Linux Storage Introduction

Paul Cobbaut

Linux Storage Introduction

Paul Cobbaut

lt-2.1

Publication date Mon 21 Apr 2014 08:07:52 PM CEST

Table of Contents

List of Tables

Chapter 1. disk devices

This chapter teaches you how to locate and recognise **hard disk devices**. This prepares you for the next chapter, where we put **partitions** on these devices.

1.1. terminology

1.1.1. platter, head, track, cylinder, sector

Data is commonly stored on magnetic or optical **disk platters**. The platters are rotated (at high speeds). Data is read by **heads**, which are very close to the surface of the platter, without touching it! The heads are mounted on an arm (sometimes called a comb or a fork).

Data is written in concentric circles called **tracks**. Track zero is (usually) on the outside. The time it takes to position the head over a certain track is called the **seek time**. Often the platters are stacked on top of each other, hence the set of tracks accessible at a certain position of the comb forms a **cylinder**. Tracks are divided into 512 byte **sectors**, with more unused space (**gap**) between the sectors on the outside of the platter.

When you break down the advertised **access time** of a hard drive, you will notice that most of that time is taken by movement of the heads (about 65%) and **rotational latency** (about 30%).

1.1.2. ide or scsi

Actually, the title should be **ata** or **scsi**, since ide is an ata compatible device. Most desktops use **ata devices**, most servers use **scsi**.

1.1.3. ata

An **ata controller** allows two devices per bus, one **master** and one **slave**. Unless your controller and devices support **cable select**, you have to set this manually with jumpers.

With the introduction of **sata** (serial ata), the original ata was renamed to **parallel ata**. Optical drives often use **atapi**, which is an ATA interface using the SCSI communication protocol.

1.1.4. scsi

A **scsi controller** allows more than two devices. When using **SCSI (small computer system interface)**, each device gets a unique **scsi id**. The **scsi controller** also needs a **scsi id**, do not use this id for a scsi-attached device.

Older 8-bit SCSI is now called **narrow**, whereas 16-bit is **wide**. When the bus speeds was doubled to 10Mhz, this was known as **fast SCSI**. Doubling to 20Mhz made it **ultra SCSI**. Take a look at http://en.wikipedia.org/wiki/SCSI for more SCSI standards.

1.1.5. block device

Random access hard disk devices have an abstraction layer called **block device** to enable formatting in fixed-size (usually 512 bytes) blocks. Blocks can be accessed independent of access to other blocks.

```
[root@centos65 ~]# lsblk
NAME                          MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sda                             8:0    0   40G  0 disk
--sda1                          8:1    0  500M  0 part /boot
--sda2                          8:2    0 39.5G  0 part
  --VolGroup-lv_root (dm-0) 253:0    0 38.6G  0 lvm  /
  --VolGroup-lv_swap (dm-1) 253:1    0  928M  0 lvm  [SWAP]
sdb                             8:16   0   72G  0 disk
sdc                             8:32   0  144G  0 disk
```

A block device has the letter b to denote the file type in the output of **ls -l**.

```
[root@centos65 ~]# ls -l /dev/sd*
brw-rw----. 1 root disk 8,  0 Apr 19 10:12 /dev/sda
brw-rw----. 1 root disk 8,  1 Apr 19 10:12 /dev/sda1
brw-rw----. 1 root disk 8,  2 Apr 19 10:12 /dev/sda2
brw-rw----. 1 root disk 8, 16 Apr 19 10:12 /dev/sdb
brw-rw----. 1 root disk 8, 32 Apr 19 10:12 /dev/sdc
```

Note that a **character device** is a constant stream of characters, being denoted by a c in **ls -l**.

Note also that the **ISO 9660** standard for cdrom uses a **2048 byte** block size.

Old hard disks (and floppy disks) use **cylinder-head-sector** addressing to access a sector on the disk. Most current disks use **LBA (Logical Block Addressing)**.

1.1.6. solid state drive

A **solid state drive** or **ssd** is a block device without moving parts. It is comparable to **flash memory**. An **ssd** is more expensive than a hard disk, but it typically has a much faster access time.

1.2. device naming

1.2.1. ata (ide) device naming

All **ata** drives on your system will start with **/dev/hd** followed by a unit letter. The master hdd on the first **ata controller** is /dev/hda, the slave is /dev/hdb. For the second controller, the names of the devices are /dev/hdc and /dev/hdd.

Table 1.1. ide device naming

controller	connection	device name
ide0	master	/dev/hda
	slave	/dev/hdb
ide1	master	/dev/hdc
	slave	/dev/hdd

It is possible to have only **/dev/hda** and **/dev/hdd**. The first one is a single ata hard disk, the second one is the cdrom (by default configured as slave).

1.2.2. scsi device naming

scsi drives follow a similar scheme, but all start with **/dev/sd**. When you run out of letters (after /dev/sdz), you can continue with /dev/sdaa and /dev/sdab and so on. (We will see later on that **lvm** volumes are commonly seen as /dev/md0, /dev/md1 etc.)

Below a **sample** of how scsi devices on a linux can be named. Adding a scsi disk or raid controller with a lower scsi address will change the naming scheme (shifting the higher scsi addresses one letter further in the alphabet).

Table 1.2. scsi device naming

device	scsi id	device name
disk 0	0	/dev/sda
disk 1	1	/dev/sdb
raid controller 0	5	/dev/sdc
raid controller 1	6	/dev/sdd

A modern Linux system will use **/dev/sd*** for scsi and sata devices, and also for sd-cards, usb-sticks, (legacy) ATA/IDE devices and solid state drives.

1.3. discovering disk devices

1.3.1. fdisk

You can start by using **/sbin/fdisk** to find out what kind of disks are seen by the kernel. Below the result on old Debian desktop, with two **ata-ide disks** present.

```
root@barry:~# fdisk -l | grep Disk
Disk /dev/hda: 60.0 GB, 60022480896 bytes
Disk /dev/hdb: 81.9 GB, 81964302336 bytes
```

And here an example of **sata and scsi disks** on a server with CentOS. Remember that **sata** disks are also presented to you with the **scsi** /dev/sd* notation.

```
[root@centos65 ~]# fdisk -l | grep 'Disk /dev/sd'
Disk /dev/sda: 42.9 GB, 42949672960 bytes
Disk /dev/sdb: 77.3 GB, 77309411328 bytes
Disk /dev/sdc: 154.6 GB, 154618822656 bytes
Disk /dev/sdd: 154.6 GB, 154618822656 bytes
```

Here is an overview of disks on a RHEL4u3 server with two real 72GB **scsi disks**. This server is attached to a **NAS** with four **NAS disks** of half a terabyte. On the NAS disks, four LVM (/dev/mdx) software RAID devices are configured.

```
[root@tsvtl1 ~]# fdisk -l | grep Disk
Disk /dev/sda: 73.4 GB, 73407488000 bytes
Disk /dev/sdb: 73.4 GB, 73407488000 bytes
Disk /dev/sdc: 499.0 GB, 499036192768 bytes
Disk /dev/sdd: 499.0 GB, 499036192768 bytes
Disk /dev/sde: 499.0 GB, 499036192768 bytes
Disk /dev/sdf: 499.0 GB, 499036192768 bytes
Disk /dev/md0: 271 MB, 271319040 bytes
Disk /dev/md2: 21.4 GB, 21476081664 bytes
Disk /dev/md3: 21.4 GB, 21467889664 bytes
Disk /dev/md1: 21.4 GB, 21476081664 bytes
```

You can also use **fdisk** to obtain information about one specific hard disk device.

```
[root@centos65 ~]# fdisk -l /dev/sdc

Disk /dev/sdc: 154.6 GB, 154618822656 bytes
255 heads, 63 sectors/track, 18798 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
Disk identifier: 0x00000000
```

Later we will use fdisk to do dangerous stuff like creating and deleting partitions.

1.3.2. dmesg

Kernel boot messages can be seen after boot with **dmesg**. Since hard disk devices are detected by the kernel during boot, you can also use dmesg to find information about disk devices.

```
[root@centos65 ~]# dmesg | grep 'sd[a-z]' | head
sd 0:0:0:0: [sda] 83886080 512-byte logical blocks: (42.9 GB/40.0 GiB)
sd 0:0:0:0: [sda] Write Protect is off
sd 0:0:0:0: [sda] Mode Sense: 00 3a 00 00
sd 0:0:0:0: [sda] Write cache: enabled, read cache: enabled, doesn't support \
DPO or FUA
sda: sda1 sda2
sd 0:0:0:0: [sda] Attached SCSI disk
sd 3:0:0:0: [sdb] 150994944 512-byte logical blocks: (77.3 GB/72.0 GiB)
sd 3:0:0:0: [sdb] Write Protect is off
sd 3:0:0:0: [sdb] Mode Sense: 00 3a 00 00
sd 3:0:0:0: [sdb] Write cache: enabled, read cache: enabled, doesn't support \
DPO or FUA
```

Here is another example of **dmesg** on a computer with a 200GB ata disk.

```
paul@barry:~$ dmesg | grep -i "ata disk"
[    2.624149] hda: ST360021A, ATA DISK drive
[    2.904150] hdb: Maxtor 6Y080L0, ATA DISK drive
[    3.472148] hdd: WDC WD2000BB-98DWA0, ATA DISK drive
```

Third and last example of **dmesg** running on RHEL5.3.

```
root@rhel53 ~# dmesg | grep -i "scsi disk"
sd 0:0:2:0: Attached scsi disk sda
sd 0:0:3:0: Attached scsi disk sdb
sd 0:0:6:0: Attached scsi disk sdc
```

1.3.3. /sbin/lshw

The **lshw** tool will **list hardware**. With the right options **lshw** can show a lot of information about disks (and partitions).

