

Notice that the terms "GUID", "GUID parition table", and "UEFI" are often associated with each other, but they're really independent things.

GUID (aka UUID) is just a scheme for creating unique identifiers without needing some central registry. It can be used for anything, and isn't limited to partition tables.

GUID partition tables use GUIDs to identify partitions, but they'd work about the same if some other ID were used. The magic of GPT is its ability to work with large disks and many partitions.

UEFI was developed with GPT in mind, but you don't need to use UEFI to use GPT.

Disk and Partition Files in /dev:

In Linux, each whole disk drive or partition is represented by a special file in the /dev directory. Programs manipulate the disks and partitions by using these special files. The files have different names, depending on the type of disk.

- **SATA, SCSI, USB (or other external) Disks:**
  These disks are represented by files named /dev/sd[a-z]. They're named in the order they're detected at boot time. Partitions have names like “sda1”, “sda2”, etc.

  Non-logical partitions on each disk are numbered sequentially, 1 through 4. Logical partitions are numbered beginning with 5.

- **NVME Disks:**
  These disks are represented by files named /dev/nvme[0-9]n[1-9]. The first number identifies the device controller, and the second identifies the device. Usually the first NVME disk will be “/dev/nvme0n1”. Partitions will have names like “nvme0n1p1”, “nvme0n1p2”, etc.

Note that you're unlikely to encounter an IDE/PATA disk these days. If you do, these disks will have names like /dev/hda, hdb, etc.

Part 2: Manipulating Partitions

Now we'll look at how to create and otherwise manipulate partitions on a disk. This can be dangerous work. You always need to be careful about which disk you're working on. I've tried to indicate clearly which commands require special caution.

Viewing MBR Partitions with “fdisk”:

You can use the “fdisk -l” or “gdisk -l” (for GPT) commands to view the partition layout on a disk:

```
[root@demo ~]# fdisk -l /dev/sda

Disk /dev/sda: 160.0 GB, 160000000000 bytes
255 heads, 63 sectors/track, 19452 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes

   Device Boot      Start         End      Blocks   Id  System
/dev/sda1   *           1          13      104391   83  Linux
/dev/sda2              14       19452   156143767+  8e  Linux LVM
```

Partition Type

Near the top, you can see the number of heads, sectors and cylinders. These may not represent physical reality, but they're the way the disk presents itself to the operating system.

Fdisk reports the size of each partition in 1024-byte “blocks”. The two partitions above are about 100 MB and about 156 GB. The “+” sign on the size of the second partition means that its size isn't an integer number of 1024-byte blocks.

The “start” and “end” values are in units of cylinders, by default. You can use the “-u” switch to cause fdisk to display start and end in terms of 512-byte “track sectors”.

To add to the confusion about terms like “block” and “sector”, fdisk uses a size of 1024 bytes (not 512) when it reports the number of “blocks” in a partition. The Linux kernel uses blocks of this size, and many Linux programs will assume that a “block” is 1024 bytes. Filesystems typically use “blocks” of 1024, 2048 or 4096 bytes.

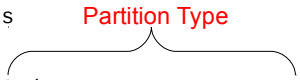
The disk-drive industry is currently pushing new standards that would make the on-disk block size 4096 bytes.

Viewing GPT Partitions with “gdisk”:

```
[root@demo ~]# gdisk -l /dev/sda

Disk /dev/sda: 488397168 sectors, 232.9 GiB
Model: ST9250315AS
Sector size (logical/physical): 512/512 bytes
Disk identifier (GUID): A1080619-34D1-4C04-96D7-97C720D8773A
Partition table holds up to 128 entries
Main partition table begins at sector 2 and ends at sector 33
First usable sector is 34, last usable sector is 488397134
Partitions will be aligned on 2048-sector boundaries
Total free space is 4397 sectors (2.1 MiB)

Number  Start (sector)    End (sector)  Size       Code  Name
   1            2048       476108799   227.0 GiB   8300  Linux filesystem
   5       476110848       488396799   5.9 GiB     8200  Linux swap
```



Here's how to view the layout of a GPT partition table. The tool in this case is "gdisk".

Partition Types:

Here's the list of partition types that fdisk knows about. The most common ones are highlighted.

