developerWorks®

Learn Linux, 101 : Hard disk layout

Planning your hard disk partitions

Ian Shields
Linux Author
Freelance

13 August 2015
(First published 24 February 2010)

Learn how to design a partition layout for disks on a Linux® system. Use the material in this tutorial to study for the Linux Professional Institute LPIC-1: Linux Server Professional Certification exam 101, or to learn for fun.

View more content in this series

Overview

In this tutorial, 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)

Linux filesystems

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 can 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:, and so on), a Linux filesystem is a single tree with the / directory as its root directory.

About this series

This series of tutorials helps you learn Linux system administration tasks. You can also use the material in these tutorials to prepare for the Linux Professional Institute's LPIC-1: Linux Server Professional Certification exams.

See "*Learn Linux, 101*: A roadmap for LPIC-1" for a description of and link to each tutorial in this series. The roadmap is in progress and reflects the version 4.0 objectives of the LPIC-1 exams as updated April 15th, 2015. As tutorials are completed, they will be added to the roadmap.

This tutorial helps you prepare for Objective 102.1 in Topic 102 of the Linux Server Professional (LPIC-1) exam 101. The objective has a weight of 2.

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

Prerequisites

To get the most from the tutorials 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 tutorial. Sometimes different versions of a program will format output differently, so your results might not always look exactly like the listings and figures shown here.

Branching out with mount

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.

Usually, the kernel starts this mount process by mounting the filesystem on some hard drive partition as /. You can mount other hard drive partitions as /boot, /tmp, or /home. You can mount the filesystem on a floppy drive as /mnt/floppy, and the filesystem on a CD-ROM as /media/cdrom1, for example. You can 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 /dev/sda 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.

MBR partitions

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 can 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 (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 used to be allocated as an integral number of cylinders (based on the possibly inaccurate notion of a cylinder). There are still vestigial remains of cylinder allocation, such as the `-C` option of `parted`, which specifies that allocation of new partitions should be on cylinder boundaries.

You can use either the `fdisk` or `parted` commands to display partition information on MBR disks. Both commands can operate interactively, or you can pass parameters on the command line. You can use the `-v` parameter of either to display information about the program version. Both support several different units for displaying and allocation partitions. Listing 1 shows an example of using `fdisk` with parameters and `parted` interactively to display the partitions on an MBR formatted disk using sectors as units. You will need to be root or have root authority via `sudo`, as shown here, to display or manipulate the partition table.

Note: All of these partitioning tools will keep changes in memory until you choose to write the updates to disk. If in doubt, **QUIT**: You might render your disk or system unusable if you do not know what you are doing or you make a mistake.

Listing 1. Displaying MBR partitions with fdisk and parted

```
[ian@atticf20 ~]$ fdisk -v
fdisk from util-linux 2.26.2
[ian@atticf20 ~]$ fdisk -l /dev/sdb
fdisk: cannot open /dev/sdb: Permission denied
[ian@atticf20 ~]$ sudo fdisk -l /dev/sdb
[sudo] password for ian:
Disk /dev/sdb: 232.9 GiB, 250059350016 bytes, 488397168 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
Disklabel type: dos
Disk identifier: 0x000404d6

Device     Boot     Start       End   Sectors   Size Id Type
/dev/sdb1              63    401624    401562 196.1M 83 Linux
/dev/sdb2          401625 252786687 252385063 120.4G 83 Linux
/dev/sdb3       252786688 375631871 122845184  58.6G 83 Linux
/dev/sdb4       375647895 488392064 112744170  53.8G 83 Linux
[ian@atticf20 ~]$ sudo parted /dev/sdb
GNU Parted 3.2
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) u s
(parted) p
Model: ATA HDT722525DLA380 (scsi)
Disk /dev/sdb: 488397168s
Sector size (logical/physical): 512B/512B
Partition Table     : msdos
Disk Flags:

Number  Start       End         Size        Type     File system  Flags
 1      63s         401624s     401562s     primary  ext3
 2      401625s     252786687s  252385063s  primary  ext4
 3      252786688s  375631871s  122845184s  primary  ext3
 4      375647895s  488392064s  112744170s  primary  ext4

(parted) q
```

Note that the sector size is given as both logical and physical. Since late 2009, hard drive companies have started migrating to larger sectors on disk. These larger sectors provide improved performance and better error recovery. Disks using these larger 4K sectors are called Advanced Format by IDEMA (The International Disk Drive Equipment and Materials Association). See Resources for more information. Because most operating systems expect 512 byte sectors, Advanced Format drives provide 512 byte sector emulation. Performance is generally improved if a partition starts on a logical sector that is a multiple of 4096 (sector 0, 8, 16, and so on).