Below a truncated screenshot on Debian 6:

```
root@debian6~# lshw -class volume | grep -A1 -B2 scsi
       description: Linux raid autodetect partition
       physical id: 1
       bus info: scsi@1:0.0.0,1
       logical name: /dev/sdb1
--
       description: Linux raid autodetect partition
       physical id: 1
       bus info: scsi@2:0.0.0,1
       logical name: /dev/sdc1
--
       description: Linux raid autodetect partition
       physical id: 1
       bus info: scsi@3:0.0.0,1
       logical name: /dev/sdd1
--
       description: Linux raid autodetect partition
       physical id: 1
       bus info: scsi@4:0.0.0,1
       logical name: /dev/sde1
--
       vendor: Linux
       physical id: 1
       bus info: scsi@0:0.0.0,1
       logical name: /dev/sda1
--
       vendor: Linux
       physical id: 2
       bus info: scsi@0:0.0.0,2
       logical name: /dev/sda2
--
       description: Extended partition
       physical id: 3
       bus info: scsi@0:0.0.0,3
       logical name: /dev/sda3
```

Redhat and CentOS do not have this tool (unless you add a repository).

1.3.4. /sbin/lsscsi

The **lsscsi** command provides a nice readable output of all scsi (and scsi emulated devices). This first screenshot shows **lsscsi** on a SPARC system.

```
root@shaka:~# lsscsi
[0:0:0:0]    disk    Adaptec  RAID5            V1.0  /dev/sda
[1:0:0:0]    disk    SEAGATE  ST336605FSUN36G  0438  /dev/sdb
root@shaka:~#
```

Below a screenshot of **lsscsi** on a QNAP NAS (which has four 750GB disks and boots from a usb stick).

```
lroot@debian6~# lsscsi
[0:0:0:0]    disk    SanDisk  Cruzer Edge      1.19  /dev/sda
[1:0:0:0]    disk    ATA      ST3750330AS      SD04  /dev/sdb
[2:0:0:0]    disk    ATA      ST3750330AS      SD04  /dev/sdc
[3:0:0:0]    disk    ATA      ST3750330AS      SD04  /dev/sdd
[4:0:0:0]    disk    ATA      ST3750330AS      SD04  /dev/sde
```

This screenshot shows the classic output of **lsscsi**.

```
root@debian6~# lsscsi -c
Attached devices:
Host: scsi0 Channel: 00 Target: 00 Lun: 00
  Vendor: SanDisk  Model: Cruzer Edge      Rev: 1.19
  Type:   Direct-Access                    ANSI SCSI revision: 02
Host: scsi1 Channel: 00 Target: 00 Lun: 00
  Vendor: ATA      Model: ST3750330AS      Rev: SD04
  Type:   Direct-Access                    ANSI SCSI revision: 05
Host: scsi2 Channel: 00 Target: 00 Lun: 00
  Vendor: ATA      Model: ST3750330AS      Rev: SD04
  Type:   Direct-Access                    ANSI SCSI revision: 05
Host: scsi3 Channel: 00 Target: 00 Lun: 00
  Vendor: ATA      Model: ST3750330AS      Rev: SD04
  Type:   Direct-Access                    ANSI SCSI revision: 05
Host: scsi4 Channel: 00 Target: 00 Lun: 00
  Vendor: ATA      Model: ST3750330AS      Rev: SD04
  Type:   Direct-Access                    ANSI SCSI revision: 05
```

1.3.5. /proc/scsi/scsi

Another way to locate **scsi** (or sd) devices is via **/proc/scsi/scsi**.

This screenshot is from a **sparc** computer with adaptec RAID5.

```
root@shaka:~# cat /proc/scsi/scsi
Attached devices:
Host: scsi0 Channel: 00 Id: 00 Lun: 00
  Vendor: Adaptec  Model: RAID5           Rev: V1.0
  Type:   Direct-Access                   ANSI SCSI revision: 02
Host: scsi1 Channel: 00 Id: 00 Lun: 00
  Vendor: SEAGATE  Model: ST336605FSUN36G  Rev: 0438
  Type:   Direct-Access                   ANSI SCSI revision: 03
root@shaka:~#
```

Here we run **cat /proc/scsi/scsi** on the QNAP from above (with Debian Linux).

```
root@debian6~# cat /proc/scsi/scsi
Attached devices:
Host: scsi0 Channel: 00 Id: 00 Lun: 00
  Vendor: SanDisk  Model: Cruzer Edge      Rev: 1.19
  Type:   Direct-Access                    ANSI  SCSI revision: 02
Host: scsi1 Channel: 00 Id: 00 Lun: 00
  Vendor: ATA      Model: ST3750330AS      Rev: SD04
  Type:   Direct-Access                    ANSI  SCSI revision: 05
Host: scsi2 Channel: 00 Id: 00 Lun: 00
  Vendor: ATA      Model: ST3750330AS      Rev: SD04
  Type:   Direct-Access                    ANSI  SCSI revision: 05
Host: scsi3 Channel: 00 Id: 00 Lun: 00
  Vendor: ATA      Model: ST3750330AS      Rev: SD04
  Type:   Direct-Access                    ANSI  SCSI revision: 05
Host: scsi4 Channel: 00 Id: 00 Lun: 00
  Vendor: ATA      Model: ST3750330AS      Rev: SD04
  Type:   Direct-Access                    ANSI  SCSI revision: 05
```

Redhat and CentOS also have this command (after a **yum install lsscsi**).

```
[root@centos65 ~]# cat /proc/scsi/scsi
Attached devices:
Host: scsi0 Channel: 00 Id: 00 Lun: 00
  Vendor: ATA      Model: VBOX HARDDISK    Rev: 1.0
  Type:   Direct-Access                    ANSI  SCSI revision: 05
Host: scsi3 Channel: 00 Id: 00 Lun: 00
  Vendor: ATA      Model: VBOX HARDDISK    Rev: 1.0
  Type:   Direct-Access                    ANSI  SCSI revision: 05
Host: scsi4 Channel: 00 Id: 00 Lun: 00
  Vendor: ATA      Model: VBOX HARDDISK    Rev: 1.0
  Type:   Direct-Access                    ANSI  SCSI revision: 05
```

1.4. erasing a hard disk

Before selling your old hard disk on the internet, it may be a good idea to erase it. By simply repartitioning, or by using the Microsoft Windows format utility, or even after an **mkfs** command, some people will still be able to read most of the data on the disk.

```
root@debian6~# aptitude search foremost autopsy sleuthkit | tr -s ' '
p autopsy - graphical interface to SleuthKit
p foremost - Forensics application to recover data
p sleuthkit - collection of tools for forensics analysis
```

Although technically the **/sbin/badblocks** tool is meant to look for bad blocks, you can use it to completely erase all data from a disk. Since this is really writing to every sector of the disk, it can take a long time!

```
root@RHELv4u2:~# badblocks -ws /dev/sdb
Testing with pattern 0xaa: done
Reading and comparing: done
Testing with pattern 0x55: done
Reading and comparing: done
Testing with pattern 0xff: done
Reading and comparing: done
Testing with pattern 0x00: done
Reading and comparing: done
```

The previous screenshot overwrites every sector of the disk **four times**. Erasing **once** with a tool like **dd** is enough to destroy all data.

Warning, this screenshot shows how to permanently destroy all data on a block device.

```
[root@rhel65 ~]# dd if=/dev/zero of=/dev/sdb
```

1.5. advanced hard disk settings

Tweaking of hard disk settings (dma, gap, ...) are not covered in this course. Several tools exists, **hdparm** and **sdparm** are two of them.

hdparm can be used to display or set information and parameters about an ATA (or SATA) hard disk device. The -i and -I options will give you even more information about the physical properties of the device.

```
root@laika:~# hdparm /dev/sdb

/dev/sdb:
 IO_support   =  0 (default 16-bit)
 readonly     =  0 (off)
 readahead    = 256 (on)
 geometry     = 12161/255/63, sectors = 195371568, start = 0
```

Below **hdparm** info about a 200GB IDE disk.

```
root@barry:~# hdparm /dev/hdd

/dev/hdd:
 multcount     =  0 (off)
 IO_support    =  0 (default)
 unmaskirq     =  0 (off)
 using_dma     =  1 (on)
 keepsettings  =  0 (off)
 readonly      =  0 (off)
 readahead     = 256 (on)
 geometry      = 24321/255/63, sectors = 390721968, start = 0
```

Here a screenshot of **sdparm** on Ubuntu 10.10.

```
root@ubu1010:~# aptitude install sdparm
...
root@ubu1010:~# sdparm /dev/sda | head -1
    /dev/sda: ATA       FUJITSU MJA2160B  0081
root@ubu1010:~# man sdparm
```

Use **hdparm** and **sdparm** with care.

1.6. practice: hard disk devices

About this lab: To practice working with hard disks, you will need some hard disks. When there are no physical hard disk available, you can use virtual disks in **vmware** or **VirtualBox**. The teacher will help you in attaching a couple of ATA and/or SCSI disks to a virtual machine. The results of this lab can be used in the next three labs (partitions, file systems, mounting).

It is adviced to attach three 1GB disks and three 2GB disks to the virtual machine. This will allow for some freedom in the practices of this chapter as well as the next chapters (raid, lvm, iSCSI).

1. Use **dmesg** to make a list of hard disk devices detected at boot-up.

2. Use **fdisk** to find the total size of all hard disk devices on your system.

3. Stop a virtual machine, add three virtual 1 gigabyte **scsi** hard disk devices and one virtual 400 megabyte **ide** hard disk device. If possible, also add another virtual 400 megabyte **ide** disk.