0	Empty	1e	Hidden W95 FAT1	80	Old Minix	be	Solaris boot
1	FAT12	24	NEC DOS	81	Minix / old Lin	bf	Solaris
2	XENIX root	39	Plan 9	82	Linux swap / So	c1	DRDOS/sec (FAT-
3	XENIX usr	3c	PartitionMagic	83	Linux	c4	DRDOS/sec (FAT-
4	FAT16 <32M	40	Venix 80286	84	OS/2 hidden C:	c6	DRDOS/sec (FAT-
5	Extended	41	PPC PReP Boot	85	Linux extended	c7	Syrinx
6	FAT16	42	SFS	86	NTFS volume set	da	Non-FS data
7	HPFS/NTFS	4d	QNX4.x	87	NTFS volume set	db	CP/M / CTOS / .
8	AIX	4e	QNX4.x 2nd part	88	Linux plaintext	de	Dell Utility
9	AIX bootable	4f	QNX4.x 3rd part	8e	Linux LVM	df	BootIt
a	OS/2 Boot Manag	50	OnTrack DM	93	Amoeba	e1	DOS access
b	W95 FAT32	51	OnTrack DM6 Aux	94	Amoeba BBT	e3	DOS R/O
c	W95 FAT32 (LBA)	52	CP/M	9f	BSD/OS	e4	SpeedStor
e	W95 FAT16 (LBA)	53	OnTrack DM6 Aux	a0	IBM Thinkpad hi	eb	BeOS fs
f	W95 Ext'd (LBA)	54	OnTrackDM6	a5	FreeBSD	ee	EFI GPT
10	OPUS	55	EZ-Drive	a6	OpenBSD	ef	EFI (FAT-12/16/
11	Hidden FAT12	56	Golden Bow	a7	NeXTSTEP	f0	Linux/PA-RISC b
12	Compaq diagnost	5c	Priam Edisk	a8	Darwin UFS	f1	SpeedStor
14	Hidden FAT16 <3	61	SpeedStor	a9	NetBSD	f4	SpeedStor
16	Hidden FAT16	63	GNU HURD or Sys	ab	Darwin boot	f2	DOS secondary
17	Hidden HPFS/NTF	64	Novell Netware	b7	BSDI fs	fd	Linux raid auto
18	AST SmartSleep	65	Novell Netware	b8	BSDI swap	fe	LANstep
1b	Hidden W95 FAT3	70	DiskSecure Mult	bb	Boot Wizard hid	ff	BBT
1c	Hidden W95 FAT3	75	PC/IX				

Note that the partition type is just a label. You can put anything you want into a partition of any type. The partition type designation just provides the operating system with clues about what to expect.

GPT incorporates and extends this list by using 4-digit identifiers. In a GUID partition table, "8300" identifies a Linux partition, and "8e00" identifies a Linux LVM partition, for example.

Creating Partitions with “fdisk”:

You can use fdisk to create or delete partitions on a disk. If you type “fdisk /dev/sda”, for example, you'll be dropped into fdisk's command-line environment, where several simple one-character commands allow you to manipulate partitions on the disk.

Some fdisk commands:

- **p** Print the partition table
- **n** Create a new partition
- **d** Delete a partition
- **t** Change a partition's type
- **q** Quit without saving changes
- **w** Write the new partition table and exit

Note: In fdisk, the term “primary” partition means one that's not an “extended” partition.

```
[root@demo ~]# fdisk /dev/sdb

Command (m for help): n

Command action
   e   extended
   p   primary partition (1-4)
p
Partition number (1-4): 1
First cylinder (1-9726, default 1): +
Using default value 1
Last cylinder or +size or +sizeM or +sizeK (1-9726,
default 9726): +40G

Command (m for help): p
Disk /dev/sdb: 80.0 GB, 80000000000 bytes
255 heads, 63 sectors/track, 9726 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes

   Device Boot      Start  End      Blocks   Id
System
/dev/sdb1               1 4864    39070048+  83  Linux
```

Notice that nothing you do in fdisk is actually written to the disk until you type “w”. If you decide you've made a mistake, you can always quit without saving anything by typing “q”.

I'll only show fdisk here, but usage for gdisk is similar.

Changing a Partition's Type:

Here's how to change a partition's type, using fdisk. In this example, we change the partition from the default type (“Linux”) to mark it as a swap partition.

```
Command (m for help): p

Disk /dev/sdb: 80.0 GB, 80000000000 bytes
255 heads, 63 sectors/track, 9726 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes

   Device Boot      Start         End      Blocks   Id  System
/dev/sdb1               1        4864    39070048+  83  Linux
/dev/sdb2            4865        9726    39054015   83  Linux

Command (m for help): t
Partition number (1-4): 2
Hex code (type L to list codes): 82
Changed system type of partition 2 to 82 (Linux swap / Solaris)

Command (m for help): p

Disk /dev/sdb: 80.0 GB, 80000000000 bytes
255 heads, 63 sectors/track, 9726 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes

   Device Boot      Start         End      Blocks   Id  System
/dev/sdb1               1        4864    39070048+  83  Linux
/dev/sdb2            4865        9726    39054015   82  Linux swap / Solaris
```

Again, gdisk usage is similar.

Formatting a Swap Partition:

Before a swap partition can be used, it needs to be formatted. You can do this with the “mkswap” command:

WARNING!

```
[root@demo ~]#  mkswap /dev/sdb2
Setting up swapspace version 1, size=39054015 kB
```

WARNING!

Note that this will re-format the designated partition immediately, without asking for confirmation, so be careful!

To start using the new swap space immediately, use the “swapon” command:

```
[root@demo ~]# swapon /dev/sdb2
```

As we'll see later, you can also cause this swap partition to be used automatically, at boot time.

