IBM

developerWorks®

Learn Linux, 101: Hard disk layout

Planning your hard disk partitions

Ian Shields
Senior Programmer
IBM

27 November 2012
(First published 24 February 2010)

Learn how to design a partition layout for disks on a Linux® system. You can use the material in this article to study for the LPI 101 exam for Linux system administrator certification, or just to learn for fun.
27 Nov 2012 - *This article updated to include material for the LPI Exam 101: Objective Changes as of July 2, 2012.*

Major updates include information about Logical Volume Manager, partition table layouts used with traditional MBR disks, and the GUID Partition Table (GPT) and occur in these sections: Logical Volume Manager (LVM), MBR, EBR, GPT and LVM internals, and Resources.

View more content in this series

About this series

This series of articles helps you learn Linux system administration tasks. You can also use the material in these articles to prepare for Linux Professional Institute Certification level 1 (LPIC-1) exams.

See our developerWorks roadmap for LPIC-1 for a description of and link to each article in this series. The roadmap is in progress and reflects the latest (April 2009 with minor updates in July 2012) objectives for the LPIC-1 exams: as we complete articles, we add them to the roadmap. In the meantime, though, you can find earlier versions of similar material, supporting previous LPIC-1 objectives prior to April 2009, in our LPI certification exam prep tutorials.

Overview

In this article, learn to design a disk partitioning layout for a Linux system. Learn to:

- Allocate filesystems and swap space to separate partitions or disks
- Tailor the design to the intended use of the system
- Ensure the system can be booted
- Understand the basic features of Logical Volume Manager (LVM)

© Copyright IBM Corporation 2010, 2012

This article helps you prepare for Objective 102.1 in Topic 102 of the Linux Professional Institute's Junior Level Administration (LPIC-1) exam 101. The objective has a weight of 2.

Note: This article deals mostly with *planning* the layout. For the *implementation* steps, see the articles for Topic 104 (described in our series roadmap).

Note: This article includes material for the LPI Exam 101: Objective Changes as of July 2, 2012. We have added basic information on Logical Volume Manager. We have also added more detail on the partition table layouts used with traditional MBR disks and we have added some basic information on the GUID Partition Table (GPT). The new code listings and figures were all done on a 64-bit Fedora 16 system.

Prerequisites

To get the most from the articles in this series, you should have a basic knowledge of Linux and a working Linux system on which you can practice the commands covered in this article. Sometimes different versions of a program will format output differently, so your results may not always look exactly like the listings and figures shown here.

Filesystem overview

Connect with Ian

Ian is one of our most popular and prolific authors. Browse all of Ian's articles on developerWorks. Check out Ian's profile and connect with him, other authors, and fellow readers in the developerWorks Community.

A Linux filesystem contains *files* that are arranged on a disk or other *block storage device* in *directories*. As with many other systems, directories on a Linux system may contain other directories called *subdirectories*. Unlike a system such as Microsoft® Windows® with a concept of separate file systems on different drive letters (A:, C:, etc.), a Linux filesystem is a single tree with the / directory as its *root* directory.

You might wonder why disk layout is important if the filesystem is just one big tree. Well, what really happens is that each block device, such as a hard drive partition, CD-ROM, or floppy disk, actually has a filesystem on it. You create the single tree view of the filesystem by *mounting* the filesystems on different devices at a point in the tree called a *mount point*.

Develop skills on this topic

This content is part of a progressive knowledge path for advancing your skills. See Basics of Linux system administration: Setting up your system and software

Usually, the kernel starts this mount process by mounting the filesystem on some hard drive partition as /. You may mount other hard drive partitions as /boot, /tmp, or /home. You may mount the filesystem on a floppy drive as /mnt/floppy, and the filesystem on a CD-ROM as /media/cdrom1, for example. You may also mount files from other systems using a networked filesystem such as NFS. There are other types of file mounts, but this gives you an idea of the process. While the mount process actually mounts the *filesystem* on some device, it is common to simply say that you "mount the device," which is understood to mean "mount the filesystem on the device."

Now, suppose you have just mounted the root file system (/) and you want to mount a CD-ROM, /dev/sr0, at the mount point /media/cdrom. The mount point must exist before you mount the CD-ROM over it. When you mount the CD-ROM, the files and subdirectories on the CD-ROM become the files and subdirectories in and below /media/cdrom. Any files or subdirectories that were already in /media/cdrom are no longer visible, although they still exist on the block device that contained the mount point /media/cdrom. If the CD-ROM is unmounted, then the original files and subdirectories become visible again. You should avoid this problem by not placing other files in a directory intended for use as a mount point.

Table 1 shows the shows the directories required in / by the Filesystem Hierarchy Standard (for more detail on FHS, see Resources).

Table 1. FHS directories in /

Directory	Description
bin	Essential command binaries
boot	Static files of the boot loader
dev	Device files
etc	Host-specific system configuration
lib	Essential shared libraries and kernel modules
media	Mount point for removable media
mnt	Mount point for mounting a filesystem temporarily
opt	Add-on application software packages
sbin	Essential system binaries
srv	Data for services provided by this system
tmp	Temporary files
usr	Secondary hierarchy
var	Variable data

Partitions

The first SCSI drive is usually /dev/sda. On an older Linux system, the first IDE hard drive is /dev/hda. With the advent of serially attached (SATA) IDE drives, a mixed PATA/SATA system would sometimes use /dev/hda for the first PATA drive and /devsda for the first SATA drive. On newer systems, all IDE drives are named /dev/sda, /dev/sdb, and so on. The change of name for IDE drives is a result of the *hotplug* system, which initially supported USB drives. Hotplug allows you to plug in new devices and use them immediately, and is now used for all devices whether they are built into the system or plugged later into a running system using USB or Firewire (IEEE 1394) or potentially other types of connection.