4. Use **dmesg** to verify that all the new disks are properly detected at boot-up.

5. Verify that you can see the disk devices in **/dev**.

6. Use **fdisk** (with **grep** and **/dev/null**) to display the total size of the new disks.

7. Use **badblocks** to completely erase one of the smaller hard disks.

8. Look at **/proc/scsi/scsi**.

9. If possible, install **lsscsi**, **lshw** and use them to list the disks.

1.7. solution: hard disk devices

1. Use **dmesg** to make a list of hard disk devices detected at boot-up.

```
Some possible answers...
```

```
dmesg | grep -i disk
```

```
Looking for ATA disks: dmesg | grep hd[abcd]
```

```
Looking for ATA disks: dmesg | grep -i "ata disk"
```

```
Looking for SCSI disks: dmesg | grep sd[a-f]
```

```
Looking for SCSI disks: dmesg | grep -i "scsi disk"
```

2. Use **fdisk** to find the total size of all hard disk devices on your system.

```
fdisk -l
```

3. Stop a virtual machine, add three virtual 1 gigabyte **scsi** hard disk devices and one virtual 400 megabyte **ide** hard disk device. If possible, also add another virtual 400 megabyte **ide** disk.

```
This exercise happens in the settings of vmware or VirtualBox.
```

4. Use **dmesg** to verify that all the new disks are properly detected at boot-up.

```
See 1.
```

5. Verify that you can see the disk devices in **/dev**.

```
SCSI+SATA: ls -l /dev/sd*
```

```
ATA: ls -l /dev/hd*
```

6. Use **fdisk** (with **grep** and **/dev/null**) to display the total size of the new disks.

```
root@rhel53 ~# fdisk -l 2>/dev/null | grep [MGT]B
Disk /dev/hda: 21.4 GB, 21474836480 bytes
Disk /dev/hdb: 1073 MB, 1073741824 bytes
Disk /dev/sda: 2147 MB, 2147483648 bytes
Disk /dev/sdb: 2147 MB, 2147483648 bytes
Disk /dev/sdc: 2147 MB, 2147483648 bytes
```

7. Use **badblocks** to completely erase one of the smaller hard disks.

```
#Verify the device (/dev/sdc??) you want to erase before typing this.
#
root@rhel53 ~# badblocks -ws /dev/sdc
Testing with pattern 0xaa: done
Reading and comparing: done
Testing with pattern 0x55: done
Reading and comparing: done
Testing with pattern 0xff: done
Reading and comparing: done
Testing with pattern 0x00: done
Reading and comparing: done
```

8. Look at **/proc/scsi/scsi**.

```
root@rhel53 ~# cat /proc/scsi/scsi
```

```
Attached devices:
Host: scsi0 Channel: 00 Id: 02 Lun: 00
  Vendor: VBOX     Model: HARDDISK         Rev: 1.0
  Type:   Direct-Access                    ANSI SCSI revision: 05
Host: scsi0 Channel: 00 Id: 03 Lun: 00
  Vendor: VBOX     Model: HARDDISK         Rev: 1.0
  Type:   Direct-Access                    ANSI SCSI revision: 05
Host: scsi0 Channel: 00 Id: 06 Lun: 00
  Vendor: VBOX     Model: HARDDISK         Rev: 1.0
  Type:   Direct-Access                    ANSI SCSI revision: 05
```

9. If possible, install **lsscsi**, **lshw** and use them to list the disks.

```
Debian,Ubuntu: aptitude install lsscsi lshw
```

```
Fedora: yum install lsscsi lshw
```

```
root@rhel53 ~# lsscsi
[0:0:2:0]    disk    VBOX     HARDDISK         1.0   /dev/sda
[0:0:3:0]    disk    VBOX     HARDDISK         1.0   /dev/sdb
[0:0:6:0]    disk    VBOX     HARDDISK         1.0   /dev/sdc
```

Chapter 2. disk partitions

This chapter continues on the **hard disk devices** from the previous one. Here we will put **partitions** on those devices.

This chapter prepares you for the next chapter, where we put **file systems** on our partitions.

2.1. about partitions

2.1.1. primary, extended and logical

Linux requires you to create one or more **partitions**. The next paragraphs will explain how to create and use partitions.

A partition's **geometry** and size is usually defined by a starting and ending cylinder (sometimes by sector). Partitions can be of type **primary** (maximum four), **extended** (maximum one) or **logical** (contained within the extended partition). Each partition has a **type field** that contains a code. This determines the computers operating system or the partitions file system.

Table 2.1. primary, extended and logical partitions

Partition Type	naming
Primary (max 4)	1-4
Extended (max 1)	1-4
Logical	5-

2.1.2. partition naming

We saw before that hard disk devices are named /dev/hdx or /dev/sdx with x depending on the hardware configuration. Next is the partition number, starting the count at 1. Hence the four (possible) primary partitions are numbered 1 to 4. Logical partition counting always starts at 5. Thus /dev/hda2 is the second partition on the first ATA hard disk device, and /dev/hdb5 is the first logical partition on the second ATA hard disk device. Same for SCSI, /dev/sdb3 is the third partition on the second SCSI disk.

Table 2.2. Partition naming

partition	device
/dev/hda1	first primary partition on /dev/hda
/dev/hda2	second primary or extended partition on /dev/hda
/dev/sda5	first logical drive on /dev/sda
/dev/sdb6	second logical on /dev/sdb

2.2. discovering partitions

2.2.1. fdisk -l

In the **fdisk -l** example below you can see that two partitions exist on **/dev/sdb**. The first partition spans 31 cylinders and contains a Linux swap partition. The second partition is much bigger.

```
root@laika:~# fdisk -l /dev/sdb

Disk /dev/sdb: 100.0 GB, 100030242816 bytes
255 heads, 63 sectors/track, 12161 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes

   Device Boot    Start      End     Blocks   Id  System
/dev/sdb1             1       31     248976   82  Linux swap / Solaris
/dev/sdb2            32    12161   97434225   83  Linux
root@laika:~#
```

2.2.2. /proc/partitions

The **/proc/partitions** file contains a table with major and minor number of partitioned devices, their number of blocks and the device name in **/dev**. Verify with **/proc/devices** to link the major number to the proper device.

```
paul@RHELv4u4:~$ cat /proc/partitions
major minor  #blocks  name

   3     0     524288 hda
   3    64     734003 hdb
   8     0    8388608 sda
   8     1     104391 sda1
   8     2    8281507 sda2
   8    16    1048576 sdb
   8    32    1048576 sdc
   8    48    1048576 sdd
 253     0    7176192 dm-0
 253     1    1048576 dm-1
```

The **major** number corresponds to the device type (or driver) and can be found in **/proc/devices**. In this case 3 corresponds to **ide** and 8 to **sd**. The **major** number determines the **device driver** to be used with this device.

The **minor** number is a unique identification of an instance of this device type. The **devices.txt** file in the kernel tree contains a full list of major and minor numbers.

2.2.3. parted and others

You may be interested in alternatives to **fdisk** like **parted**, **cfdisk**, **sfdisk** and **gparted**. This course mainly uses **fdisk** to partition hard disks.

parted is recommended by some Linux distributions for handling storage with **gpt** instead of **mbr**.

Below a screenshot of **parted** on CentOS.

```
[root@centos65 ~]# rpm -q parted
parted-2.1-21.el6.x86_64
[root@centos65 ~]# parted /dev/sda
GNU Parted 2.1
Using /dev/sda
Welcome to GNU Parted! Type 'help' to view a list of commands.
(parted) print
Model: ATA VBOX HARDDISK (scsi)
Disk /dev/sda: 42.9GB
Sector size (logical/physical): 512B/512B
Partition Table: msdos

Number  Start   End     Size    Type     File system  Flags
 1      1049kB  525MB   524MB   primary  ext4         boot
 2      525MB   42.9GB  42.4GB  primary               lvm

(parted)
```

2.3. partitioning new disks

In the example below, we bought a new disk for our system. After the new hardware is properly attached, you can use **fdisk** and **parted** to create the necessary partition(s). This example uses **fdisk**, but there is nothing wrong with using **parted**.

2.3.1. recognising the disk

First, we check with **fdisk -l** whether Linux can see the new disk. Yes it does, the new disk is seen as /dev/sdb, but it does not have any partitions yet.

```
root@RHELv4u2:~# fdisk -l

Disk /dev/sda: 12.8 GB, 12884901888 bytes
255 heads, 63 sectors/track, 1566 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes

Device Boot      Start      End      Blocks   Id  System
/dev/sda1   *       1        13      104391   83  Linux
/dev/sda2          14      1566    12474472+  8e  Linux LVM

Disk /dev/sdb: 1073 MB, 1073741824 bytes
255 heads, 63 sectors/track, 130 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes

Disk /dev/sdb doesn't contain a valid partition table
```

2.3.2. opening the disk with fdisk

Then we create a partition with fdisk on /dev/sdb. First we start the fdisk tool with /dev/sdb as argument. Be very very careful not to partition the wrong disk!!

```
root@RHELv4u2:~# fdisk /dev/sdb
Device contains neither a valid DOS partition table, nor Sun, SGI...
Building a new DOS disklabel. Changes will remain in memory only,
until you decide to write them. After that, of course, the previous
content won't be recoverable.

Warning: invalid flag 0x0000 of partition table 4 will be corrected...
```

2.3.3. empty partition table

Inside the fdisk tool, we can issue the **p** command to see the current disks partition table.

```
Command (m for help): p

Disk /dev/sdb: 1073 MB, 1073741824 bytes
255 heads, 63 sectors/track, 130 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes

Device Boot      Start         End      Blocks   Id  System
```

2.3.4. create a new partition

No partitions exist yet, so we issue **n** to create a new partition. We choose p for primary, 1 for the partition number, 1 for the start cylinder and 14 for the end cylinder.

```
Command (m for help): n
Command action
e   extended
p   primary partition (1-4)
p
Partition number (1-4): 1
First cylinder (1-130, default 1):
Using default value 1
Last cylinder or +size or +sizeM or +sizeK (1-130, default 130): 14
```