Here's one of those very dangerous commands. Please make sure you point mkswap at the right disk partition.

A swap partition is a chunk of disk space that can be used as "fake memory" if your computer runs out of physical memory. This prevents programs from crashing, as they otherwise would. Swap space is much slower than physical memory, though.

Saving Partition Layout with “sfdisk”:

You can save a partition layout into a file, so that it can later be restored. One way to do this is the “sfdisk” command. For example, this command will save the disk partitioning information into the file sdb.out:

```
[root@demo ~]#  sfdisk -d /dev/sdb > sdb.out
```

If the disk is replaced later, or if you have another identical disk that you want to partition in the same way, you can use this command:

WARNING!

```
[root@demo ~]#  sfdisk /dev/sdb < sdb.out
```

WARNING!

Note that this command should be used very carefully, since it will (without asking for confirmation) wipe out any existing partition table on the disk. The content of hda.out looks like this:

```
# partition table of /dev/sdb
unit: sectors

/dev/sdb1 : start=       63, size=   208782, Id=83, bootable
/dev/sdb2 : start=   208845, size=312287535, Id=8e
/dev/sdb3 : start=        0, size=        0, Id= 0
/dev/sdb4 : start=        0, size=        0, Id= 0
```

Another dangerous command.

Saving Partition Layout with “sgdisk”:

For disks with GPT partition tables, you can use “sgdisk” to save or restore partition layouts. For example, this command will save the disk partitioning information into the file sdb.out:

```
[root@demo ~]#  sgdisk -b - /dev/sdb > sdb.out
```

If the disk is replaced later, or if you have another identical disk that you want to partition in the same way, you can use this command:

WARNING!

```
[root@demo ~]#  sgdisk -l - -Gg /dev/sdb< sdb.out
```

WARNING!

Note that this command should be used very carefully, since it will (without asking for confirmation) wipe out any existing partition table on the disk.

This is the GPT equivalent of sfdisk.

Part 3: Filesystem Structure

In order to understand filesystems, it's important to have a little knowledge about how they're laid out on disk. Terms like “superblock” and “block group” show up in error messages sometimes, and knowing what they mean can save you a lot of grief. Here's a primer on filesystem structure.

What is a Filesystem?

A filesystem is a way of organizing data on a block device. The filesystem organizes data into "files", each of which has a name and other metadata attributes. These files are grouped into hierarchical "directories", making it possible to locate a particular file by specifying its name and directory path. Some of the metadata typically associated with each file are:

- **Timestamps**, recording file creation or modification times.
- **Ownership**, specifying a user or group to whom the file belongs.
- **Permissions**, specifying who has access to the file.

Linux originally used the "minix" filesystem, from the operating system of the same name, but quickly switched to what was called the "Extended Filesystem" (in 1992) followed by an improved "Second Extended Filesystem" (in 1993). The two latter filesystems were developed by French software developer Remy Card.

The Second Extended Filesystem (ext2) remained the standard Linux filesystem until the early years of the next century, when it was supplanted by the "Third Extended Filesystem" (ext3), written by Scottish software developer Stephen Tweedie. In 2008 this has been superseded by ext4, developed by Ted Ts'o. Since version 7 of RHEL/CentOS, that distribution has switched to the xfs filesystem, developed by SGI in 1991.

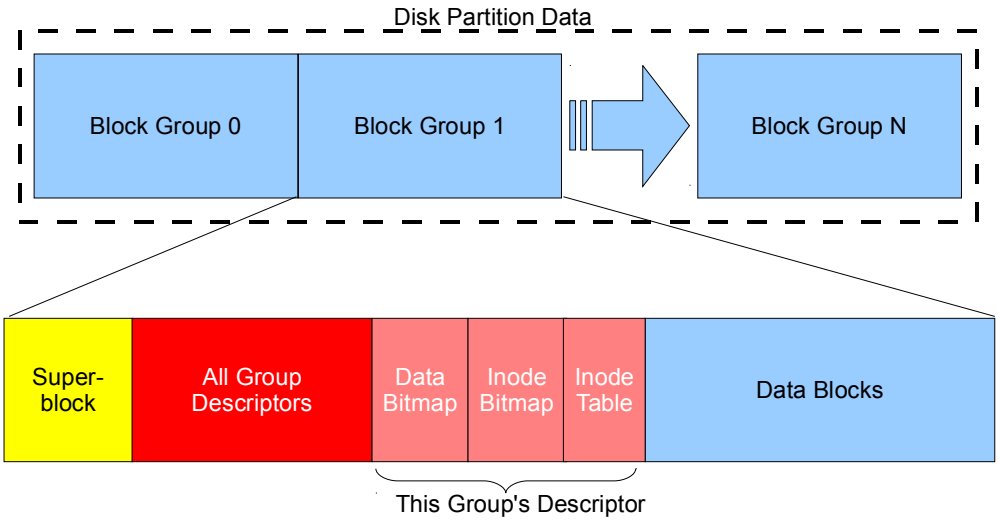
Linux also supports many other filesystems, including Microsoft's VFAT and NTFS, and the ISO9660 filesystem used on CDs and DVDs.