You can see the nominal geometry on a Linux system using either `parted` or `fdisk`. 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.

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 might 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 use `fdisk` to display units in cylinders, using the `-u=cylinders` option, or you can use the `u` subcommand in interactive mode to toggle between sectors and cylinders. Recent versions of `fdisk` will tell you that the cylinders unit is now deprecated. The `parted` command supports several different units. Listing 2 displays partition information in cylinders using `fdisk` and `parted` for the same disk as used in Listing 1. We also illustrate greater detail using the `chs` (cylinder, head, sector) unit available in `parted`. For this example we'll use `fdisk` interactively, and show you how to enter a string of `parted` subcommands on the command line.

Listing 2. Displaying MBR disk geometry with fdisk and parted

```
[ian@atticf20 ~]$ sudo fdisk /dev/sdb

Welcome to fdisk (util-linux 2.26.2).
Changes will remain in memory only, until you decide to write them.
Be careful before using the write command.


Command (m for help): u
Changing display/entry units to cylinders (DEPRECATED!).

Command (m for help): p
Disk /dev/sdb: 232.9 GiB, 250059350016 bytes, 488397168 sectors
Geometry: 255 heads, 63 sectors/track, 30401 cylinders
Units: cylinders of 16065 * 512 = 8225280 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0x000404d6

Device     Boot Start   End Cylinders   Size Id Type
/dev/sdb1           1    25        25 196.1M 83 Linux
/dev/sdb2          26 15736     15711 120.4G 83 Linux
/dev/sdb3       15736 23383      7647  58.6G 83 Linux
/dev/sdb4       23384 30401      7019  53.8G 83 Linux
```

```
Command (m for help): q

[ian@atticf20 ~]$ sudo parted  /dev/sdb u cyl p u chs p
Model: ATA HDT722525DLA380 (scsi)
Disk /dev/sdb: 30401cyl
Sector size (logical/physical): 512B/512B
BIOS cylinder,head,sector geometry: 30401,255,63.  Each cylinder is 8225kB.
Partition Table   : msdos
Disk Flags:

Number  Start     End       Size      Type     File system  Flags
 1      0cyl      24cyl     24cyl     primary  ext3
 2      25cyl     15735cyl  15710cyl  primary  ext4
 3      15735cyl  23382cyl  7646cyl   primary  ext3
 4      23383cyl  30400cyl  7018cyl   primary  ext4

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
Disk Flags:

Number  Start       End           Type     File system  Flags
 1      0,1,0       24,254,62     primary  ext3
 2      25,0,0      15735,62,6    primary  ext4
 3      15735,62,7  23382,0,41    primary  ext3
 4      23383,0,0   30400,254,62  primary  ext4
```

Note the apparent discrepancy between the starting cylinder and ending cylinders shown by `parted` and `fdisk` output when using cylinders as the unit. This is due to the fact that `parted` starts counting cylinders at 0, while `fdisk` starts counting them at 1. As you can see from the `parted` display of chs information, two cylinders don't start on a cylinder boundary (head not equal to 0) and two do not end on a cylinder boundary (sector not equal to 62).

GPT partitions

Disks formatted with the GUID Partition Table (GPT) do not have the concept of geometry as MBR disks do. For these disks you use the `gdisk` (GPT fdisk) command or `parted`. If you use `gdisk` on an MBR formatted disk it will do an in-memory conversion of the disk to GPT format. You **can** write this to convert the disk, but this is a very risky operation, so make sure you have a good backup before you try it.

Note: GPT was designed for use with a UEFI-based (Unified Extensible Firmware Interface) system rather than a BIOS-based system to boot from a GPT disk, although there are ways around this. When planning your partitions keep this in mind.

In Listing 3 we use `parted` with the `-l` option to display information about all the disks on a system that has two GPT disks. One (/dev/sda) is the original solid state drive (SSD) as supplied by Lenovo with Windows 8. The second (/dev/sdb) is a 16GB USB flash drive with Ubuntu 15.04 installed.