Traditionally, a hard drive is formatted into 512 byte *sectors*. All the sectors on a disk platter that can be read without moving the head constitute a *track*. Disks usually have more than one platter. The collection of tracks on the various platters that can be read without moving the head is called a *cylinder*. The *geometry* of a hard drive is expressed in cylinders, tracks (or *heads)* per cylinder, and sectors/track. At the time of this writing, drive manufacturers are starting to introduce disks with

4K sectors. If a filesystem still assumes 512-byte sectors, you may lose performance if a partition does not start at a sector that is on a 4K boundary.

Limitations on the possible sizes for cylinders, heads, and sectors used with DOS operating systems on PC systems resulted in BIOS translating geometry values so that larger hard drives could be supported. Eventually, even these methods were insufficient. More recent developments in disk drive technology have led to *logical block addressing (LBA)*, so the CHS geometry measurements are less important, and the reported geometry on a modern disk bears little or no relation to the real physical sector layout. The larger disks in use today have forced an extension to LBA known as LBA48, which reserves up to 48 bits for sector numbers. A new format called *GUID Partition Table* (or *GPT*) is now being used for larger drives instead of the MBR format. GPT drives support up to 128 partitions by default.

The space on a hard drive is divided (or partitioned) into *partitions*. Partitions cannot overlap; space that is not allocated to a partition is called *free space*. The partitions have names like /dev/hda1, /dev/hda2, /dev/hda3, /dev/sda1, and so on. IDE drives are limited to 63 partitions on systems that do not use hotplug support for IDE drives. SCSI drives, USB drives, and IDE drives supported by hotplug are limited to 15 partitions. A partition is often allocated as an integral number of cylinders (based on the possibly inaccurate notion of a cylinder).

If two different partitioning programs have different understandings of the nominal disk geometry, it is possible for one partitioning program to report an error or possible problem with partitions created by another partitioning program. You may also see this kind of problem if a disk is moved from one system to another, particularly if the BIOS capabilities are different.

You can see the nominal geometry on a Linux system using either the `parted` or `fdisk` tools. Older Linux systems also reported geometry in the /proc filesystem, in a file such as /proc/ide/hda/geometry, a file that might not be present on newer systems. Listing 1 shows how to use the `fdisk` command to display the partitions and geometry of an IDE hard drive with SATA attachment. The `-v` parameter of `fdisk` shows the version. You will need to be root or have root authority via `sudo`, as shown here, to display or manipulate the partition table.

Listing 1. Hard disk geometry

```
ian@attic4:~$ fdisk -v
fdisk (util-linux-ng 2.16)
ian@attic4:~$ sudo fdisk /dev/sdb
[sudo] password for ian:

The number of cylinders for this disk is set to 30401.
There is nothing wrong with that, but this is larger than 1024,
and could in certain setups cause problems with:
1) software that runs at boot time (e.g., old versions of LILO)
2) booting and partitioning software from other OSs
   (e.g., DOS FDISK, OS/2 FDISK)

Command (m for help): p

Disk /dev/sdb: 250.1 GB, 250059350016 bytes
255 heads, 63 sectors/track, 30401 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes
Disk identifier: 0x000404d6
```

```
   Device Boot      Start         End      Blocks   Id  System
/dev/sdb1               1          25      200781   83  Linux
/dev/sdb2              26       12965   103940550   83  Linux
/dev/sdb3           12966       30401   140054670   83  Linux

Command (m for help):
```

In Listing 1, note that `fdisk` prints a warning about the nominal position for the end of cylinder 1024. Cylinder 1024 is important in some older systems where the BIOS is only able to boot partitions that are completely located within the first 1024 cylinders of a disk. This is most likely to occur in a BIOS that does not have LBA support, or some older boot managers. It is not usually a problem in modern machines, although you should be aware that the limit may exist.

You can use `fdisk` to display units in sectors, using the `-u` option, or you can use the `u` subcommand in interactive mode to toggle between sectors and cylinders. The `parted` command supports several different units. Listing 2 illustrates the use of different units in `parted` for the same disk as used in Listing 1.

Listing 2. Using different units with parted

```
ian@attic4:~$ sudo parted /dev/sdb
[sudo] password for ian:
GNU Parted 1.8.8.1.159-1e0e
Using /dev/sdb
Welcome to GNU Parted! Type 'help' to view a list of commands.
(parted) help u
  unit UNIT                                set the default unit to UNIT

 UNIT is one of: s, B, kB, MB, GB, TB, compact, cyl, chs, %, kiB, MiB,
        GiB, TiB
(parted) p
Model: ATA HDT722525DLA380 (scsi)
Disk /dev/sdb: 250GB
Sector size (logical/physical): 512B/512B
Partition Table: msdos

Number  Start   End    Size   Type     File system  Flags
 1      32.3kB  206MB  206MB  primary  ext3
 2      206MB   107GB  106GB  primary  ext4
 3      107GB   250GB  143GB  primary  ext3

(parted) u s
(parted) p
Model: ATA HDT722525DLA380 (scsi)
Disk /dev/sdb: 488397168s
Sector size (logical/physical): 512B/512B
Partition Table: msdos

Number  Start       End         Size        Type     File system  Flags
 1      63s         401624s     401562s     primary  ext3
 2      401625s     208282724s  207881100s  primary  ext4
 3      208282725s  488392064s  280109340s  primary  ext3

(parted) u chs
(parted) p
Model: ATA HDT722525DLA380 (scsi)
Disk /dev/sdb: 30401,80,62
Sector size (logical/physical): 512B/512B
BIOS cylinder,head,sector geometry: 30401,255,63.  Each cylinder is 8225kB.
Partition Table: msdos
```

```
Number  Start      End           Type     File system  Flags
 1      0,1,0      24,254,62     primary  ext3
 2      25,0,0     12964,254,62  primary  ext4
 3      12965,0,0  30400,254,62  primary  ext3

(parted)
```

Note that the apparent discrepancy between the starting cylinder and ending cylinders shown by `parted` and `fdisk` output is due to the fact that `parted` starts counting cylinders at 0, while `fdisk` starts counting them at 1. Listing 3 shows that `fdisk` does have the same starting and ending sector as `parted`.