We can now issue p again to verify our changes, but they are not yet written to disk. This means we can still cancel this operation! But it looks good, so we use **w** to write the changes to disk, and then quit the fdisk tool.

```
Command (m for help): p

Disk /dev/sdb: 1073 MB, 1073741824 bytes
255 heads, 63 sectors/track, 130 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes

Device Boot      Start         End      Blocks   Id  System
/dev/sdb1               1          14      112423+  83  Linux

Command (m for help): w
The partition table has been altered!

Calling ioctl() to re-read partition table.
Syncing disks.
root@RHELv4u2:~#
```

2.3.5. display the new partition

Let's verify again with **fdisk -l** to make sure reality fits our dreams. Indeed, the screenshot below now shows a partition on /dev/sdb.

```
root@RHELv4u2:~# fdisk -l

Disk /dev/sda: 12.8 GB, 12884901888 bytes
255 heads, 63 sectors/track, 1566 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes

Device Boot      Start         End      Blocks   Id  System
/dev/sda1   *           1          13      104391   83  Linux
/dev/sda2              14        1566    12474472+  8e  Linux LVM

Disk /dev/sdb: 1073 MB, 1073741824 bytes
255 heads, 63 sectors/track, 130 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes

Device Boot      Start         End      Blocks   Id  System
/dev/sdb1               1          14      112423+  83  Linux
root@RHELv4u2:~#
```

2.4. about the partition table

2.4.1. master boot record

The **partition table** information (primary and extended partitions) is written in the **master boot record** or **mbr**. You can use **dd** to copy the mbr to a file.

This example copies the master boot record from the first SCSI hard disk.

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

The same tool can also be used to wipe out all information about partitions on a disk. This example writes zeroes over the master boot record.

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

Or to wipe out the whole partition or disk.

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

2.4.2. partprobe

Don't forget that after restoring a **master boot record** with **dd**, that you need to force the kernel to reread the partition table with **partprobe**. After running **partprobe**, the partitions can be used again.

```
[root@RHEL5 ~]# partprobe
[root@RHEL5 ~]#
```

2.4.3. logical drives

The **partition table** does not contain information about **logical drives**. So the **dd** backup of the **mbr** only works for primary and extended partitions. To backup the partition table including the logical drives, you can use **sfdisk**.

This example shows how to backup all partition and logical drive information to a file.

```
sfdisk -d /dev/sda > parttable.sda.sfdisk
```

The following example copies the **mbr** and all **logical drive** info from /dev/sda to /dev/sdb.

```
sfdisk -d /dev/sda | sfdisk /dev/sdb
```

2.5. practice: partitions

1. Use **fdisk -l** to display existing partitions and sizes.

2. Use **df -h** to display existing partitions and sizes.

3. Compare the output of **fdisk** and **df**.

4. Create a 200MB primary partition on a small disk.

5. Create a 400MB primary partition and two 300MB logical drives on a big disk.

6. Use **df -h** and **fdisk -l** to verify your work.

7. Compare the output again of **fdisk** and **df**. Do both commands display the new partitions ?

8. Create a backup with **dd** of the **mbr** that contains your 200MB primary partition.

9. Take a backup of the **partition table** containing your 400MB primary and 300MB logical drives. Make sure the logical drives are in the backup.

10. (optional) Remove all your partitions with fdisk. Then restore your backups.

2.6. solution: partitions

1. Use **fdisk -l** to display existing partitions and sizes.

```
as root: # fdisk -l
```

2. Use **df -h** to display existing partitions and sizes.

```
df -h
```

3. Compare the output of **fdisk** and **df**.

```
Some partitions will be listed in both outputs (maybe /dev/sda1 or /dev/hda1).
```

4. Create a 200MB primary partition on a small disk.

```
Choose one of the disks you added (this example uses /dev/sdc).
root@rhel53 ~# fdisk /dev/sdc
...
Command (m for help): n
Command action
   e   extended
   p   primary partition (1-4)
p
Partition number (1-4): 1
First cylinder (1-261, default 1): 1
Last cylinder or +size or +sizeM or +sizeK (1-261, default 261): +200m
Command (m for help): w
The partition table has been altered!
Calling ioctl() to re-read partition table.
Syncing disks.
```

5. Create a 400MB primary partition and two 300MB logical drives on a big disk.

```
Choose one of the disks you added (this example uses /dev/sdb)
```

```
fdisk /dev/sdb
```

```
inside fdisk : n p 1 +400m enter --- n e 2 enter enter --- n l +300m (twice)
```

6. Use **df -h** and **fdisk -l** to verify your work.

```
fdisk -l ; df -h
```

7. Compare the output again of **fdisk** and **df**. Do both commands display the new partitions ?

```
The newly created partitions are visible with fdisk.
```

```
But they are not displayed by df.
```

8. Create a backup with **dd** of the **mbr** that contains your 200MB primary partition.

```
dd if=/dev/sdc of=bootsector.sdc.dd count=1 bs=512
```

9. Take a backup of the **partition table** containing your 400MB primary and 300MB logical drives. Make sure the logical drives are in the backup.

```
sfdisk -d /dev/sdb > parttable.sdb.sfdisk
```

Chapter 3. file systems

When you are finished partitioning the hard disk, you can put a **file system** on each partition.

This chapter builds on the **partitions** from the previous chapter, and prepares you for the next one where we will **mount** the filesystems.

3.1. about file systems

A file system is a way of organizing files on your partition. Besides file-based storage, file systems usually include **directories** and **access control**, and contain meta information about files like access times, modification times and file ownership.

The properties (length, character set, ...) of filenames are determined by the file system you choose. Directories are usually implemented as files, you will have to learn how this is implemented! Access control in file systems is tracked by user ownership (and group owner- and membership) in combination with one or more access control lists.

3.1.1. man fs

The manual page about filesystems is accessed by typing **man fs**.

```
[root@rhel65 ~]# man fs
```

3.1.2. /proc/filesystems

The Linux kernel will inform you about currently loaded file system drivers in **/proc/filesystems**.

```
root@rhel53 ~# cat /proc/filesystems  | grep -v nodev
 ext2
 iso9660
 ext3
```

3.1.3. /etc/filesystems

The **/etc/filesystems** file contains a list of autodetected filesystems (in case the **mount** command is used without the **-t** option.

Help for this file is provided by **man mount**.

```
[root@rhel65 ~]# man mount
```

3.2. common file systems

3.2.1. ext2 and ext3

Once the most common Linux file systems is the **ext2** (the second extended) file system. A disadvantage is that file system checks on ext2 can take a long time.

ext2 was being replaced by **ext3** on most Linux machines. They are essentially the same, except for the **journaling** which is only present in ext3.

Journaling means that changes are first written to a journal on the disk. The journal is flushed regularly, writing the changes in the file system. Journaling keeps the file system in a consistent state, so you don't need a file system check after an unclean shutdown or power failure.

3.2.2. creating ext2 and ext3

You can create these file systems with the **/sbin/mkfs** or **/sbin/mke2fs** commands. Use **mke2fs -j** to create an **ext3** file system.

You can convert an ext2 to ext3 with **tune2fs -j**. You can mount an ext3 file system as ext2, but then you lose the journaling. Do not forget to run **mkinitrd** if you are booting from this device.

3.2.3. ext4

The newest incarnation of the ext file system is named **ext4** and is available in the Linux kernel since 2008. **ext4** supports larger files (up to 16 terabyte) and larger file systems than **ext3** (and many more features).

Development started by making **ext3** fully capable for 64-bit. When it turned out the changes were significant, the developers decided to name it **ext4**.

3.2.4. xfs

Redhat Enterprise Linux 7 will have **XFS** as the default file system. This is a highly scalable high-performance file system.

xfs was created for **Irix** and for a couple of years it was also used in **FreeBSD**. It is supported by the Linux kernel, but rarely used in dsitributions outside of the Redhat/CentOS realm.

3.2.5. vfat

The **vfat** file system exists in a couple of forms : **fat12** for floppy disks, **fat16** on **ms-dos**, and **fat32** for larger disks. The Linux **vfat** implementation supports all of these, but vfat lacks a lot of features like security and links. **fat** disks can be read by every operating system, and are used a lot for digital cameras, **usb** sticks and to exchange data between different OS'ses on a home user's computer.

3.2.6. iso 9660

iso 9660 is the standard format for cdroms. Chances are you will encounter this file system also on your hard disk in the form of images of cdroms (often with the .iso extension). The **iso 9660** standard limits filenames to the 8.3 format. The Unix world didn't like this, and thus added the **rock ridge** extensions, which allows for filenames up to 255 characters and Unix-style file-modes, ownership and symbolic links. Another extensions to **iso 9660** is **joliet**, which adds 64 unicode characters to the filename. The **el torito** standard extends **iso 9660** to be able to boot from CD-ROM's.

3.2.7. udf

Most optical media today (including cd's and dvd's) use **udf**, the Universal Disk Format.

3.2.8. swap

All things considered, swap is not a file system. But to use a partition as a **swap partition** it must be formatted and mounted as swap space.

3.2.9. gfs

Linux clusters often use a dedicated cluster filesystem like GFS, GFS2, ClusterFS, ...

3.2.10. and more...

You may encounter **reiserfs** on older Linux systems. Maybe you will see Sun's **zfs** or the open source **btrfs**. This last one requires a chapter on itself.