Listing 3. Displaying GPT partitions with parted and gdisk

```
ian@yoga-u15:~$ sudo parted -l
```

```
Model: ATA SAMSUNG MZMTD512 (scsi)
Disk /dev/sda: 512GB
Sector size (logical/physical): 512B/512B
Partition Table  : gpt
Disk Flags:

Number  Start   End     Size    File system  Name                          Flags
 1      1049kB  1050MB  1049MB  ntfs         Basic data partition          hidden, diag
 2      1050MB  1322MB  273MB   fat32        EFI system partition          boot, hidden, esp
 3      1322MB  2371MB  1049MB  fat32        Basic data partition          hidden
 4      2371MB  2505MB  134MB                Microsoft reserved partition  msftres
 5      2505MB  470GB   467GB   ntfs         Basic data partition          msftdata
 6      470GB   497GB   26.8GB  ntfs         Basic data partition          msftdata
 7      497GB   512GB   15.5GB  ntfs         Basic data partition          hidden, diag


Model:  USB DISK 2.0 (scsi)
Disk /dev/sdb: 32.1GB
Sector size (logical/physical): 512B/512B
Partition Table  : gpt
Disk Flags:

Number  Start   End     Size    File system     Name  Flags
 1      1049kB  316MB   315MB                         bios_grub
 2      316MB   4510MB  4194MB  linux-swap(v1)
 3      4510MB  32.1GB  27.6GB  ext4

ian@yoga-u15:~$ sudo gdisk -l /dev/sdb
GPT fdisk (gdisk) version 0.8.10

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

Found valid GPT with protective MBR; using GPT.
Disk /dev/sdb: 62685184 sectors, 29.9 GiB
Logical sector size: 512 bytes
Disk identifier (GUID): FEE4E37D-AE90-43A1-A6A7-97ADA4618384
Partition table holds up to 128 entries
First usable sector is 34, last usable sector is 62685150
Partitions will be aligned on 2048-sector boundaries
Total free space is 4029 sectors (2.0 MiB)

Number  Start (sector)    End (sector)  Size       Code  Name
   1            2048          616447   300.0 MiB   EF02
   2          616448         8808447   3.9 GiB     8200
   3         8808448        62683135   25.7 GiB    8300
```

MBR Partition types

IDE drives formatted for MBR use 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 four partitions on a drive using the MBR layout.

The MBR layout also limits the maximum size of disk that is supported to approximately 2TB. 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. You can have only one extended partition and the space within it must be contiguous. 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 can have four primary partitions, /dev/sda1, /dev/sda2, /dev/sda3, and /dev/sda4. Or it can have up to three primary partitions and one extended partition. A common layout is to 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 might or might 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.

Listing 4 shows the output from the `parted` command `p` for an internal drive with primary, extended, and logical partitions on a Fedora 22 system. Note the different filesystem types and compare them with earlier listings, including the GPT listings.

Listing 4. An example of MBR partition types

```
[ian@atticf20 ~]$ sudo parted /dev/sda p
Model: ATA WDC WD6401AALS-0 (scsi)
Disk /dev/sda: 640GB
Sector size (logical/physical): 512B/512B
Partition Table  : msdos
Disk Flags:

Number  Start   End     Size    Type      File system     Flags
 1      32.3kB  1045MB  1045MB  primary   ext3
 2      1045MB  11.5GB  10.5GB  primary   linux-swap(v1)
 4      11.5GB  640GB   629GB   extended
 5      11.5GB  85.7GB  74.2GB  logical   ext4            boot
 6      85.7GB  159GB   73.4GB  logical   ext4
 7      159GB   233GB   74.1GB  logical   ext4
 8      233GB   307GB   74.1GB  logical   ext3
 9      307GB   381GB   73.9GB  logical   ext3
10      381GB   470GB   89.1GB  logical   ext4
11      470GB   552GB   82.2GB  logical   ext4
12      552GB   640GB   87.7GB  logical   ext4
```

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 the section 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 is illustrated 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 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 larger 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 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. As with other partitioning tools, you have only limited function unless you have root authority.

Listing 5. The lvm command and its subcommands

```
[[ian@atticf20 ~]$ lvm
  WARNING: Running as a non-root user. Functionality might be unavailable.
lvm> help
  /run/lvm/lvmetad.socket: connect failed: Permission denied
  WARNING: Failed to connect to lvmetad. Falling back to internal scanning.
  Available lvm commands:
  Use 'lvm help <command>' for more information

  devtypes        Display recognised built-in block device types
  dumpconfig      Dump 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 can 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
  tags            List tags defined on this host
  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, Listing 6 shows two PVs on a USB drive where you use the `pvscan` command to display the PVs on the system.

Listing 6. Displaying your physical volumes