A block device is a device like a disk where you can directly address individual "blocks" of data. Linux separates devices into "character" devices, which just read and write streams of bytes, and "block" devices, in which parts of the device's storage can be directly addressed.

RHEL/CentOS switched to xfs primarily because it gives better performance than ext4 for very large filesystems. See: http://tate.cx/using-the-xfs-file-system/ for some details.

How ext2, ext3 and ext4 Work:

The ext2, ext3 and ext4 filesystems are very similar. Each divides a disk partition into “block groups” of a fixed size. At the beginning of each block group is metadata about the filesystem in general, and that block group in particular. There is much redundancy in this metadata, making it possible to detect and correct damage to the filesystem.

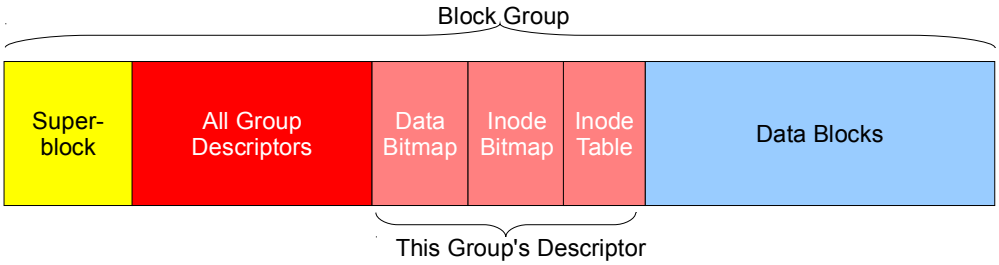


Superblocks:

The ext2/ext3/ext4 filesystem as a whole is described in a chunk of data called the “Superblock”. The superblock contains:

- a name for the filesystem (a “label”),
- the size of the filesystem's block groups,
- timestamps showing when the filesystem was last mounted,
- a flag saying whether it was unmounted cleanly,
- a number showing the amount of unused space in the filesystem,

and much other information. The superblock is duplicated at the beginning of many block groups. Normally, the operating system only uses the copy at the beginning of block group 0, but if this is lost or damaged, the data can be recovered from one of the other copies. During normal operation, the operating system keeps all copies of the superblock synchronized.



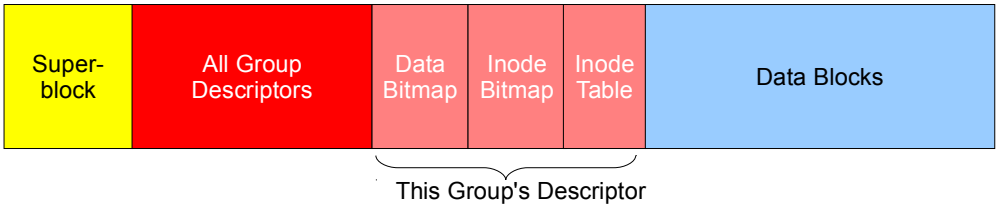
The superblock is actually duplicated at the beginning of each block group for ext2 filesytems. For ext3 and ext4, there's the option of only duplicating it in some block groups. If this option is turned on (as it is by default), the superblock is only duplicated in block groups 0, 1 and powers of 3, 5 and 7.

Inodes and Group Descriptors:

Each file's data is stored in the "data blocks" section of a block group. Files are described by records stored in chunks called "index nodes" (inodes). The inodes are stored in the "inode table" in a part of the block group called the "group descriptor". Data in each inode includes:

- the file's name,
- the file's owner,
- the group to which the file belongs,
- several timestamps,
- permission settings for the file,
- pointers to the data blocks that contain the file's data,

and other information. The group descriptors are so important that copies of the block descriptors for every block group are stored in each block group. Normally, the operating system only uses the descriptors stored in block group 0 for all block groups, but if a filesystem is damaged or has been uncleanly unmounted it's possible to verify the filesystem's integrity and repair damage by using other copies.



As with the superblock, the operating system normally keeps all of the copies of a given group's group descriptors in sync.

Directories are also described by inodes. Each inode has a "type" that identifies it a a file, a directory, or some other special type of thing.

The inodes are numbered sequentially, and files can be identified by their inode number as well as their name.

The "data bitmap" is a set of ones and zeroes, each corresponding to one of the blocks in the block group's data section. If a one is set in the bitmap, that means that this block is used. A zero means that it's free. The data bitmap lets us know which blocks we can use.

Similarly, the "inode bitmap" tells us which entries in the inode table are free.

The Journal:

Although ext2, ext3 and ext4 are very similar, ext3/4 have one important feature that ext2 lacks: journaling. We say that ext3/4 are "journaled" filesystems because, instead of writing data directly into data blocks, the filesystem drivers first write a list of tasks into a journal. These tasks describe any changes that need to be made to the data blocks.