3.2.11. /proc/filesystems

The **/proc/filesystems** file displays a list of supported file systems. When you mount a file system without explicitly defining one, then mount will first try to probe **/etc/filesystems** and then probe **/proc/filesystems** for all the filesystems without the **nodev** label. If **/etc/filesystems** ends with a line containing only an asterisk (*) then both files are probed.

```
paul@RHELv4u4:~$ cat /proc/filesystems
nodev   sysfs
nodev   rootfs
nodev   bdev
nodev   proc
nodev   sockfs
nodev   binfmt_misc
nodev   usbfs
nodev   usbdevfs
nodev   futexfs
nodev   tmpfs
nodev   pipefs
nodev   eventpollfs
nodev   devpts
        ext2
nodev   ramfs
nodev   hugetlbfs
        iso9660
nodev   relayfs
nodev   mqueue
nodev   selinuxfs
        ext3
nodev   rpc_pipefs
nodev   vmware-hgfs
nodev   autofs
paul@RHELv4u4:~$
```

3.3. putting a file system on a partition

We now have a fresh partition. The system binaries to make file systems can be found with ls.

```
[root@RHEL4b ~]# ls -lS /sbin/mk*
-rwxr-xr-x  3 root root 34832 Apr 24  2006 /sbin/mke2fs
-rwxr-xr-x  3 root root 34832 Apr 24  2006 /sbin/mkfs.ext2
-rwxr-xr-x  3 root root 34832 Apr 24  2006 /sbin/mkfs.ext3
-rwxr-xr-x  3 root root 28484 Oct 13  2004 /sbin/mkdosfs
-rwxr-xr-x  3 root root 28484 Oct 13  2004 /sbin/mkfs.msdos
-rwxr-xr-x  3 root root 28484 Oct 13  2004 /sbin/mkfs.vfat
-rwxr-xr-x  1 root root 20313 Apr 10  2006 /sbin/mkinitrd
-rwxr-x---  1 root root 15444 Oct  5  2004 /sbin/mkzonedb
-rwxr-xr-x  1 root root 15300 May 24  2006 /sbin/mkfs.cramfs
-rwxr-xr-x  1 root root 13036 May 24  2006 /sbin/mkswap
-rwxr-xr-x  1 root root  6912 May 24  2006 /sbin/mkfs
-rwxr-xr-x  1 root root  5905 Aug  3  2004 /sbin/mkbootdisk
[root@RHEL4b ~]#
```

It is time for you to read the manual pages of **mkfs** and **mke2fs**. In the example below, you see the creation of an **ext2 file system** on /dev/sdb1. In real life, you might want to use options like -m0 and -j.

```
root@RHELv4u2:~# mke2fs /dev/sdb1
mke2fs 1.35 (28-Feb-2004)
Filesystem label=
OS type: Linux
Block size=1024 (log=0)
Fragment size=1024 (log=0)
28112 inodes, 112420 blocks
5621 blocks (5.00%) reserved for the super user
First data block=1
Maximum filesystem blocks=67371008
14 block groups
8192 blocks per group, 8192 fragments per group
2008 inodes per group
Superblock backups stored on blocks:
8193, 24577, 40961, 57345, 73729

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

This filesystem will be automatically checked every 37 mounts or
180 days, whichever comes first.  Use tune2fs -c or -i to override.
```

3.4. tuning a file system

You can use **tune2fs** to list and set file system settings. The first screenshot lists the reserved space for root (which is set at five percent).

```
[root@rhel4 ~]# tune2fs -l /dev/sda1 | grep -i "block count"
Block count:              104388
Reserved block count:     5219
[root@rhel4 ~]#
```

This example changes this value to ten percent. You can use tune2fs while the file system is active, even if it is the root file system (as in this example).

```
[root@rhel4 ~]# tune2fs -m10 /dev/sda1
tune2fs 1.35 (28-Feb-2004)
Setting reserved blocks percentage to 10 (10430 blocks)
[root@rhel4 ~]# tune2fs -l /dev/sda1 | grep -i "block count"
Block count:              104388
Reserved block count:     10430
[root@rhel4 ~]#
```

3.5. checking a file system

The **fsck** command is a front end tool used to check a file system for errors.

```
[root@RHEL4b ~]# ls /sbin/*fsck*
/sbin/dosfsck  /sbin/fsck         /sbin/fsck.ext2  /sbin/fsck.msdos
/sbin/e2fsck   /sbin/fsck.cramfs  /sbin/fsck.ext3  /sbin/fsck.vfat
[root@RHEL4b ~]#
```

The last column in **/etc/fstab** is used to determine whether a file system should be checked at boot-up.

```
[paul@RHEL4b ~]$ grep ext /etc/fstab
/dev/VolGroup00/LogVol00   /             ext3    defaults        1 1
LABEL=/boot                /boot         ext3    defaults        1 2
[paul@RHEL4b ~]$
```

Manually checking a mounted file system results in a warning from fsck.

```
[root@RHEL4b ~]# fsck /boot
fsck 1.35 (28-Feb-2004)
e2fsck 1.35 (28-Feb-2004)
/dev/sda1 is mounted.

WARNING!!!  Running e2fsck on a mounted filesystem may cause
SEVERE filesystem damage.

Do you really want to continue (y/n)? no

check aborted.
```

But after unmounting fsck and **e2fsck** can be used to check an ext2 file system.

```
[root@RHEL4b ~]# fsck  /boot
fsck 1.35 (28-Feb-2004)
e2fsck 1.35 (28-Feb-2004)
/boot: clean, 44/26104 files, 17598/104388 blocks
[root@RHEL4b ~]# fsck -p /boot
fsck 1.35 (28-Feb-2004)
/boot: clean, 44/26104 files, 17598/104388 blocks
[root@RHEL4b ~]# e2fsck -p /dev/sda1
/boot: clean, 44/26104 files, 17598/104388 blocks
```

3.6. practice: file systems

1. List the filesystems that are known by your system.
2. Create an **ext2** filesystem on the 200MB partition.
3. Create an **ext3** filesystem on one of the 300MB logical drives.
4. Create an **ext4** on the 400MB partition.
5. Set the reserved space for root on the ext3 filesystem to 0 percent.
6. Verify your work with **fdisk** and **df**.
7. Perform a file system check on all the new file systems.

3.7. solution: file systems

1. List the filesystems that are known by your system.

```
man fs

cat /proc/filesystems

cat /etc/filesystems (not on all Linux distributions)
```

2. Create an **ext2** filesystem on the 200MB partition.

```
mke2fs /dev/sdc1 (replace sdc1 with the correct partition)
```

3. Create an **ext3** filesystem on one of the 300MB logical drives.

```
mke2fs -j /dev/sdb5 (replace sdb5 with the correct partition)
```

4. Create an **ext4** on the 400MB partition.

```
mkfs.ext4 /dev/sdb1 (replace sdb1 with the correct partition)
```

5. Set the reserved space for root on the ext3 filesystem to 0 percent.

```
tune2fs -m 0 /dev/sdb5
```

6. Verify your work with **fdisk** and **df**.

```
mkfs (mke2fs) makes no difference in the output of these commands

The big change is in the next topic: mounting
```

7. Perform a file system check on all the new file systems.

```
fsck /dev/sdb1
fsck /dev/sdc1
fsck /dev/sdb5
```

Chapter 4. mounting

Once you've put a file system on a partition, you can **mount** it. Mounting a file system makes it available for use, usually as a directory. We say **mounting a file system** instead of mounting a partition because we will see later that we can also mount file systems that do not exists on partitions.

On all **Unix** systems, every file and every directory is part of one big file tree. To access a file, you need to know the full path starting from the root directory. When adding a **file system** to your computer, you need to make it available somewhere in the file tree. The directory where you make a file system available is called a **mount point**.

4.1. mounting local file systems

4.1.1. mkdir

This example shows how to create a new **mount point** with **mkdir**.

```
root@RHELv4u2:~# mkdir /home/project42
```

4.1.2. mount

When the **mount point** is created, and a **file system** is present on the partition, then **mount** can **mount** the **file system** on the **mount point directory**.

```
root@RHELv4u2:~# mount -t ext2 /dev/sdb1 /home/project42/
```

Once mounted, the new file system is accessible to users.

4.1.3. /etc/filesystems

Actually the explicit **-t ext2** option to set the file system is not always necessary. The **mount** command is able to automatically detect a lot of file systems.

When mounting a file system without specifying explicitly the file system, then **mount** will first probe **/etc/filesystems**. Mount will skip lines with the **nodev** directive.

```
paul@RHELv4u4:~$ cat /etc/filesystems
ext3
ext2
nodev proc
nodev devpts
iso9660
vfat
hfs
```

4.1.4. /proc/filesystems

When **/etc/filesystems** does not exist, or ends with a single * on the last line, then **mount** will read **/proc/filesystems**.

```
[root@RHEL52 ~]# cat /proc/filesystems | grep -v ^nodev
 ext2
 iso9660
 ext3
```

4.1.5. umount

You can **unmount** a mounted file system using the **umount** command.

```
root@pasha:~# umount /home/reet
```

4.2. displaying mounted file systems

To display all mounted file systems, issue the **mount** command. Or look at the files **/proc/mounts** and **/etc/mtab**.