```
[ian@atticf20 ~]$ sudo pvscan
  PV /dev/sde8         lvm2 [137.58 GiB]
  PV /dev/sde7         lvm2 [137.06 GiB]
  Total: 2 [274.64 GiB] / in use: 0 [0   ] / in no VG: 2 [274.64 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

```
[ian@atticf20 ~]$ sudo vgcreate demo-vg /dev/sde7 /dev/sde8
  Volume group "demo-vg" successfully created
[ian@atticf20 ~]$ sudo lvcreate -L 200G -n demo-lv demo-vg
  Logical volume "demo-lv" created.
[ian@atticf20 ~]$ sudo lvscan
  ACTIVE            '/dev/demo-vg/demo-lv' [200.00 GiB] inherit
[ian@atticf20 ~]$ sudo mkfs -t ext4 /dev/demo-vg/demo-lv
mke2fs 1.42.12 (29-Aug-2014)
Creating filesystem with 52428800 4k blocks and 13107200 inodes
Filesystem UUID: d8589116-f58f-4e09-ac70-77543afd49da
Superblock backups stored on blocks:
 32768, 98304, 163840, 229376, 294912, 819200, 884736, 1605632, 2654208,
 4096000, 7962624, 11239424, 20480000, 23887872

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

[ian@atticf20 ~]$ sudo mkdir /mnt/lvdemo
[ian@atticf20 ~]$ sudo mount /dev/demo-vg/demo-lv /mnt/lvdemo
[ian@atticf20 ~]$ sudo df -h /mnt/lvdemo/
Filesystem                    Size  Used Avail Use% Mounted on
/dev/mapper/demo--vg-demo--lv  197G   60M  187G   1% /mnt/lvdemo
```

Note:

- Filesystem commands such as `mkfs` and `mount` access the LV using a name like /dev/<vg-name>/<lv-name>.
- 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 unmount any mounted filesystems that are on it and then deactivate it using the `lvchange` command before removing the drive from the system.

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

The Master Boot Record (MBR) 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 510B of the MBR on /dev/sda, then using the `tail` command to select only the last 64B of the record, which you then display in hexadecimal.

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

```
[root@atticf20 ~]# dd if=/dev/sda bs=510 count=1 2>/dev/null|tail -c 64 |hexdump -C
00000000  00 01 01 00 83 fe 3f 7e  3f 00 00 00 80 21 1f 00  |......?~?....!..|
00000010  00 00 01 7f 82 fe ff ff  bf 21 1f 00 3b 8b 38 01  |.........!..;.8.|
00000020  00 fe ff ff 83 fe ff ff  00 70 85 4a 00 10 00 00  |.........p.J....|
00000030  00 fe ff ff 05 fe ff ff  37 ad 57 01 8a c1 2d 49  |........7.W...-I|
00000040
```

Notice that none of these records has a status of 80h indicating a bootable partition. The partition types are 83h (Linux), 82h (Linux swap), 83h (Linux), and 05h (Extended), so the disk has three primary partitions and one extended partition. All of the CHS values except the first 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 80 21 1f 00 represents 001f2180h, which is 2040192 in decimal. So the first partition starts at sector 63 (3fh) and extends for 2040192 () sectors. Thus, `fdisk` would show an ending sector of 63+2040255-1=2040254.

In the example, the last line (partition type 05h) describes the extended partition, which starts at sector 0157ad37h or 22523191 decimal. You'll need that number in just a moment when you look at logical partitions.

Notes:

- The partition table in the MBR does not contain any information about logical partitions, other than a record describing the extended partition container. You'll see how those are found shortly.
- 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 CHS computations won't be covered in this short introduction.

Listing 9. Fdisk output for /dev/sda

```
[ian@atticf20 ~]$ sudo fdisk -l /dev/sda
Disk /dev/sda: 596.2 GiB, 640135028736 bytes, 1250263728 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
Disklabel type: dos
Disk identifier: 0x00064a1a

Device     Boot      Start        End    Sectors   Size Id Type
/dev/sda1               63    2040254    2040192 996.2M 83 Linux
/dev/sda2          2040255   22523129   20482875   9.8G 82 Linux swap / Solaris
/dev/sda3       1250258944 1250263039       4096     2M 83 Linux
/dev/sda4         22523191 1250258624 1227735434 585.4G  5 Extended
/dev/sda5  *      22523193  167397299  144874107  69.1G 83 Linux
/dev/sda6        167397363  310761359  143363997  68.4G 83 Linux
/dev/sda7        310761423  455442749  144681327    69G 83 Linux
/dev/sda8        455442813  600092009  144649197    69G 83 Linux
/dev/sda9        600092073  744436034  144343962  68.8G 83 Linux
/dev/sda10       744436098  918439935  174003838    83G 83 Linux
/dev/sda11       918441984 1079044095  160602112  76.6G 83 Linux
/dev/sda12      1079053983 1250258624  171204642  81.7G 83 Linux

Partition table entries are not in disk order.
```

And indeed /dev/sda1 ends at 2040254.