Listing 3. Checking start and end sector numbers

```
ian@attic4:~$ sudo fdisk -ul /dev/sdb

Disk /dev/sdb: 250.1 GB, 250059350016 bytes
255 heads, 63 sectors/track, 30401 cylinders, total 488397168 sectors
Units = sectors of 1 * 512 = 512 bytes
Disk identifier: 0x000404d6

   Device Boot      Start         End      Blocks   Id  System
/dev/sdb1              63      401624      200781   83  Linux
/dev/sdb2          401625   208282724   103940550   83  Linux
/dev/sdb3       208282725   488392064   140054670   83  Linux
ian@attic4:~$ echo $(( 208282725 / 255 / 63 ))
12965
```

Partition types

IDE drives have three types of partition: *primary*, *logical*, and *extended*. The *partition table* is located in the *master boot record (MBR)* of a disk. The MBR is the first sector on the disk, so the partition table is not a very large part of it. This limits the number of primary partitions on a disk to four. When more than four partitions are required, as is often the case, one of the primary partitions must instead become an *extended partition*. An extended partition is a container for one or more *logical* partitions. In this way, you can have more than 4 partitions on a drive using the MBR layout.

The MBR layout also limits the maximum size of disk that is supported to approximately two terabytes. The newer GUID Partition Table (or GPT) layout solves this size limitation and also the rather small limitation on the number of partitions. A disk formatted using GPT layout supports up to 128 primary partitions by default and does not use extended or logical partitions. For more information on MBR internals and how the GUID Partition Table (GPT) works, see MBR, EBR, GPT and LVM internals.

An *extended partition* is simply a container for one, or usually more, logical partitions. This partitioning scheme was originally used with MS DOS and PC DOS and permits PC disks to be used by DOS, Windows, or Linux systems. A disk may contain only one extended partition. Data is stored in the logical partitions within the extended partition. You cannot store data in an extended partition without first creating a logical partition within it.

Linux numbers primary or extended partitions as 1 through 4, so dev/sda may have four primary partitions, /dev/sda1, /dev/sda2, /dev/sda3, and /dev/sda4. Or it may have a single primary partition /dev/sda1 and an extended partition /dev/sda2. If logical partitions are defined, they are numbered starting at 5, so the first logical partition on /dev/sda will be /dev/sda5, even if there are no primary partitions and one extended partition (/dev/sda1) on the disk. So if you want more than four partitions on an IDE drive, you will lose one partition number to the extended partition. Although the theoretical maximum number of partitions on an IDE drive is now limited to 15 for kernels with hotplug, you may or may not be able to create the last few. Be careful to check that everything can work if you plan on using more than 12 partitions on a drive.

The disk used in the earlier examples has three primary partitions, all formatted for Linux use. Two use the ext3 filesystem, while the other one uses ext4. Listing 4 shows the output from the `parted` command `p` for an internal drive with primary, extended, and logical partitions on a Ubuntu 9.10 system and for a USB drive attached to a Fedora 12 system. Note the different filesystem types. Note also that you can specify one or more `parted` commands on the command line to avoid interactive mode.

Listing 4. Displaying the partition table with parted

```
ian@attic4:~$ sudo parted /dev/sda u s p
[sudo] password for ian:
Model: ATA WDC WD6401AALS-0 (scsi)
Disk /dev/sda: 1250263728s
Sector size (logical/physical): 512B/512B
Partition Table: msdos

Number  Start       End          Size         Type      File system     Flags
 1      63s         2040254s     2040192s     primary   ext3
 2      2040255s    22523129s    20482875s    primary   linux-swap(v1)
 4      22523130s   1250258624s  1227735495s  extended                  boot
 5      22523193s   167397299s   144874107s   logical   ext3
 6      167397363s  310761359s   143363997s   logical   ext3
 7      310761423s  455442749s   144681327s   logical   ext3
 8      455442813s  600092009s   144649197s   logical   ext3
[root@echidna ~]# parted /dev/sdc p
Model: WD My Book (scsi)
Disk /dev/sdc: 750GB
Sector size (logical/physical): 512B/512B
Partition Table: msdos

Number  Start   End    Size    Type      File system  Flags
 1      32.3kB  135GB  135GB   primary   fat32        lba
 2      135GB   750GB  616GB   extended
 5      135GB   292GB  157GB   logical   ext3
 6      292GB   479GB  187GB   logical   ext3
 7      479GB   555GB  76.5GB  logical   ext3
 8      555GB   750GB  195GB   logical   ext3
```

The `fdisk` command does not understand GPT formatted disks. Instead you can use either `parted`, `gparted`, or `gdisk`. See Listing 14 for an example of the output from `gdisk`.

Warning: If you use gdisk on a disk that uses MBR layout, then it will offer to convert the disk to GPT format for you. Be careful!

Logical Volume Manager (LVM)

Now that you understand the various types of partitions, you might wonder what happens if you don't plan the right sizes for your various partitions. How do you expand or contract them? Or what happens if you need more space than the capacity of a single disk for a large filesystem? Enter *Logical Volume Manager* (*LVM*).

With LVM, you can abstract the management of your disk space so a single filesystem can span multiple disks or partitions and you easily can add or remove space from filesystems. The current version is lvm2 which is backwards compatible with the original lvm (sometimes now called lvm1).