4.2.1. mount

The simplest and most common way to view all mounts is by issuing the **mount** command without any arguments.

```
root@RHELv4u2:~# mount | grep /dev/sdb
/dev/sdb1 on /home/project42 type ext2 (rw)
```

4.2.2. /proc/mounts

The kernel provides the info in **/proc/mounts** in file form, but **/proc/mounts** does not exist as a file on any hard disk. Looking at **/proc/mounts** is looking at information that comes directly from the kernel.

```
root@RHELv4u2:~# cat /proc/mounts | grep /dev/sdb
/dev/sdb1 /home/project42 ext2 rw 0 0
```

4.2.3. /etc/mtab

The **/etc/mtab** file is not updated by the kernel, but is maintained by the **mount** command. Do not edit **/etc/mtab** manually.

```
root@RHELv4u2:~# cat /etc/mtab | grep /dev/sdb
/dev/sdb1 /home/project42 ext2 rw 0 0
```

4.2.4. df

A more user friendly way to look at mounted file systems is **df**. The **df (diskfree)** command has the added benefit of showing you the free space on each mounted disk. Like a lot of Linux commands, **df** supports the **-h** switch to make the output more **human readable**.

```
root@RHELv4u2:~# df
Filesystem           1K-blocks      Used Available Use% Mounted on
/dev/mapper/VolGroup00-LogVol00
11707972   6366996   4746240  58% /
/dev/sda1              101086    9300    86567  10% /boot
none                   127988       0   127988   0% /dev/shm
/dev/sdb1              108865    1550   101694   2% /home/project42
root@RHELv4u2:~# df -h
Filesystem            Size  Used Avail Use% Mounted on
/dev/mapper/VolGroup00-LogVol00
12G  6.1G  4.6G  58% /
/dev/sda1              99M  9.1M   85M  10% /boot
none                  125M     0  125M   0% /dev/shm
/dev/sdb1             107M  1.6M  100M   2% /home/project42
```

4.2.5. df -h

In the **df -h** example below you can see the size, free space, used gigabytes and percentage and mount point of a partition.

```
root@laika:~# df -h | egrep -e "(sdb2|File)"
Filesystem            Size Used Avail Use% Mounted on
/dev/sdb2              92G   83G  8.6G  91% /media/sdb2
```

4.2.6. du

The **du** command can summarize **disk usage** for files and directories. By using **du** on a mount point you effectively get the disk space used on a file system.

While **du** can go display each subdirectory recursively, the **-s** option will give you a total summary for the parent directory. This option is often used together with **-h**. This means **du -sh** on a mount point gives the total amount used by the file system in that partition.

```
root@debian6~# du -sh /boot /srv/wolf
6.2M /boot
1.1T /srv/wolf
```

4.3. from start to finish

Below is a screenshot that show a summary roadmap starting with detection of the hardware (/dev/sdb) up until mounting on **/mnt**.

```
[root@centos65 ~]# dmesg | grep '\[sdb\]'
sd 3:0:0:0: [sdb] 150994944 512-byte logical blocks: (77.3 GB/72.0 GiB)
sd 3:0:0:0: [sdb] Write Protect is off
sd 3:0:0:0: [sdb] Mode Sense: 00 3a 00 00
sd 3:0:0:0: [sdb] Write cache: enabled, read cache: enabled, doesn't support \
DPO or FUA
sd 3:0:0:0: [sdb] Attached SCSI disk

[root@centos65 ~]# parted /dev/sdb

(parted) mklabel msdos
(parted) mkpart primary ext4 1 77000
(parted) print
Model: ATA VBOX HARDDISK (scsi)
Disk /dev/sdb: 77.3GB
Sector size (logical/physical): 512B/512B
Partition Table: msdos

Number  Start   End     Size    Type     File system  Flags
 1      1049kB  77.0GB  77.0GB  primary

(parted) quit
[root@centos65 ~]# mkfs.ext4 /dev/sdb1
mke2fs 1.41.12 (17-May-2010)
Filesystem label=
OS type: Linux
Block size=4096 (log=2)
Fragment size=4096 (log=2)
Stride=0 blocks, Stripe width=0 blocks
4702208 inodes, 18798592 blocks
939929 blocks (5.00%) reserved for the super user
First data block=0
Maximum filesystem blocks=4294967296
574 block groups
32768 blocks per group, 32768 fragments per group
8192 inodes per group
( output truncated )
...
[root@centos65 ~]# mount /dev/sdb1 /mnt
[root@centos65 ~]# mount | grep mnt
/dev/sdb1 on /mnt type ext4 (rw)
[root@centos65 ~]# df -h | grep mnt
/dev/sdb1              71G  180M   67G   1% /mnt
[root@centos65 ~]# du -sh /mnt
20K     /mnt
[root@centos65 ~]# umount /mnt
```

4.4. permanent mounts

Until now, we performed all mounts manually. This works nice, until the next reboot. Luckily there is a way to tell your computer to automatically mount certain file systems during boot.

4.4.1. /etc/fstab

The file system table located in **/etc/fstab** contains a list of file systems, with an option to automtically mount each of them at boot time.

Below is a sample **/etc/fstab** file.

```
root@RHELv4u2:~# cat /etc/fstab
/dev/VolGroup00/LogVol00 /                ext3    defaults        1 1
LABEL=/boot              /boot            ext3    defaults        1 2
none                     /dev/pts         devpts  gid=5,mode=620  0 0
none                     /dev/shm         tmpfs   defaults        0 0
none                     /proc            proc    defaults        0 0
none                     /sys             sysfs   defaults        0 0
/dev/VolGroup00/LogVol01 swap             swap    defaults        0 0
```

By adding the following line, we can automate the mounting of a file system.

```
/dev/sdb1                /home/project42      ext2    defaults    0 0
```

4.4.2. mount /mountpoint

Adding an entry to **/etc/fstab** has the added advantage that you can simplify the **mount** command. The command in the screenshot below forces **mount** to look for the partition info in **/etc/fstab**.

```
root@rhel65:~# mount /home/project42
```

4.5. securing mounts

File systems can be secured with several **mount options**. Here are some examples.

4.5.1. ro

The **ro** option will mount a file system as read only, preventing anyone from writing.

```
root@rhel53 ~# mount -t ext2 -o ro /dev/hdb1 /home/project42
root@rhel53 ~# touch /home/project42/testwrite
touch: cannot touch `/home/project42/testwrite': Read-only file system
```

4.5.2. noexec

The **noexec** option will prevent the execution of binaries and scripts on the mounted file system.

```
root@rhel53 ~# mount -t ext2 -o noexec /dev/hdb1 /home/project42
root@rhel53 ~# cp /bin/cat /home/project42
root@rhel53 ~# /home/project42/cat /etc/hosts
-bash: /home/project42/cat: Permission denied
root@rhel53 ~# echo echo hello > /home/project42/helloscript
root@rhel53 ~# chmod +x /home/project42/helloscript
root@rhel53 ~# /home/project42/helloscript
-bash: /home/project42/helloscript: Permission denied
```

4.5.3. nosuid

The **nosuid** option will ignore **setuid** bit set binaries on the mounted file system.

Note that you can still set the **setuid** bit on files.

```
root@rhel53 ~# mount -o nosuid /dev/hdb1 /home/project42
root@rhel53 ~# cp /bin/sleep /home/project42/
root@rhel53 ~# chmod 4555 /home/project42/sleep
root@rhel53 ~# ls -l /home/project42/sleep
-r-sr-xr-x 1 root root 19564 Jun 24 17:57 /home/project42/sleep
```

But users cannot exploit the **setuid** feature.

```
root@rhel53 ~# su - paul
[paul@rhel53 ~]$ /home/project42/sleep 500 &
[1] 2876
[paul@rhel53 ~]$ ps -f 2876
UID        PID  PPID  C STIME TTY      STAT   TIME CMD
paul      2876  2853  0 17:58 pts/0    S      0:00 /home/project42/sleep 500
[paul@rhel53 ~]$
```

4.5.4. noacl

To prevent cluttering permissions with **acl's**, use the **noacl** option.

```
root@rhel53 ~# mount -o noacl /dev/hdb1 /home/project42
```

More **mount options** can be found in the manual page of **mount**.

4.6. practice: mounting file systems

1. Mount the small 200MB partition on /home/project22.

2. Mount the big 400MB primary partition on /mnt, the copy some files to it (everything in /etc). Then umount, and mount the file system as read only on /srv/nfs/salesnumbers. Where are the files you copied ?

3. Verify your work with **fdisk**, **df** and **mount**. Also look in **/etc/mtab** and **/proc/mounts**.

4. Make both mounts permanent, test that it works.

5. What happens when you mount a file system on a directory that contains some files ?

6. What happens when you mount two file systems on the same mount point ?

7. (optional) Describe the difference between these commands: find, locate, updatedb, make-whatis, whereis, apropos, which and type.

8. (optional) Perform a file system check on the partition mounted at /srv/nfs/salesnumbers.

4.7. solution: mounting file systems

1. Mount the small 200MB partition on /home/project22.

```
mkdir /home/project22
mount /dev/sdc1 /home/project22
```

2. Mount the big 400MB primary partition on /mnt, the copy some files to it (everything in /etc). Then umount, and mount the file system as read only on /srv/nfs/salesnumbers. Where are the files you copied ?

```
mount /dev/sdb1 /mnt
cp -r /etc /mnt
ls -l /mnt
```

```
umount /mnt
ls -l /mnt
```

```
mkdir -p /srv/nfs/salesnumbers
mount /dev/sdb1 /srv/nfs/salesnumbers
```

```
You see the files in /srv/nfs/salenumbers now...
```

```
But physically they are on ext3 on partition /dev/sdb1
```

3. Verify your work with **fdisk**, **df** and **mount**. Also look in **/etc/mtab** and **/proc/mounts**.

```
fdisk -l
df -h
mount
```

```
All three the above commands should show your mounted partitions.
```

```
grep project22 /etc/mtab
grep project22 /proc/mounts
```

4. Make both mounts permanent, test that it works.

```
add the following lines to /etc/fstab
```

```
/dev/sdc1 /home/project22 auto defaults 0 0
/dev/sdb1 /srv/nfs/salesnumbers auto defaults 0 0
```

5. What happens when you mount a file system on a directory that contains some files ?

```
The files are hidden until umount.
```

6. What happens when you mount two file systems on the same mount point ?

```
Only the last mounted fs is visible.
```

7. (optional) Describe the difference between these commands: find, locate, updatedb, makewhatis, whereis, apropos, which and type.

```
man find
man locate
...
```

8. (optional) Perform a file system check on the partition mounted at /srv/nfs/salesnumbers.

```
# umount /srv/nfs/salesnumbers (optional but recommended)
# fsck /dev/sdb1
```

Chapter 5. introduction to raid

5.1. hardware or software

Redundant Array of Independent (originally Inexpensive) Disks or **RAID** can be set up using hardware or software. Hardware RAID is more expensive, but offers better performance. Software RAID is cheaper and easier to manage, but it uses your CPU and your memory.