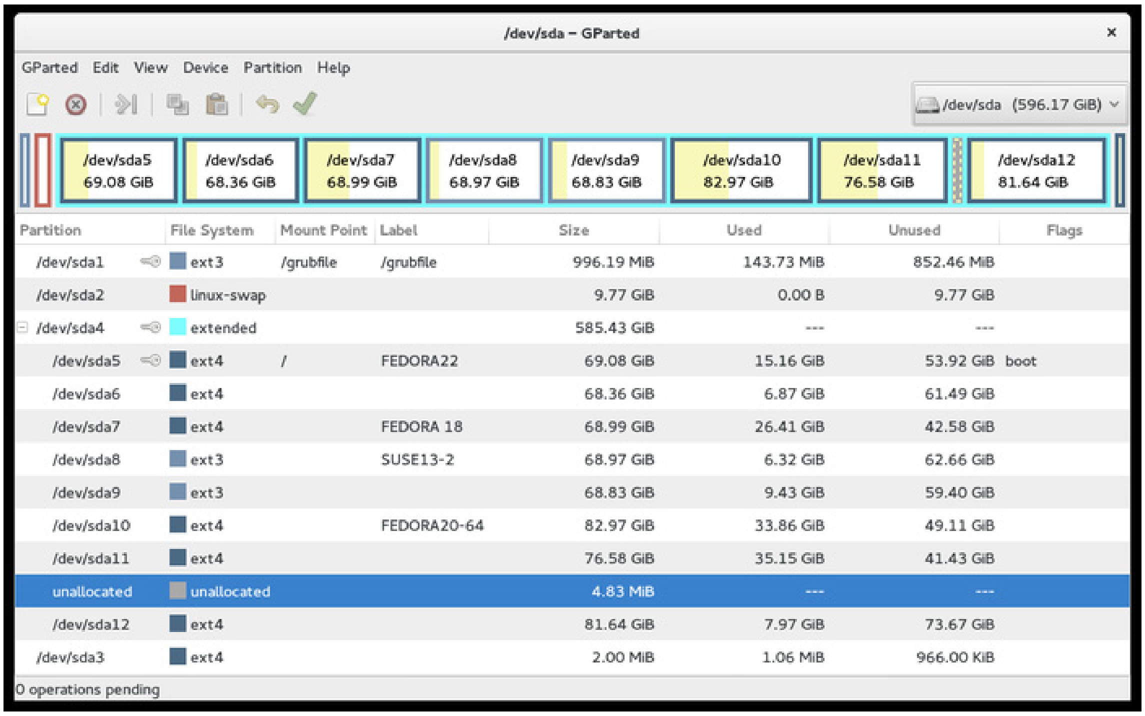
Note the warning that the partition table entries are not in order. This is because /dev/sda3 is a small primary partition that was just added to the end of the disk to illustrate this point. This is not an error, although the `fdisk` program and some other tools have options to rewrite the partition 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/sda2, and /dev/sda4) and one out-of-order 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 2 shows the partitions on the USB drive that contain our LVM partitions. Note that these are flagged as LVM and /dev/sda6 has the boot flag set.

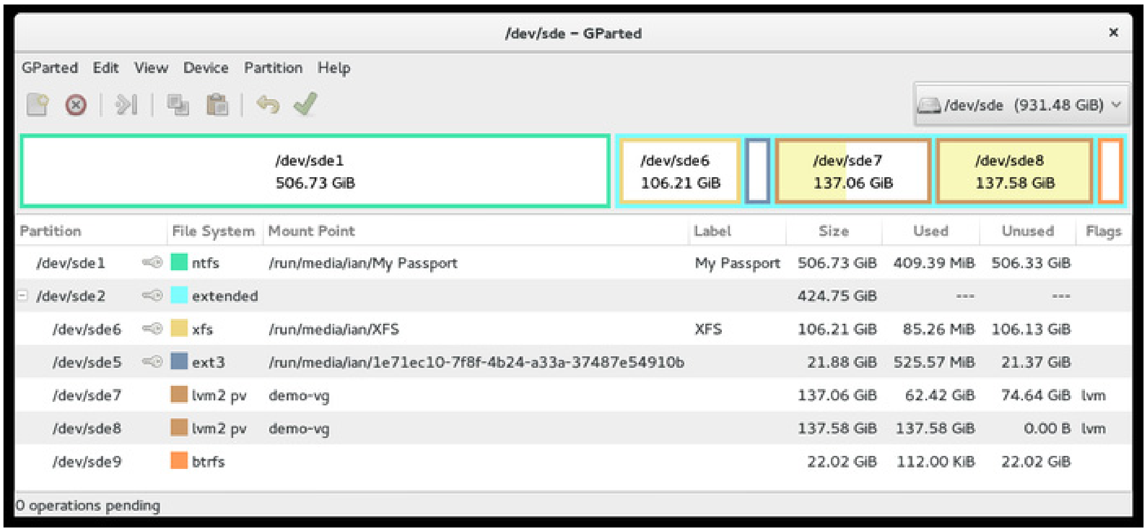
Figure 1. Using gparted to display /dev/sda with several partition types