The operating system then periodically looks at the journal to see if there are any tasks that need doing. These tasks are then done, in order, and each completed task is marked as "done" in the journal.

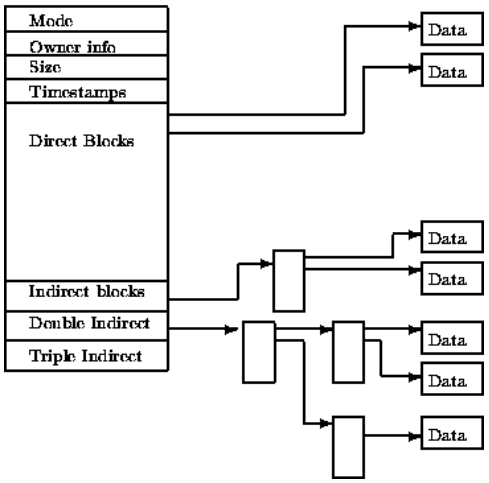
If the computer crashes, the journal is examined at the next reboot to see if there were any outstanding tasks that needed to be done. If so, they're done. Any garbled information left at the end of the journal is ignored and cleared.

Journaling makes it much quicker to check the integrity of a filesytem after a crash, since only a few items in the journal need to be looked at. In contrast, when an ext2 filesystem crashes, the operating system needs to scan the entire filesystem looking for problems.

Other than journaling, ext2 and ext3 are largely the forward- and backward-compatible. An ext2 filesystem can easily be converted to ext3 by adding a journal. Going the other way may be possible, too, if an ext3 filesystem doesn't use any features that aren't present in an ext2 filesystem. Similar considerations apply when going between ext3 and ext4 filesystems.

The journal is described by a special inode, usually inode number 8.

Inode Structure, and Filesystem Limits:



Some size limits for filesystems:

Size Limits	ext2	ext3	ext4
Max. File Size:	2 TB	2 TB	16 TB
Max. Filesystem Size:	16 TB	16 TB	1 EB

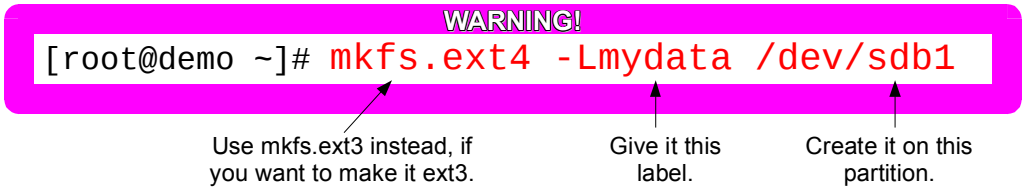
The diagram above shows the structure of a single inode. The file it represents might have data stored in many different data blocks. (And note that the blocks aren't necessarily contiguous.)

Part 4: Filesystem Tools:

Now lets look at some tools for creating and manipulating filesystems.

Making an ext3 or ext4 Filesystem:

To make an ext3 or ext4 filesystem, use one of the “mkfs” commands:



Note that the command above will format (or re-format) the designated partition without asking for any confirmation. Please make sure you point it at the partition you really want to format.

The filesystem label can be any text you choose, but usually the label is chosen to be the same as the name of the location at which you expect to mount the filesystem. For example, a filesystem intended to be mounted at “/boot”, would probably probably be created with “-L/boot”. For the “/” and “/boot” filesystems, this should always be done, but it's good practice for other filesystems, too.

Another dangerous command.

Example mkfs.ext4 Output:

```
[root@demo ~]# mkfs.ext4 -Lmydata /dev/sdb1
mke2fs 1.41.12 (17-May-2010)
Filesystem label=mydata
OS type: Linux
Block size=4096 (log=2)
Fragment size=4096 (log=2)
Stride=0 blocks, Stripe width=0 blocks
121896960 inodes, 487585272 blocks
24379263 blocks (5.00%) reserved for the super user
First data block=0
Maximum filesystem blocks=4294967296
14880 block groups
32768 blocks per group, 32768 fragments per group
8192 inodes per group
Superblock backups stored on blocks:
        32768, 98304, 163840, 229376, 294912, 819200,
884736, 1605632, 2654208,
        4096000, 7962624, 11239424, 20480000, 23887872,
71663616, 78675968,
        102400000, 214990848
Writing inode tables: done.
creating root dir
```

Note that mke2fs divides the disk up into 14,880 block groups, but only (!) 18 copies of the superblock are created. If this were an ext2 filesystem, there would be 14,880 copies. The total number of inodes available (including all inodes in all block groups) is 121,896,960. This is the maximum number of files that this filesystem will hold.

You might notice the reference to “fragments per group” in the output above. In this context, a “fragment” is a chunk of storage space that's smaller than a block. This is seldom used, and in this case you can see that the fragment size is just set to the block size.

Changing the Attributes of a Filesystem:

The tune2fs command can be used to change the attributes of an ext2/ext3/ext4 filesystem after it has been created. For example, to change the filesystem's label:

```
[root@demo ~]# tune2fs -L/data /dev/sdb1
```

Some other useful things that tune2fs can do:

-l List superblock information.