LVM manages disk space using:

- *Physical Volumes* (*PVs*)
- *Volume Groups* (*VGs*)
- *Logical Volumes* (*LVs*)

A Physical Volume is either a whole drive or a partition on a drive. Although LVM can use the whole drive without having a partition defined, this is not usually a good idea as you will see in MBR, EBR, GPT and LVM internals.

A Volume Group is a collection of one or more PVs. The space in a VG is managed as if it were one large disk, even if the underlying PVs are spread across multiple partitions or multiple disks. The underlying PVs can be different sizes and on different kinds of disks as we'll illustrate shortly.

A Logical Volume is analogous to a physical GPT or MBR partition in the sense that it is the unit of space that is formatted with a particular filesystem type such as ext4 or XFS and it is then mounted as part of your Linux filesystem. An LV resides entirely within a VG.

Think of a PV as the unit of physical space that is aggregated into an abstraction called a VG which is rather like a virtual drive. The VG or virtual drive is then partitioned into LVs for use by filesystems.

Within a VG, you manage space in terms of *extents*. The default extent size is 4MB which is usually adequate. If you use a large extent size, be aware that all PVs in a VG must use the same extent size. When you allocate or resize an LV, the allocation unit is the extent size. So the default results in LVs that are multiples of 4MB and they must be incremented or shrunk in multiples of 4MB.

The final piece of the LVM puzzle is the *device mapper*. This is a piece of the Linux kernel that provides a generic foundation for virtual devices such as LVM or software RAID.

The commands for working with LVM are usually in the lvm2 package. You can run a number of commands from the command line, or you can run the `lvm` command which provides a shell for running the various LVM commands. Listing 5 shows the `lvm` command and the various commands that can run from it.

Listing 5. The lvm command and its subcommands

```
[root@echidna ~]# lvm
lvm> help
  Available lvm commands:
```

```
  Use 'lvm help <command>' for more information

  dumpconfig      Dump active configuration
  formats         List available metadata formats
  help            Display help for commands
  lvchange        Change the attributes of logical volume(s)
  lvconvert       Change logical volume layout
  lvcreate        Create a logical volume
  lvdisplay       Display information about a logical volume
  lvextend        Add space to a logical volume
  lvmchange       With the device mapper, this is obsolete and does nothing.
  lvmdiskscan     List devices that may be used as physical volumes
  lvmsadc         Collect activity data
  lvmsar          Create activity report
  lvreduce        Reduce the size of a logical volume
  lvremove        Remove logical volume(s) from the system
  lvrename        Rename a logical volume
  lvresize        Resize a logical volume
  lvs             Display information about logical volumes
  lvscan          List all logical volumes in all volume groups
  pvchange        Change attributes of physical volume(s)
  pvresize        Resize physical volume(s)
  pvck            Check the consistency of physical volume(s)
  pvcreate        Initialize physical volume(s) for use by LVM
  pvdata          Display the on-disk metadata for physical volume(s)
  pvdisplay       Display various attributes of physical volume(s)
  pvmove          Move extents from one physical volume to another
  pvremove        Remove LVM label(s) from physical volume(s)
  pvs             Display information about physical volumes
  pvscan          List all physical volumes
  segtypes        List available segment types
  vgcfgbackup     Backup volume group configuration(s)
  vgcfgrestore    Restore volume group configuration
  vgchange        Change volume group attributes
  vgck            Check the consistency of volume group(s)
  vgconvert       Change volume group metadata format
  vgcreate        Create a volume group
  vgdisplay       Display volume group information
  vgexport        Unregister volume group(s) from the system
  vgextend        Add physical volumes to a volume group
  vgimport        Register exported volume group with system
  vgmerge         Merge volume groups
  vgmknodes       Create the special files for volume group devices in /dev
  vgreduce        Remove physical volume(s) from a volume group
  vgremove        Remove volume group(s)
  vgrename        Rename a volume group
  vgs             Display information about volume groups
  vgscan          Search for all volume groups
  vgsplit         Move physical volumes into a new or existing volume group
  version         Display software and driver version information
lvm> quit
  Exiting.
```

To give you a brief sampling of LVM in action, I created a GPT partition for LVM (partition type 0083) on /dev/sdc5 and an MBR partition for LVM (partition type 83) on /dev/sdd1. You have already created PVs on these as shown in Listing 6 where you use the `pvscan` command to display the PVs on the system.

Listing 6. Displaying your physical volumes

```
[root@echidna ~]# pvscan
  PV /dev/sdc5                      lvm2 [146.43 GiB]
  PV /dev/sdd1                      lvm2 [232.88 GiB]
  Total: 2 [379.32 GiB] / in use: 0 [0   ] / in no VG: 2 [379.32 GiB]
```

Now you will use the `vgcreate` command to create a volume group from these two PVs and then use the `lvcreate` command to create a logical volume that is larger than either of the PVs. Finally you will format your new LV as ext4 and mount it at /mnt/lvdemo as shown in Listing 7.

Listing 7. Creating a volume group and a logical volume

```
[root@echidna ~]# vgcreate demo-vg /dev/sdc5 /dev/sdd1
  Volume group "demo-vg" successfully created
[root@echidna ~]# lvcreate -L 300G -n demo-lv demo-vg
  Logical volume "demo-lv" created
[root@echidna ~]# lvscan
  ACTIVE            '/dev/demo-vg/demo-lv' [300.00 GiB] inherit
[root@echidna ~]# mkfs -t ext4 /dev/demo-vg/demo-lv
mke2fs 1.41.14 (22-Dec-2010)
Filesystem label=
OS type: Linux
Block size=4096 (log=2)
Fragment size=4096 (log=2)
Stride=0 blocks, Stripe width=0 blocks
19660800 inodes, 78643200 blocks
3932160 blocks (5.00%) reserved for the super user
First data block=0
Maximum filesystem blocks=4294967296
2400 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

Writing inode tables: done
Creating journal (32768 blocks): done
Writing superblocks and filesystem accounting information: done

This filesystem will be automatically checked every 36 mounts or
180 days, whichever comes first.  Use tune2fs -c or -i to override.
[root@echidna ~]# mount /dev/demo-vg/demo-lv /mnt/lvdemo
[root@echidna ~]# df -h /mnt/lvdemo/
Filesystem                      Size  Used Avail Use% Mounted on
/dev/mapper/demo--vg-demo--lv  296G  191M  281G   1% /mnt/lvdemo
```

Note:

1. Filesystem commands such as `mkfs` and `mount` access the LV using a name like /dev/<vg-name>/<lv-name>.
2. By default, when you create an LV it immediately becomes *active* as shown by the output of the `vgscan` command in Listing 7. If the LV is on a removable drive you need to deactivate it using the `lvchange` command before removing the drive from the system.