Partition	File System	Mount Point	Label	Size	Used	Unused	Flags
/dev/sda1	ext3	/grubfile	/grubfile	996.19 MiB	143.73 MiB	852.46 MiB	
/dev/sda2	linux-swap			9.77 GiB	0.00 B	9.77 GiB	
/dev/sda4	extended			585.43 GiB	---	---	
/dev/sda5	ext4	/	FEDORA22	69.08 GiB	15.16 GiB	53.92 GiB	boot
/dev/sda6	ext4			68.36 GiB	6.87 GiB	61.49 GiB	
/dev/sda7	ext4		FEDORA 18	68.99 GiB	26.41 GiB	42.58 GiB	
/dev/sda8	ext3		SUSE13-2	68.97 GiB	6.32 GiB	62.66 GiB	
/dev/sda9	ext3			68.83 GiB	9.43 GiB	59.40 GiB	
/dev/sda10	ext4		FEDORA20-64	82.97 GiB	33.86 GiB	49.11 GiB	
/dev/sda11	ext4			76.58 GiB	35.15 GiB	41.43 GiB	
unallocated	unallocated			4.83 MiB	---	---	
/dev/sda12	ext4			81.64 GiB	7.97 GiB	73.67 GiB	
/dev/sda3	ext4			2.00 MiB	1.06 MiB	966.00 KiB	

0 operations pending

Figure 2. Using gparted to display /dev/sde with several more partition types



Partition	File System	Mount Point	Label	Size	Used	Unused	Flags
/dev/sde1	ntfs	/run/media/ian/My Passport	My Passport	506.73 GiB	409.39 MiB	506.33 GiB	
/dev/sde2	extended			424.75 GiB	---	---	
/dev/sde6	xfs	/run/media/ian/XFS	XFS	106.21 GiB	85.26 MiB	106.13 GiB	
/dev/sde5	ext3	/run/media/ian/1e71ec10-7f8f-4b24-a33a-37487e54910b		21.88 GiB	525.57 MiB	21.37 GiB	
/dev/sde7	lvm2 pv	demo-vg		137.06 GiB	62.42 GiB	74.64 GiB	lvm
/dev/sde8	lvm2 pv	demo-vg		137.58 GiB	137.58 GiB	0.00 B	lvm
/dev/sde9	btrfs			22.02 GiB	112.00 KiB	22.02 GiB	

0 operations pending

As you saw previously, 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 (EBR) for **each** logical partition. Like the MBR, the EBR is 512B 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. Paste this function definition into your Bash terminal session to try it yourself.

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 of 0157AD37h or 22523191 decimal that you found in the MBR in Listing 8.

Listing 11 shows the output.

Listing 11. The first EBR on /dev/sda

```
[root@atticf20 ~]# showebr /dev/sda 22523191
0000000  80 fe ff ff 83 fe ff ff            2    144874107
0000010  00 fe ff ff 05 fe ff ff    144874109    143364060
```

The first entry (/dev/sda5) in the EBR partition table shows that the partition (/dev/sda5) starts at offset 2 from this EBR and the partition contains 144874107 sectors, so the partition ends at sector 22523191 + 2 + 144874107 -1 = 167397299. The second entry (/dev/sda6) shows the next EBR starting at offset 144874109 from the **first** EBR in the extended partition or sector 22523191 + 144874109 = 167397300. The count is the count of sectors starting at the **next** EBR to the end of the next logical partition, or 143364060, so the last sector is 167397300 + 143364060 -1 = 310761359.