-c Set the maximum mount count for the filesystem, after which a filesystem check will occur (0 = never check).

-i Set the interval between filesystem checks (0 = never check).

Changing a filesystem's label is perfectly safe. It won't cause you to lose any data. (But it might cause confusion if you're already referring to the old label somewhere.) The same is true for the other flags listed above.

Looking at Filesystem Metadata:

"tune2fs -l" will show you a filesystem's superblock information:

```
[root@demo ~]# tune2fs -l /dev/sda1
tune2fs 1.39 (29-May-2006)
Filesystem volume name:   /boot
Filesystem state:         clean
Inode count:              26104
Block count:              104388
Reserved block count:     5219
Free blocks:              55562
Free inodes:              26037
First block:              1
Block size:               1024
Blocks per group:         8192
Inodes per group:         2008
Inode blocks per group:   251
Filesystem created:       Mon Sep 10 10:58:16 2007
Last mount time:          Fri Dec 26 10:23:03 2008
Last write time:          Fri Dec 26 10:23:03 2008
Mount count:              60
Maximum mount count:      -1
Last checked:             Mon Sep 10 10:58:16 2007
Check interval:           0 (<none>)
Reserved blocks uid:      0 (user root)
Reserved blocks gid:      0 (group root)
First inode:              11
Inode size:               128
Journal inode:            8
etc...
```

You can see this plus block group information by using the "dumpe2fs" command.

Note the "mount count", "maximum mount count", "last checked" and "check interval" entries. We'll see later that the "fsck" command uses these.

Checking a Filesystem:

```
[root@demo ~]# fsck /dev/sdb1
```

If a computer loses power unexpectedly, the filesystems on its disks may be left in an untidy state. The "filesystem check" (fsck) command looks at ext2/ext3/ext4 filesystems and tries to find and repair damage. Fsck can only be run on unmounted filesystems.

Each filesystem's superblock contains a flag saying whether the filesystem was cleanly unmounted. If it was, fsck just exits without doing anything further.

If the filesystem wasn't cleanly unmounted, fsck checks it. Under ext3/ext4, fsck first just looks at the journal and completes any outstanding operations, if possible. If this works, then fsck exits.

If the ext3/ext4 journal is damaged, or if this is an ext2 filesystem, fsck scans the filesystem for damage. It does this primarily by looking for inconsistencies between the various copies of the superblock and block group descriptors. If inconsistencies are found, fsck tries to resolve them, using various strategies.

The filesystem's superblock also contains a "mount count", "maximum mount count", "last check date" and "check interval". If the mount count exceeds the maximum, a scan of the filesystem is forced even if it was cleanly unmounted. If the time since the last check date exceeds the check interval, a scan is also forced. Both of these forced checks can be disabled, by using tune2fs.

Modifying fsck's Behavior:

Some useful fsck options:

-f Force a scan, even if the filesystem appears to have been cleanly unmounted.

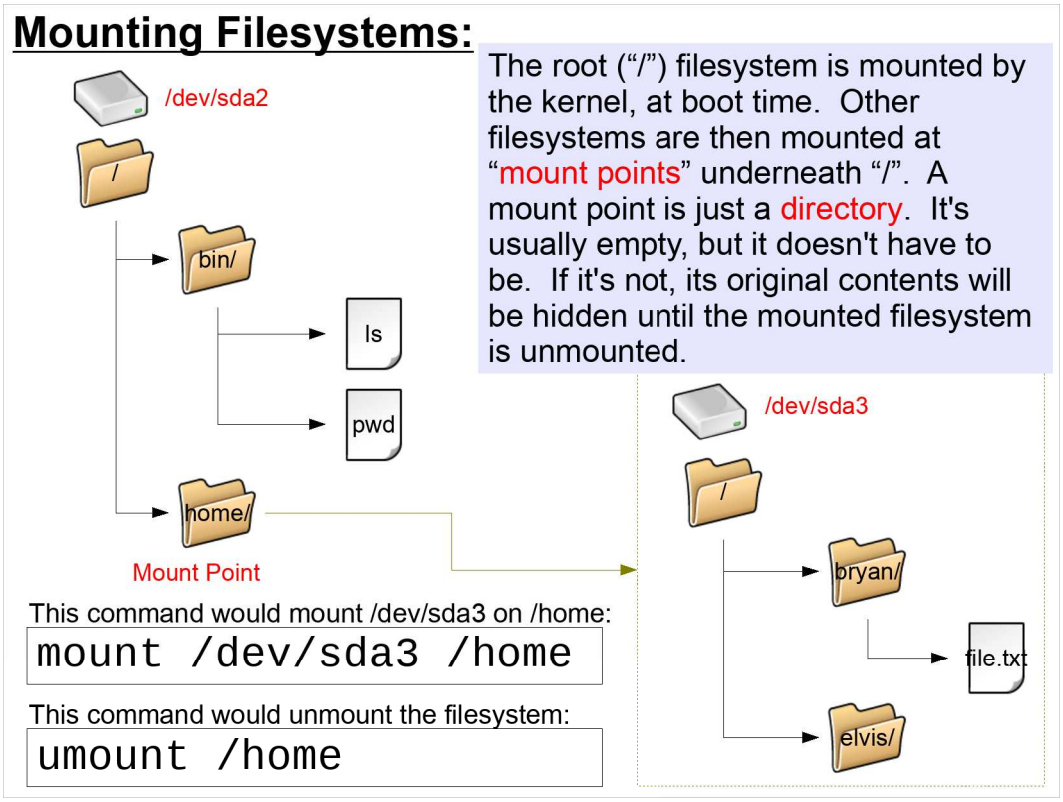
-b Specify an alternative superblock, in case the primary superblock has been damaged.