To learn more about LVM than this brief introduction provides, refer to the developerWorks article "Logical volume management" (see Resources).

MBR, EBR, GPT and LVM internals

Before you learn about allocating disk space, take a little detour through the internals of MBR, EBR, GPT and LVM partition tables to help reinforce the concepts as they can be difficult to grasp.

Note: You do not need this level of detail for the LPI exam, so feel free to skip to Allocating disk space if you are pressed for time or less interested in internals.

Master Boot Record (MBR)

The Master Boot Record is the first sector on a hard drive. The MBR contains the bootstrap code, and possibly some other information, followed by the 64-byte partition table and a two-byte boot signature. The 64-byte partition table has four 16-byte entries and starts at offset 446 (1BEh). Table 2 shows the layout of each 16-byte entry.

Table 2. Partition table entry format

Offset (hex)	Length	Description
0h	1	Status. 80h indicates active (or bootable) partition.
1h	3	CHS (Cylinder-Head-Sector) address of first absolute sector in partition
4h	1	Partition type.
5h	3	CHS (Cylinder-Head-Sector) address of last absolute sector in partition
8h	4	Logical Block Address (LBA) of first absolute sector in partition.
Ch	4	Count of sectors in partition

Look at a real example. The root user can read sectors from a disk directly using the `dd` command. Listing 8 shows the output from dumping the first 510 bytes of the MBR on /dev/sda, then using the `tail` command to select only the last 64 bytes of the record which you then display in hexadecimal.

Listing 8. Displaying the partition table on /dev/sda

```
[root@echidna ~]# dd if=/dev/sda bs=510 count=1 2>/dev/null|tail -c 64 |hexdump -C
00000000  80 01 01 00 07 fe ff ff  3f 00 00 00 98 66 b9 08  |........?....f..|
00000010  00 fe ff ff 83 fe ff ff  61 5c 39 09 21 c7 17 00  |........a\9.!...|
00000020  00 fe ff ff 05 fe ff ff  82 23 51 09 85 ab 68 66  |.........#Q...hf|
00000030  00 fe ff ff 82 fe ff ff  d7 66 b9 08 8a f5 7f 00  |.........f......|
00000040
```

Notice that the first record has a status of 80h indicating a bootable partition and a partition type of 07h, indicating an NTFS partition. The remaining partition types are 83h (Linux), 05h (Extended), and 82h (Linux swap), so the disk has three primary partitions and one extended partition. All of the CHS values are feffff, which is typical on disks that use LBA. The LBA starting sectors and sector counts are interpreted as 32-bit little-endian integers, so 98 66 b9 08 represents 08b96698h. So the first (NTFS) partition starts at sector 63 (3fh) and extends for 146368152 (08b96698h) sectors. Thus `fdisk` would show an ending sector of 63+146368152-1=146368214.

Notes:

1. The partition table in the MBR does not contain any information about logical partitions. You'll see how those are found shortly.
2. If your disk uses the CHS values rather than the LBA values, you will need to do some additional arithmetic to translate the CHS values into absolute sectors. Other than using

different arithmetic to find the absolute sectors, the principles are the same as those outlined here, so I won't cover CHS computations in this short introduction.

Listing 9. Fdisk output for /dev/sda

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

Disk /dev/sda: 1000.2 GB, 1000204886016 bytes
255 heads, 63 sectors/track, 121601 cylinders, total 1953525168 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x000de20f

   Device Boot      Start         End      Blocks   Id  System
/dev/sda1   *          63   146368214    73184076    7  HPFS/NTFS/exFAT
/dev/sda2       154754145   156312449      779152+  83  Linux
/dev/sda3       156312450  1874448134   859067842+   5  Extended
/dev/sda4       146368215   154754144     4192965   82  Linux swap / Solaris
/dev/sda5       156312513   336031604    89859546   83  Linux
/dev/sda6       336031668   636880859   150424596   83  Linux
/dev/sda7       636880923   865983824   114551451   83  Linux
/dev/sda8       865983888   949891319    41953716   83  Linux
/dev/sda9       949891383   954003959     2056288+   b  W95 FAT32
/dev/sda10      954007552  1187801087   116896768   83  Linux
/dev/sda11     1187803136  1229760511    20978688   83  Linux
/dev/sda12     1229762560  1874446335   322341888   83  Linux