Listing 12 shows the partition information for the extended and logical partitions on /dev/sda. Note that the sector count for /dev/sda6 is 143363997, which is 63 sectors smaller than the 143364060 you just saw. The start at 167397363 rather than the 167397300 that you just calculated is also off by 63 sectors. You'll see in a moment that the 63 sector discrepancy is the offset that you will see in the NEXT EBR as you walk the EBR chain.

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

```
Device     Boot      Start        End    Sectors   Size Id Type
/dev/sda4         22523191 1250258624 1227735434 585.4G  5 Extended
/dev/sda5  *      22523193  167397299  144874107  69.1G 83 Linux
/dev/sda6        167397363  310761359  143363997  68.4G 83 Linux
/dev/sda7        310761423  455442749  144681327    69G 83 Linux
/dev/sda8        455442813  600092009  144649197    69G 83 Linux
/dev/sda9        600092073  744436034  144343962  68.8G 83 Linux
/dev/sda10       744436098  918439935  174003838    83G 83 Linux
/dev/sda11       918441984 1079044095  160602112  76.6G 83 Linux
/dev/sda12      1079053983 1250258624  171204642  81.7G 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@atticf20 ~]# showebr /dev/sda 22523191
0000000  80 fe ff ff 83 fe ff ff            2    144874107
0000010  00 fe ff ff 05 fe ff ff    144874109    143364060
[root@atticf20 ~]# showebr /dev/sda $(( 22523191 + 144874109 ))
0000000  00 fe ff ff 83 fe ff ff           63    143363997
0000010  00 fe ff ff 05 fe ff ff    288238169    144681390
[root@atticf20 ~]# showebr /dev/sda $(( 22523191 + 288238169 ))
0000000  00 fe ff ff 83 fe ff ff           63    144681327
0000010  00 fe ff ff 05 fe ff ff    432919559    144649260
[root@atticf20 ~]# showebr /dev/sda $(( 22523191 + 432919559 ))
0000000  00 fe ff ff 83 fe ff ff           63    144649197
0000010  00 fe ff ff 05 fe ff ff    577568819    144344025
[root@atticf20 ~]# showebr /dev/sda $(( 22523191 + 577568819 ))
0000000  00 fe ff ff 83 fe ff ff           63    144343962
0000010  00 fe ff ff 05 fe ff ff    721912844    174003901
[root@atticf20 ~]# showebr /dev/sda $(( 22523191 + 721912844 ))
0000000  00 fe ff ff 83 fe ff ff           63    174003838
0000010  00 fe ff ff 05 fe ff ff    895916745    160604160
[root@atticf20 ~]# showebr /dev/sda $(( 22523191 + 895916745 )) #/dev/sda11
0000000  00 fe ff ff 83 fe ff ff         2048    160602112
0000010  00 fe ff ff 05 fe ff ff   1056530729    171204705
[root@atticf20 ~]# showebr /dev/sda $(( 22523191 + 1056530729 ))
0000000  00 fe ff ff 83 fe ff ff           63    171204642
0000010  00 00 00 00 00 00 00 00            0            0
```

You now see that /dev/sda6 does indeed have an offset of 63 sectors, accounting for the discrepancy just noted. Most other partitions also have an offset of 63 in this example. This was traditional in most Windows and Linux systems. With the advent of advanced format disks and larger sectors, and also solid state drives (SSDs) where partition alignment is more important for performance, you now have the ability to align partitions to specific boundaries. Note that /dev/sda11 has an offset of 2048 sectors

You might have noticed in Figure 1 that there is a small gap between /dev/sda11 and /dev/sda12. With what you know now, you should be able to calculate the size of the gap using the extended partition start and the values from the EBR for /dev/sda11.

GUID Partition Table (GPT)

I mentioned earlier that the MBR layout limits disks with 512B sectors to 2TB in size. With the advent of disks larger than 2TB, a newer layout called GUID Partition Table (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 also support booting from GPT partitions using either the traditional PC BIOS or the newer Extensible Firmware Interface (EFI). Recent 64-bit versions of Windows also support booting from GPT partitions using EFI and Windows 8 requires this.

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 SSD primary disk (/dev/sda) on a Window 8.1 laptop and a 32GB USB flash drive (/dev/sdb) containing the necessary partitions for the Ubuntu 15.04 system used in these examples. Note the EFI system partition (/dev/sda2) which is where the UEFI boot loader looks for files to boot.