-y Answer "yes" to any questions fsck asks.

-A Check all filesystems.

-C Show a progress bar as fsck works. (It can sometimes take a very long time.)

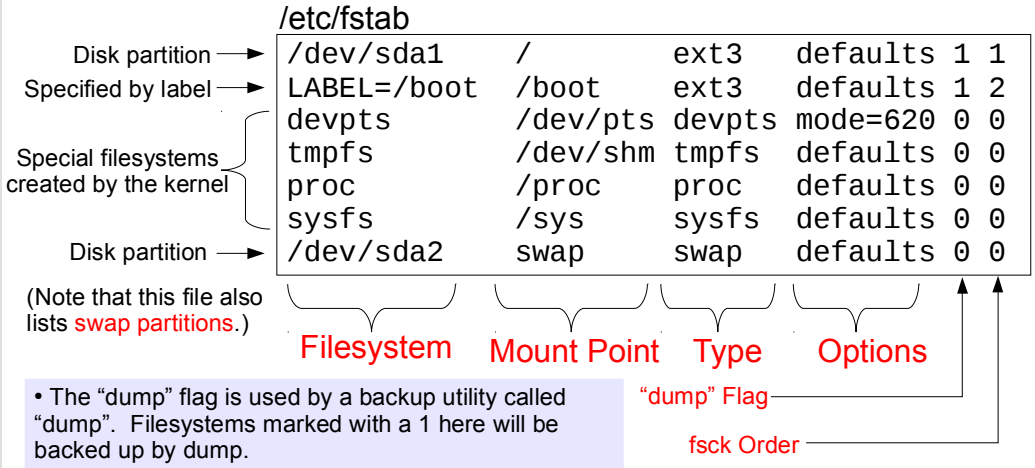
Fsck is actually a wrapper that calls a different type-specific filesystem checker for each different type of filesystem that it knows about.



The directory tree of each physical device is grafted onto the same tree, with the root directory ("/") at the top. There are no "C:" or "D:" drives under Linux. Every file you have access to lives in the same tree, and you don't need to care what device the file lives on.

Mounting Filesystems Automatically at Boot Time:

The file /etc/fstab ("filesystem table") contains a list of filesystems to be mounted automatically at boot time. It looks like this:



- The "dump" flag is used by a backup utility called "dump". Filesystems marked with a 1 here will be backed up by dump.
- The "fsck order" field determines what order filesystems are checked when fsck is run automatically at boot time. A value of zero means that this filesystem won't be checked. Others are checked in ascending order of these values.

Among the "options" settings you can use "noauto" to cause the given filesystem not to be automatically mounted at boot time. In that case, you'd need to manually mount it later, using the "mount" command.

If you have a filesystem listed in /etc/fstab, you can mount it either like this, with two arguments:

```
mount /dev/sda1 /
```

or like this, with one argument:

```
mount /dev/sda1
```

or

```
mount /
```

since /etc/fstab lets "mount" know what you mean by these.

Part 5: Logical Volume Management

RHEL/CentOS/Fedora distributions use Logical Volume Management by default. You'll need to know a little about LVM to understand how these computers' filesystems are laid out.

The LVM System:

The ext2, ext3 and ext4 filesystems are limited by the size of the partitions they occupy. Partitions are difficult to resize, and they can't grow beyond the whole size of the disk. What can we do if we need more space than that for our filesystem?