Partition table entries are not in disk order
```

And indeed /dev/sda1 ends at 146368214.

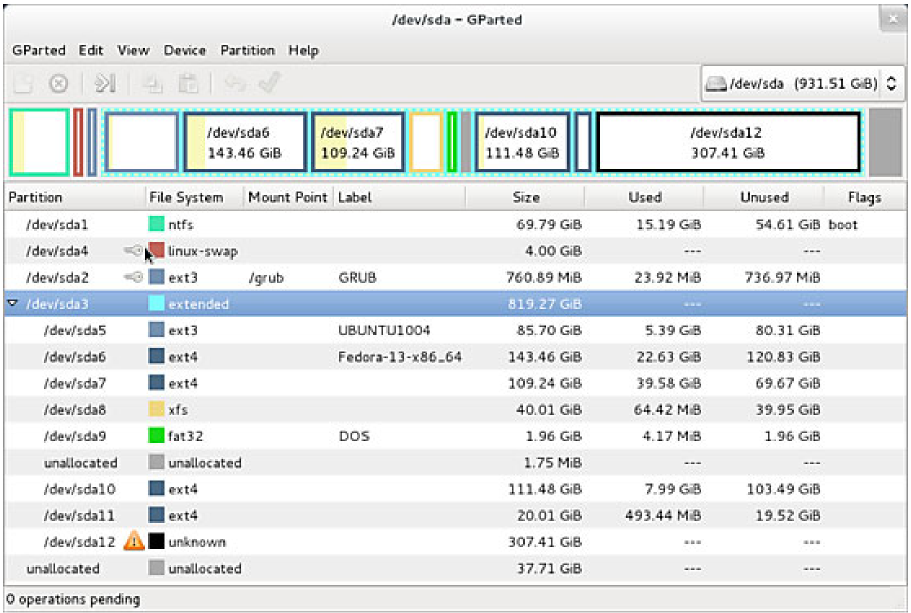
Note the warning that the partition table entries are not in order. This is because /dev/sda4, the Linux swap partition is physically before /dev/sda3, the extended partition. This is not an error, although the `fdisk` program and some other tools have options to rewrite the partition table so that it is neatly in order.

Extended Boot Record (EBR)

Before you explore extended partitions further, take a quick look at the graphical output from the `gparted` command as that will give you a graphical picture of the partition layout and show the container nature of the extended partition,

Figure 1 shows the display from the `gparted` command for /dev/sda. As noted earlier, the disk has three primary partitions (/dev/sda1, /dev/sda4, and /dev/sda2) and one extended partition (/dev/sda3). The extended partition contains logical partitions /dev/sda5 through /dev/sda12. In the pictorial display at the top of the image the extended partition is shown as frame (colored light blue) around the logical partitions.

Figure 1. Using gparted to display of a disk with several partition types



As you saw above, the MBR does not contain partition table entries for the logical partitions; it defines a container that looks to the rest of the system something like a special partition. The logical partitions are defined within this container. So how does this work?

With no hard limit on the number of logical partitions inside an extended partition, no fixed-size partition table defines the logical partitions. Instead, there is an *Extended Boot Record* (or *EBR*) for **each** logical partition. Like the MRB, the EBR is 512 bytes in length and it uses a partition table at offset 446 (1BEh) as for the MBR. Only two entries in the EBR partition table are used. The first defines the offset and size of the current partition and the second defines the offset and count to the end of the next logical partition. For the last logical partition in this singly-linked chain the second entry contains zeroes.

To ease the job of converting little-endian hex digits to decimal numbers, create a small Bash function to display the first two entries from an EBR. Then use the formatting string capability of the `hexdump` command to display the LBA starting sector and sector count as decimal numbers instead of hex digits. Listing 10 shows the `showebr` function.

Listing 10. Function to display an EBR

```
showebr ()
{
    dd if=$1 skip=$2 bs=512 count=1 2> /dev/null | tail -c 66 |
     hexdump -n 32 -e '"%07.7_ax  " 8/1 "%2.2x " 2/4 " %12d" "\n"'
}
```

The first EBR is the first sector of your extended partition, so use your new `showebr` function to display it using the sector offset from Listing 9. Listing 11 shows the output.

Listing 11. The first EBR on /dev/sda

```
[root@echidna ~]# showebr /dev/sda 156312450
0000000  00 fe ff ff 83 fe ff ff          63    179719092
0000010  00 fe ff ff 05 fe ff ff   179719155    300849255
```

The first entry in the EBR partition table shows that the partition (/dev/sda5) starts at offset 63 from this EBR and the partition contains 179719092 sectors. The second entry shows the next EBR starting at offset 179719155 from the **first** EBR in the extended partition, while the count is the count of sectors starting at the **next** EBR to the end of the next logical partition. Listing 12 shows the partition information for the extended and logical partitions on /dev/sda. The sector count for /dev/sda6 is 636880859-336031668+1=300849192. Given the sector count of 300849255 in the second entry of Listing 11 you can deduce that /dev/sda6 starts at offset 300849255-300849192=63. This is a common offset for systems, including Windows XP.

Listing 12. Extended and logical partitions on /dev/sda

```
dev/sda3       156312450  1874448134   859067842+   5  Extended
/dev/sda5       156312513   336031604    89859546   83  Linux
/dev/sda6       336031668   636880859   150424596   83  Linux
/dev/sda7       636880923   865983824   114551451   83  Linux
/dev/sda8       865983888   949891319    41953716   83  Linux
/dev/sda9       949891383   954003959     2056288+   b  W95 FAT32
/dev/sda10      954007552  1187801087   116896768   83  Linux
/dev/sda11     1187803136  1229760511    20978688   83  Linux
/dev/sda12     1229762560  1874446335   322341888   83  Linux
```

Using what you just learned, along with the `showebr` function and some inline shell arithmetic, you can now walk the entire EBR chain for /dev/sda as shown in Listing 13.

Listing 13. Walking the chain of logical partitions on /dev/sda