Where ten years ago nobody was arguing about the best choice being hardware RAID, this has changed since technologies like mdadm, lvm and even zfs focus more on managability. The workload on the cpu for software RAID used to be high, but cpu's have gotten a lot faster.

5.2. raid levels

5.2.1. raid 0

raid 0 uses two or more disks, and is often called **striping** (or stripe set, or striped volume). Data is divided in **chunks**, those chunks are evenly spread across every disk in the array. The main advantage of **raid 0** is that you can create **larger drives**. **raid 0** is the only **raid** without redundancy.

5.2.2. jbod

jbod uses two or more disks, and is often called **concatenating** (spanning, spanned set, or spanned volume). Data is written to the first disk, until it is full. Then data is written to the second disk... The main advantage of **jbod** (Just a Bunch of Disks) is that you can create **larger drives**. JBOD offers no redundancy.

5.2.3. raid 1

raid 1 uses exactly two disks, and is often called **mirroring** (or mirror set, or mirrored volume). All data written to the array is written on each disk. The main advantage of raid 1 is **redundancy**. The main disadvantage is that you lose at least half of your available disk space (in other words, you at least double the cost).

5.2.4. raid 2, 3 and 4 ?

raid 2 uses bit level striping, **raid 3** byte level, and **raid 4** is the same as **raid 5**, but with a dedicated parity disk. This is actually slower than **raid 5**, because every write would have to write parity to this one (bottleneck) disk. It is unlikely that you will ever see these **raid** levels in production.

5.2.5. raid 5

raid 5 uses **three** or more disks, each divided into chunks. Every time chunks are written to the array, one of the disks will receive a **parity** chunk. Unlike **raid 4**, the parity chunk will alternate between all disks. The main advantage of this is that **raid 5** will allow for full data recovery in case of **one** hard disk failure.

5.2.6. raid 6

raid 6 is very similar to **raid 5**, but uses two parity chunks. **raid 6** protects against two hard disk failures. Oracle Solaris **zfs** calls this **raidz2** (and also had **raidz3** with triple parity).

5.2.7. raid 0+1

raid 0+1 is a mirror(1) of stripes(0). This means you first create two **raid 0 stripe** sets, and then you set them up as a mirror set. For example, when you have six 100GB disks, then the stripe sets are each 300GB. Combined in a mirror, this makes 300GB total. **raid 0+1** will survive one disk failure. It will only survive the second disk failure if this disk is in the same stripe set as the previous failed disk.

5.2.8. raid 1+0

raid 1+0 is a stripe(0) of mirrors(1). For example, when you have six 100GB disks, then you first create three mirrors of 100GB each. You then stripe them together into a 300GB drive. In this example, as long as not all disks in the same mirror fail, it can survive up to three hard disk failures.

5.2.9. raid 50

raid 5+0 is a stripe(0) of **raid 5** arrays. Suppose you have nine disks of 100GB, then you can create three **raid 5** arrays of 200GB each. You can then combine them into one large stripe set.

5.2.10. many others

There are many other nested **raid** combinations, like **raid** 30, 51, 60, 100, 150, ...

5.3. building a software raid5 array

5.3.1. do we have three disks?

First, you have to attach some disks to your computer. In this scenario, three brand new disks of eight gigabyte each are added. Check with **fdisk -l** that they are connected.

```
[root@rhel6c ~]# fdisk -l 2> /dev/null | grep MB
Disk /dev/sdb: 8589 MB, 8589934592 bytes
Disk /dev/sdc: 8589 MB, 8589934592 bytes
Disk /dev/sdd: 8589 MB, 8589934592 bytes
```

5.3.2. fd partition type

The next step is to create a partition of type **fd** on every disk. The **fd** type is to set the partition as **Linux RAID autodetect**. See this (truncated) screenshot:

```
[root@rhel6c ~]# fdisk /dev/sdd
...
Command (m for help): n
Command action
   e   extended
   p   primary partition (1-4)
p
Partition number (1-4): 1
First cylinder (1-1044, default 1):
Using default value 1
Last cylinder, +cylinders or +size{K,M,G} (1-1044, default 1044):
Using default value 1044

Command (m for help): t
Selected partition 1
Hex code (type L to list codes): fd
Changed system type of partition 1 to fd (Linux raid autodetect)

Command (m for help): w
The partition table has been altered!

Calling ioctl() to re-read partition table.
Syncing disks.
```

5.3.3. verify all three partitions

Now all three disks are ready for **raid 5**, so we have to tell the system what to do with these disks.

```
[root@rhel6c ~]# fdisk -l 2> /dev/null | grep raid
/dev/sdb1       1        1044     8385898+  fd  Linux raid autodetect
/dev/sdc1       1        1044     8385898+  fd  Linux raid autodetect
/dev/sdd1       1        1044     8385898+  fd  Linux raid autodetect
```

5.3.4. create the raid5

The next step used to be *create the **raid table** in **/etc/raidtab***. Nowadays, you can just issue the command **mdadm** with the correct parameters.

The command below is split on two lines to fit this print, but you should type it on one line, without the backslash (\).

```
[root@rhel6c ~]# mdadm --create /dev/md0 --chunk=64 --level=5 --raid-\
devices=3 /dev/sdb1 /dev/sdc1 /dev/sdd1
mdadm: Defaulting to version 1.2 metadata
mdadm: array /dev/md0 started.
```

Below a partial screenshot how fdisk -l sees the **raid 5**.

```
[root@rhel6c ~]# fdisk -l /dev/md0

Disk /dev/md0: 17.2 GB, 17172135936 bytes
2 heads, 4 sectors/track, 4192416 cylinders
Units = cylinders of 8 * 512 = 4096 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 65536 bytes / 131072 bytes
Disk identifier: 0x00000000

Disk /dev/md0 doesn't contain a valid partition table
```

We could use this software **raid 5** array in the next topic: **lvm**.

5.3.5. /proc/mdstat

The status of the raid devices can be seen in **/proc/mdstat**. This example shows a **raid 5** in the process of rebuilding.

```
[root@rhel6c ~]# cat /proc/mdstat
Personalities : [raid6] [raid5] [raid4]
md0 : active raid5 sdd1[3] sdc1[1] sdb1[0]
      16769664 blocks super 1.2 level 5, 64k chunk, algorithm 2 [3/2] [UU_]
      [============>........]  recovery = 62.8% (5266176/8384832) finish=0\
.3min speed=139200K/sec
```

This example shows an active software **raid 5**.

```
[root@rhel6c ~]# cat /proc/mdstat
Personalities : [raid6] [raid5] [raid4]
md0 : active raid5 sdd1[3] sdc1[1] sdb1[0]
    16769664 blocks super 1.2 level 5, 64k chunk, algorithm 2 [3/3] [UUU]
```

5.3.6. mdadm --detail

Use **mdadm --detail** to get information on a raid device.

```
[root@rhel6c ~]# mdadm --detail /dev/md0
/dev/md0:
        Version : 1.2
  Creation Time : Sun Jul 17 13:48:41 2011
     Raid Level : raid5
     Array Size : 16769664 (15.99 GiB 17.17 GB)
  Used Dev Size : 8384832 (8.00 GiB 8.59 GB)
   Raid Devices : 3
  Total Devices : 3
    Persistence : Superblock is persistent

    Update Time : Sun Jul 17 13:49:43 2011
          State : clean
 Active Devices : 3
Working Devices : 3
 Failed Devices : 0
  Spare Devices : 0

         Layout : left-symmetric
     Chunk Size : 64K

           Name : rhel6c:0  (local to host rhel6c)
           UUID : c10fd9c3:08f9a25f:be913027:999c8e1f
         Events : 18

    Number   Major   Minor   RaidDevice State
       0       8       17        0      active sync   /dev/sdb1
       1       8       33        1      active sync   /dev/sdc1
       3       8       49        2      active sync   /dev/sdd1
```

5.3.7. removing a software raid

The software raid is visible in **/proc/mdstat** when active. To remove the raid completely so you can use the disks for other purposes, you stop (de-activate) it with **mdadm**.

```
[root@rhel6c ~]# mdadm --stop /dev/md0
mdadm: stopped /dev/md0
```

The disks can now be repartitioned.

5.3.8. further reading

Take a look at the man page of **mdadm** for more information. Below an example command to add a new partition while removing a faulty one.

```
mdadm /dev/md0 --add /dev/sdd1 --fail /dev/sdb1 --remove /dev/sdb1
```

5.4. practice: raid

1. Add three virtual disks of 1GB each to a virtual machine.
2. Create a software **raid 5** on the three disks. (It is not necessary to put a filesystem on it)
3. Verify with **fdisk** and in **/proc** that the **raid 5** exists.
4. Stop and remove the **raid 5**.
5. Create a **raid 1** to mirror two disks.