Listing 14. Output from gdisk for /dev/sdc

```
ian@yoga-u15:~$ sudo gdisk -l /dev/sda
GPT fdisk (gdisk) version 0.8.10

The protective MBR's 0xEE partition is oversized! Auto-repairing.

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

Found valid GPT with protective MBR; using GPT.
Disk /dev/sda: 1000215216 sectors, 476.9 GiB
Logical sector size: 512 bytes
Disk identifier (GUID): 2D3D303D-0161-4AB2-A713-41DD1A88A647
Partition table holds up to 128 entries
First usable sector is 34, last usable sector is 1000215182
Partitions will be aligned on 2048-sector boundaries
Total free space is 2669 sectors (1.3 MiB)

Number  Start (sector)    End (sector)  Size       Code  Name
   1            2048         2050047   1000.0 MiB  2700  Basic data partition
   2         2050048         2582527   260.0 MiB   EF00  EFI system partition
   3         2582528         4630527   1000.0 MiB  ED01  Basic data partition
   4         4630528         4892671   128.0 MiB   0C01  Microsoft reserved ...
   5         4892672       917575679   435.2 GiB   0700  Basic data partition
   6       917575680       970004479   25.0 GiB    0700  Basic data partition
   7       970004480      1000214527   14.4 GiB    2700  Basic data partition
ian@yoga-u15:~$ sudo gdisk -l /dev/sdb
GPT fdisk (gdisk) version 0.8.10
```

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

Found valid GPT with protective MBR; using GPT.
Disk /dev/sdb: 62685184 sectors, 29.9 GiB
Logical sector size: 512 bytes
Disk identifier (GUID): FEE4E37D-AE90-43A1-A6A7-97ADA4618384
Partition table holds up to 128 entries
First usable sector is 34, last usable sector is 62685150
Partitions will be aligned on 2048-sector boundaries
Total free space is 4029 sectors (2.0 MiB)

Number  Start (sector)    End (sector)  Size       Code  Name
   1            2048          616447   300.0 MiB   EF02
   2          616448         8808447   3.9 GiB     8200
   3         8808448        62683135   25.7 GiB    8300
```

You can find more information on GPT in the tutorial "Make the most of large drives with GPT and Linux" (see Resources).

LVM

I mentioned previously 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, I will not further go into the on-disk data structures used by LVM.

Allocating disk space

As mentioned earlier, a Linux filesystem is a single large tree rooted at /. You will at least want the filesystems described in the Filesystem Hierarchy Standard. Refer back to Table 1 for a list of these or see Resources. 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, UEFI, 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 can share static files such as executable program files. Dynamic files such as user home directories or mail spool files can 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.

Finally, you might need to consider partitions needed for systems that have UEFI firmware rather than BIOS and whether you should use GPT or MBR formatting of your disks.

Making choices

Let's assume that you are setting up a system that has at least one hard drive, and you want to boot from the hard drive. (This tutorial 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 might 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 might 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 5GB or more of disk space plus space for user needs. Some larger development tools can require several gigabytes each. I usually allocate somewhere between 40GB and 60GB per operating system, and I leave the rest of my disk free to load other distributions. The 32GB USB flash drive has 3,9GB used with a basic Ubuntu 15.04 graphical workstation install and no additional tools.

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 can 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 might be on separate partitions, or network mounted, according to your installation needs and preferences.

This concludes your introduction to hard disk layout.

Resources

Learn

- Use the developerWorks roadmap for LPIC-1 to find the developerWorks tutorials to help you study for LPIC-1 certification based on the LPI Version 4.0 April 2015 objectives.
- At the Linux Professional Institute website, find detailed objectives, task lists, and sample questions for the certifications. In particular, see:
  - The LPIC-1: Linux Server Professional Certification program details
  - LPIC-1 exam 101 objectives
  - LPIC-1 exam 102 objectives

  Always refer to the Linux Professional Institute website for the latest objectives.
- Learn more about FHS at the Filesystem Hierarchy Standard home page.
- Learn more about GUID Partition Table s (GPT) in the developerWorks tutorial "Make the most of large drives with GPT and Linux" (developerWorks, July 2012).
- Read more about large sector disks at Transition to Advanced Format 4K Sector Hard Drives.
- See the Microsoft Technet tutorial Master Boot Record for additional detail on master and extended boot records.
- 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.
- Stay current with developerWorks technical events and webcasts focused on a variety of IBM products and IT industry topics.
- 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.

Discuss

- 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 is a freelance Linux writer. He retired from IBM at the Research Triangle Park, NC. Ian joined IBM in Canberra, Australia, as a systems engineer in 1973, and has worked in Montreal, Canada, and RTP, NC in both systems engineering and software development. He has been using, developing on, and writing about Linux since the late 1990s. 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. He enjoys orienteering and likes to travel.