```
[root@echidna ~]# showebr /dev/sda 156312450
0000000  00 fe ff ff 83 fe ff ff          63    179719092
0000010  00 fe ff ff 05 fe ff ff   179719155    300849255
[root@echidna ~]# showebr /dev/sda $(( 156312450 + 179719155 ))
0000000  00 fe ff ff 83 fe ff ff          63    300849192
0000010  00 fe ff ff 05 fe ff ff   480568410    229102965
[root@echidna ~]# showebr /dev/sda $(( 156312450 + 480568410 ))
0000000  00 fe ff ff 83 fe ff ff          63    229102902
0000010  00 fe ff ff 05 fe ff ff   709671375     83907495
[root@echidna ~]# showebr /dev/sda $(( 156312450 + 709671375 ))
0000000  00 fe ff ff 83 fe ff ff          63     83907432
0000010  00 fe ff ff 05 fe ff ff   793578870      4112640
[root@echidna ~]# showebr /dev/sda $(( 156312450 + 793578870 ))
0000000  00 fe ff ff 0b fe ff ff          63      4112577
0000010  00 fe ff ff 05 fe ff ff   797691510    233797128
[root@echidna ~]# showebr /dev/sda $(( 156312450 + 797691510 ))
0000000  00 fe ff ff 83 fe ff ff        3592    233793536
0000010  00 fe ff ff 05 fe ff ff  1031488638     41959424
[root@echidna ~]# showebr /dev/sda $(( 156312450 + 1031488638 ))
0000000  00 fe ff ff 83 fe ff ff        2048     41957376
0000010  00 fe ff ff 05 fe ff ff  1073448062    644685824
[root@echidna ~]# showebr /dev/sda $(( 156312450 + 1073448062 ))
0000000  00 fe ff ff 83 fe ff ff        2048    644683776
0000010  00 00 00 00 00 00 00 00           0            0
```

Notice that one partition has an offset of 3592. This corresponds to the 1.75MB of free space between /dev/sda9 and /dev/sda10. Note also that /dev/sda12 starts at offset 2048 instead of

63. You might find this on drives which use 2048-byte sectors or solid state drives (SSDs) where sector alignment is important.

GUID Partition Table (GPT)

I mentioned earlier that the MBR layout limits disks with 512-byte sectors to 2TB in size. With the advent of disks larger than 2TB a newer layout called *GUID Partition Table* or (GPT) was designed. This format can be used on smaller disks, but is required for larger disks. Disk formatting puts a so-called *protective* MBR in the usual MBR location so that operating systems and utilities that do not understand GPT see the whole disk as an unknown partition type (EEh), with the size appearing to be clipped to either the whole disk or the maximum supported by MBR for the sector size of the disk.

Recent Linux distributions include tools for formatting and booting from GPT disks, including `gdisk` (similar to `fdisk`), `parted`, `gparted`, `grub2`, and recent versions of `grub`.

Recent Linux systems support booting from GPT partitions using either the traditional PC BIOS or the newer *Extensible Firmware Interface* (or *EFI*). Recent 64-bit versions of Windows also support booting from GPT partitions using EFI.

GPT partitions contain a GPT header and a GPT partition array. The GPT partition array contains at least 128 entries, so a GPT disk can support at least 128 primary partitions. The GPT header is in the first sector of the GPT partition and the partition array follows in the next sector. Both the GPT header and the partition array are replicated at the end of the partition. For additional protection the header also contains a 32-bit CRC of the partition array.

Listing 14 shows the output from the `gdisk` command for a removable disk (/dev/sdc) with five primary partitions.

Listing 14. Output from gdisk for /dev/sdc

```
[root@echidna ~]# gdisk -l /dev/sdc
GPT fdisk (gdisk) version 0.8.4

Partition table scan:
  MBR: protective
  BSD: not present
  APM: not present
  GPT: present

Found valid GPT with protective MBR; using GPT.
Disk /dev/sdc: 3906963456 sectors, 1.8 TiB
Logical sector size: 512 bytes
Disk identifier (GUID): 7E637BAC-33FC-46D3-9860-6A300CDD0E5F
Partition table holds up to 128 entries
First usable sector is 34, last usable sector is 3906963422
Partitions will be aligned on 2048-sector boundaries
Total free space is 1361893309 sectors (649.4 GiB)