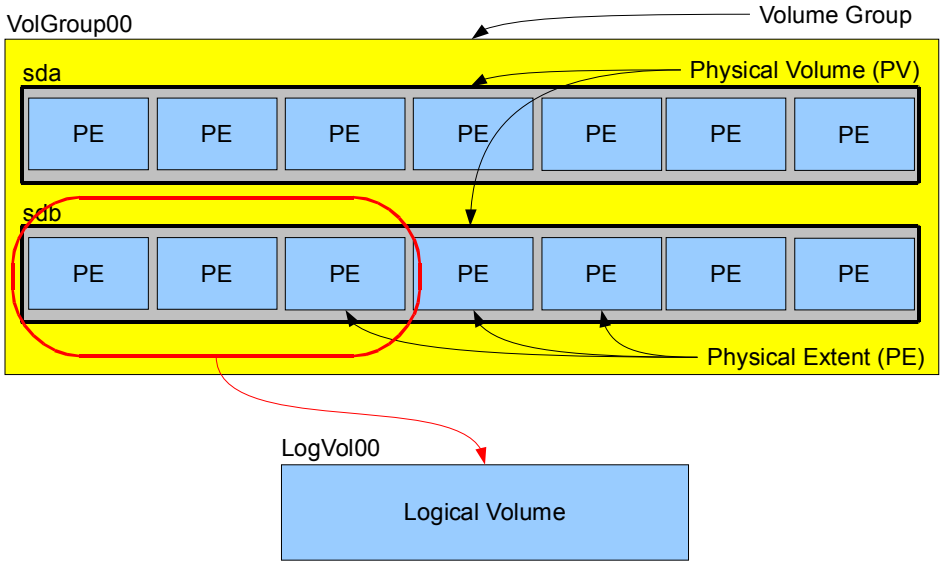
One solution is the Logical Volume Management (LVM) system. LVM lets you define “logical volumes” that can be used like disk partitions. Unlike partitions, logical volumes can span multiple disks, and they can easily grow or shrink.

These days, when you install a Linux distribution on a computer, some of the filesystems that are created will (by default) be on logical volumes, not physical disk partitions. This makes it important to understand how LVM works.

As we'll see, LVM also provides us with another, software-based, way to avoid the 2 TB partition limit imposed by MBR-style partition tables.

Logical Volume Structure:

LVM divides each disk (or "physical volume") into chunks called "physical extents" (PEs). Disks are added to "volume groups" (VGs). Each VG is a pool of physical extents from which "logical volumes" (LVs) can be formed. An LV can be expanded by adding more PEs from the pool. If an LV needs to grow even larger, more PEs can be added to the pool by adding disks to the volume group.



Note that LVM can use either a whole disk or a disk partition as a physical volume.

Creating Logical Volumes:

First, let's make a new disk available to the LVM system by initializing it as an LVM “physical volume” using “pvcreate”:

```
[root@demo ~]# pvcreate /dev/sdb
```

Then, let's create a new volume group and add the newly-initialized disk to it:

```
[root@demo ~]# vgcreate VolGroup01 /dev/sdb
```

Now, let's create a 500 GB logical volume from the pool of space in our new volume group:

```
[root@demo ~]# lvcreate -L500G -nLogVol00 VolGroup01
```

Now we can create a filesystem on the logical volume, just as we'd use a partition:

```
[root@demo ~]# mkfs.ext4 -L/data /dev/VolGroup01/LogVol00
```

Finally, we can mount the logical volume just as we'd mount a partition:

```
[root@demo ~]# mount /dev/VolGroup01/LogVol00 /data
```

Note that you can point pvcreate at either a whole disk, as above, or a disk partition (like “/dev/sdb1”). If you use a whole disk, the disk's partition table is wiped out, since LVM doesn't need it. Thus, LVM can be used to completely avoid the 2 TB limit imposed by MBR-style partition tables.

This is one reason some distributions began using LVM by default. LVM provides a way to support large disks without requiring GUID partition tables.

Examining Volume Groups:

You can find out about a volume group by using the “vgdisplay” command:

```
[root@demo ~]# vgdisplay VolGroup00
  --- Volume group ---
  VG Name               VolGroup00
  System ID
  Format                lvm2
  Metadata Areas        1
  Metadata Sequence No  3
  VG Access             read/write
  VG Status             resizable
  MAX LV                0
  Cur LV                2
  Open LV               2
  Max PV                0
  Cur PV                1
  Act PV                1
  VG Size               148.91 GB
  PE Size               32.00 MB
  Total PE              4765
  Alloc PE / Size       4765 / 148.91 GB
  Free  PE / Size       0 / 0
  VG UUID               blHfoy-z03Z-DzTQ-PH4p-uYfJ-jkHS-29Hxob
```

Notice these. They tell you how many physical extents are in the volume group, and how many are still available for making new logical volumes.

If you move a disk to a different computer that already has a volume group with the same name, you may need to use the UUID of the volume groups to rename one of them. Use “vgrename” for this.

Growing a Logical Volume:

If we don't have any free PEs in our volume group, we can add another disk:

```
[root@demo ~]# vgextend VolGroup01 /dev/sdc
```

Now that we have more PEs, we can assign some of them to one of our existing logical volumes, to make it bigger:

```
[root@demo ~]# lvextend -L+100G /dev/VolGroup01/LogVol00
```

Extending the logical volume doesn't extend the filesystem on top of it. We have to do that by hand. For ext2/ext3/ext4 filesystems, you can use the resize2fs command to do this. The command below will just resize the filesystem so that it occupies all of the available space in the logical volume:

```
[root@demo ~]# resize2fs /dev/VolGroup01/LogVol00
```

For many more “stupid LVM tricks” see: http://www.howtoforge.com/linux_lvm

Thanks!