Number  Start (sector)    End (sector)  Size       Code  Name
   1            2048       773580799   368.9 GiB   0700  Ian's partition 1
   2       773580800      1387724799   292.8 GiB   0700  Ian's partition 2
   3      1387724800      1831110655   211.4 GiB   0700
   4      1831110656      2237978623   194.0 GiB   0700
   5      2237978624      2545072127   146.4 GiB   8E00
```

You will find lots more information on GPT in our article "Make the most of large drives with GPT and Linux" (see Resources).

Partitioning for physical and logical volume management

We mentioned above that LVM can utilize a whole disk without the disk first being partitioned. For the same reason that GPT uses a protective MBR, you should create a partition on your disk and then create a PV in the partition rather than using the native LVM capabilities to define the whole disk as a PV. That way, other operating systems that do not understand LVM will recognize that the disk has an unknown partition on it rather than thinking it is not formatted.

The management of the abstraction layers used in LVM obviously requires considerably more information than the original MBR partition table. Other than advising you to create a partition for your PV, we will not further go into the on-disk data structures used by LVM. To learn more about LVM, refer to the developerWorks article "Logical volume management" (see Resources).

Allocating disk space

As mentioned earlier, a Linux filesystem is a single large tree rooted at /. It is fairly obvious why data on floppy disks or CD-ROMs must be mounted, but perhaps less obvious why you should consider separating data that is stored on hard drives. Some good reasons for separating filesystems include:

- Boot files. Some files must be accessible to the BIOS or boot loader at boot time.
- Multiple hard drives. Typically each hard drive will be divided into one or more partitions, each with a filesystem that must be mounted somewhere in the filesystem tree.
- Shareable files. Several system images may share static files such as executable program files. Dynamic files such as user home directories or mail spool files may also be shared, so that users can log in to any one of several machines on a network and still use the same home directory and mail system.
- Potential overflow. If a filesystem might fill to 100 percent of its capacity, it is usually a good idea to separate this from files that are needed to run the system.
- Quotas. Quotas limit the amount of space that users or groups can take on a filesystem.
- Read-only mounting. Before the advent of journalling filesystems, recovery of a filesystem after a system crash often took a long time. Therefore, you can mount filesystems that seldom change (such as a directory of executable programs) as read-only to reduce the time spent checking it after a system crash.

In addition to the filesystem use covered so far, you also need to consider allocating swap space on disk. For a Linux system, this is usually one, or possibly multiple, dedicated partitions.

Making choices

Let's assume you are setting up a system that has at least one hard drive, and you want to boot from the hard drive. (This article does not cover setup for a diskless workstation that is booted over a LAN or considerations for using a live CD or DVD Linux system.) Although it may be possible to change partition sizes later, this usually takes some effort, so making good choices up front is important. Let's get started.

Your first consideration is to ensure that your system will be bootable. Some older systems have a limitation that the BIOS can boot only from a partition that is wholly located within the first 1024 cylinders of disk. If you have such a system, then you **must** create a partition that will eventually be mounted as /boot that will hold the key files needed to boot the system. Once these have been loaded, the Linux system will take over operation of the disk, and the 1024 cylinder limit will not affect further operation of the system. If you need to create a partition for /boot, approximately 100 megabytes (MB) is usually sufficient.

Your next consideration is likely to be the amount of required swap space. With current memory prices, swap space represents a very slow secondary memory. A once common rule of thumb was to create swap space equivalent to the amount of real RAM. Today, you might want to configure one or two times real RAM for a workstation so that you can use several large programs without running out of RAM. Even if it is slow to switch between them, you are probably working in only one or two at any given time.

A large swap space is also advisable for a system with very small memory. For a server, you might want to use a swap space of about half of your RAM, unless you are running an application that recommends a different value. In any event, you should monitor server memory usage so that you can add real RAM or distribute the workload across additional servers if needed. Too much swapping is seldom good on a server. It is possible to use a swap file, but a dedicated partition performs better.

Now we come to a point of divergence. Use of a personal workstation tends to be much less predictable than use of a server. My preference, particularly for new users, is to allocate most of the standard directories (/usr, /opt, /var, etc.) into a single large partition. This is especially useful for new users who may not have a clear idea of what will be installed down the line. A workstation running a graphical desktop and a reasonable number of development tools will likely require 5 or more gigabytes of disk space plus space for user needs. Some larger development tools may require several gigabytes each. I usually allocate somewhere between 40 GB and 60 GB per operating system, and I leave the rest of my disk free to load other distributions.

Server workloads will be more stable, and running out of space in a particular filesystem is likely to be more catastrophic. So, for them, you will generally create multiple partitions, spread across multiple disks, possibly using hardware or software RAID or logical volume groups.

You will also want to consider the workload on a particular filesystem and whether the filesystem is shared among several systems or used by just one system. You may use a combination of experience, capacity planning tools, and estimated growth to determine the best allocations for your system.

Regardless of whether you are configuring a workstation or a server, you will have certain files that are unique for each system located on the local drive. Typically, these include /etc for system parameters, /boot for files needed during boot, /sbin for files needed for booting or system recovery, /root for the root user's home directory, /var/lock for lock files, /var/run for running system information, and /var/log for log files for this system. Other filesystems, such as /home for user home directories, /usr, /opt, /var/mail, or /var/spool/news may be on separate partitions, or network mounted, according to your installation needs and preferences.

Resources

Learn

- Use the developerWorks roadmap for LPIC-1 to find the developerWorks articles to help you study for LPIC-1 certification based on the April 2009 objectives.
- At the LPIC Program site, find detailed objectives, task lists, and sample questions for the three levels of the Linux Professional Institute's Linux system administration certification. In particular, see their April 2009 objectives for LPI exam 101 and LPI exam 102. Always refer to the LPIC Program site for the latest objectives. Note the Exam 101: Objective Changes as of July 2, 2012.
- Review the entire LPI exam prep series on developerWorks to learn Linux fundamentals and prepare for system administrator certification based on earlier LPI exam objectives prior to April 2009.
- Learn more about FHS at the Filesystem Hierarchy Standard home page.
- Learn more about GUID Partition Tables (GPT) in the developerWorks article Make the most of large drives with GPT and Linux (developerWorks, July 2012).
- Read Basic tasks for new Linux developers (developerWorks, April 2011), learn to open a terminal window or shell prompt and much more.
- The Linux Documentation Project has a variety of useful documents, especially its HOWTOs.
- In the developerWorks Linux zone, find more resources for Linux developers and system administrators.
- Stay current with developerWorks technical events and Webcasts.
- Follow developerWorks on Twitter.

Get products and technologies

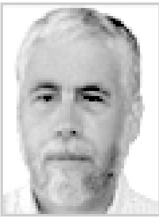
- Download the System rescue CD-Rom, one of many tools available online to help you recover a system after a crash.
- With IBM trial software, available for download directly from developerWorks, build your next development project on Linux.

Discuss

- Participate in the discussion forum for this content.
- Get involved in the developerWorks Community. Connect with other developerWorks users while exploring the developer-driven blogs, forums, groups, and wikis.

About the author

Ian Shields



Ian Shields works on a multitude of Linux projects for the developerWorks Linux zone. He is a Senior Programmer at IBM at the Research Triangle Park, NC. He joined IBM in Canberra, Australia, as a Systems Engineer in 1973, and has since worked on communications systems and pervasive computing in Montreal, Canada, and RTP, NC. He has several patents and has published several papers. His undergraduate degree is in pure mathematics and philosophy from the Australian National University. He has an M.S. and Ph.D. in computer science from North Carolina State University. Learn more about Ian in Ian's profile on developerWorks Community.

© Copyright IBM Corporation 2010, 2012
(www.ibm.com/legal/copytrade.shtml)
Trademarks
(www.ibm.com/developerworks/ibm/trademarks/)