5.5. solution: raid

1. Add three virtual disks of 1GB each to a virtual machine.

2. Create a software **raid 5** on the three disks. (It is not necessary to put a filesystem on it)

3. Verify with **fdisk** and in **/proc** that the **raid 5** exists.

4. Stop and remove the **raid 5**.

5. Create a **raid 1** to mirror two disks.

```
[root@rhel6c ~]# mdadm --create /dev/md0 --level=1 --raid-devices=2 \
/dev/sdb1 /dev/sdc1
mdadm: Defaulting to version 1.2 metadata
mdadm: array /dev/md0 started.
[root@rhel6c ~]# cat /proc/mdstat
Personalities : [raid6] [raid5] [raid4] [raid1]
md0 : active raid1 sdc1[1] sdb1[0]
      8384862 blocks super 1.2 [2/2] [UU]
      [====>.............]  resync = 20.8% (1745152/8384862) \
finish=0.5min speed=218144K/sec
```

Chapter 6. introduction to uuid's

A **uuid** or **universally unique identifier** is used to uniquely identify objects. This 128bit standard allows anyone to create a unique **uuid**.

This chapter takes a brief look at **uuid's**.

6.1. about unique objects

Older versions of Linux have a **vol_id** utility to display the **uuid** of a file system.

```
root@debian5:~# vol_id --uuid /dev/sda1
193c3c9b-2c40-9290-8b71-4264ee4d4c82
```

Red Hat Enterprise Linux 5 puts **vol_id** in **/lib/udev/vol_id**, which is not in the $PATH. The syntax is also a bit different from Debian/Ubuntu/Mint.

```
root@rhel53 ~# /lib/udev/vol_id -u /dev/hda1
48a6a316-9ca9-4214-b5c6-e7b33a77e860
```

This utility is not available in standard installations of RHEL6 or Debian6.

6.2. tune2fs

Use **tune2fs** to find the **uuid** of a file system.

```
[root@RHEL5 ~]# tune2fs -l /dev/sda1 | grep UUID
Filesystem UUID:          11cfc8bc-07c0-4c3f-9f64-78422ef1dd5c
[root@RHEL5 ~]# /lib/udev/vol_id -u /dev/sda1
11cfc8bc-07c0-4c3f-9f64-78422ef1dd5c
```

6.3. uuid

There is more information in the manual of **uuid**, a tool that can generate uuid's.

```
[root@rhel65 ~]# yum install uuid
(output truncated)
[root@rhel65 ~]# man uuid
```

6.4. uuid in /etc/fstab

You can use the **uuid** to make sure that a volume is universally uniquely identified in **/etc/fstab**. The device name can change depending on the disk devices that are present at boot time, but a **uuid** never changes.

First we use **tune2fs** to find the **uuid**.

```
[root@RHEL5 ~]# tune2fs -l /dev/sdc1 | grep UUID
Filesystem UUID:          7626d73a-2bb6-4937-90ca-e451025d64e8
```

Then we check that it is properly added to **/etc/fstab**, the **uuid** replaces the variable device-name /dev/sdc1.

```
[root@RHEL5 ~]# grep UUID /etc/fstab
UUID=7626d73a-2bb6-4937-90ca-e451025d64e8 /home/pro42 ext3 defaults 0 0
```

Now we can mount the volume using the mount point defined in **/etc/fstab**.

```
[root@RHEL5 ~]# mount /home/pro42
[root@RHEL5 ~]# df -h | grep 42
/dev/sdc1             397M   11M  366M   3% /home/pro42
```

The real test now, is to remove **/dev/sdb** from the system, reboot the machine and see what happens. After the reboot, the disk previously known as **/dev/sdc** is now **/dev/sdb**.

```
[root@RHEL5 ~]# tune2fs -l /dev/sdb1 | grep UUID
Filesystem UUID:          7626d73a-2bb6-4937-90ca-e451025d64e8
```

And thanks to the **uuid** in **/etc/fstab**, the mountpoint is mounted on the same disk as before.

```
[root@RHEL5 ~]# df -h | grep sdb
/dev/sdb1             397M   11M  366M   3% /home/pro42
```

6.5. uuid as a boot device

Recent Linux distributions (Debian, Ubuntu, ...) use **grub** with a **uuid** to identify the root file system.

This example shows how a **root=/dev/sda1** is replaced with a **uuid**.

```
title           Ubuntu 9.10, kernel 2.6.31-19-generic
uuid            f001ba5d-9077-422a-9634-8d23d57e782a
kernel          /boot/vmlinuz-2.6.31-19-generic \
root=UUID=f001ba5d-9077-422a-9634-8d23d57e782a ro quiet splash
initrd          /boot/initrd.img-2.6.31-19-generic
```

The screenshot above contains only four lines. The line starting with **root=** is the continuation of the **kernel** line.

RHEL and CentOS boot from LVM after a default install.

6.6. practice: uuid and filesystems

1. Find the **uuid** of one of your **ext3** partitions with **tune2fs** (and **vol_id** if you are on RHEL5).

2. Use this **uuid** in **/etc/fstab** and test that it works with a simple **mount**.

3. (optional) Test it also by removing a disk (so the device name is changed). You can edit settings in vmware/Virtualbox to remove a hard disk.

4. Display the **root=** directive in **/boot/grub/menu.lst**. (We see later in the course how to maintain this file.)

5. (optional on ubuntu) Replace the **/dev/xxx** in **/boot/grub/menu.lst** with a **uuid** (use an extra stanza for this). Test that it works.

6.7. solution: uuid and filesystems

1. Find the **uuid** of one of your **ext3** partitions with **tune2fs** (and **vol_id** if you are on RHEL5).

```
root@rhel55:~# /lib/udev/vol_id -u /dev/hda1
60926898-2c78-49b4-a71d-c1d6310c87cc

root@ubu1004:~# tune2fs -l /dev/sda2 | grep UUID
Filesystem UUID:          3007b743-1dce-2d62-9a59-cf25f85191b7
```

2. Use this **uuid** in **/etc/fstab** and test that it works with a simple **mount**.

```
tail -1 /etc/fstab
UUID=60926898-2c78-49b4-a71d-c1d6310c87cc /home/pro42 ext3 defaults 0 0
```

3. (optional) Test it also by removing a disk (so the device name is changed). You can edit settings in vmware/Virtualbox to remove a hard disk.

4. Display the **root=** directive in **/boot/grub/menu.lst**. (We see later in the course how to maintain this file.)

```
paul@deb503:~$ grep ^[^#] /boot/grub/menu.lst | grep root=
kernel          /boot/vmlinuz-2.6.26-2-686 root=/dev/hda1 ro selinux=1 quiet
kernel          /boot/vmlinuz-2.6.26-2-686 root=/dev/hda1 ro selinux=1 single
```

5. (optional on ubuntu) Replace the **/dev/xxx** in **/boot/grub/menu.lst** with a **uuid** (use an extra stanza for this). Test that it works.

Chapter 7. logical volume management

Most **lvm** implementations support **physical storage grouping**, **logical volume resizing** and **data migration**.

Physical storage grouping is a fancy name for grouping multiple block devices (hard disks, but also iSCSI etc) into a logical mass storage device. To enlarge this physical group, block devices (including partitions) can be added at a later time.

The size of **lvm volumes** on this **physical group** is independent of the individual size of the components. The total size of the group is the limit.

One of the nice features of **lvm** is the logical volume resizing. You can increase the size of an **lvm volume**, sometimes even without any downtime. Additionally, you can migrate data away from a failing hard disk device, create mirrors and create snapshots.

7.1. introduction to lvm

7.1.1. problems with standard partitions

There are some problems when working with hard disks and standard partitions. Consider a system with a small and a large hard disk device, partitioned like this. The first disk (/dev/sda) is partitioned in two, the second disk (/dev/sdb) has three partitions.

Table 7.1. disk partitioning example

/dev/sda		**/dev/sdb**			
/dev/sda1	/dev/sda2	/dev/sdb1	/dev/sdb2	/dev/sdb3	unused
/boot	/	/var	/home	/project42	
ext2	ext3	ext2	reiserfs	ext3	

In the example above, consider the options when you want to enlarge the space available for **/project42**. What can you do ? The solution will always force you to unmount the filesystem, take a backup of the data, remove and recreate partitions, and then restore the data and remount the file system.

7.1.2. solution with lvm

Using **lvm** will create a virtual layer between the mounted file systems and the hardware devices. This virtual layer will allow for an administrator to enlarge a mounted file system in use. When **lvm** is properly used, then there is no need to unmount the file system to enlarge it.

Table 7.2. LVM Example

/dev/sda		**/dev/sdb**			
Volume Group					
/boot	/	/var	/home	/project42	
ext2	ext3	ext2	reiserfs	ext3	

7.2. lvm terminology

7.2.1. physical volume (pv)

A **physical volume** is any block device (a disk, a partition, a RAID device or even an iSCSI device). All these devices can become a member of a **volume group**.

The commands used to manage a **physical volume** start with pv.

```
[root@centos65 ~]# pv
pvchange   pvck       pvcreate   pvdisplay  pvmove     pvremove
pvresize   pvs        pvscan
```

7.2.2. volume group (vg)

A **volume group** is an abstraction layer between **block devices** and **logical volumes**.

The commands used to manage a **volume group** start with vg.

```
[root@centos65 ~]# vg
vgcfgbackup    vgconvert      vgextend       vgmknodes      vgs
vgcfgrestore   vgcreate       vgimport       vgreduce       vgscan
vgchange       vgdisplay      vgimportclone  vgremove       vgsplit
vgck           vgexport       vgmerge        vgrename
```

7.2.3. logical volume (lv)

A **logical volume** is created in a **volume group**. Logical volumes that contain a file system can be mounted. The use of **logical volumes** is similar to the use of **partitions** and is accomplished with the same standard commands (mkfs, mount, fsck, df, ...).

The commands used to manage a **logical volume** start with lv.

```
[root@centos65 ~]# lv
lvchange     lvextend     lvmdiskscan  lvmsar       lvresize
lvconvert    lvm          lvmdump      lvreduce     lvs
lvcreate     lvmchange    lvmetad      lvremove     lvscan
lvdisplay    lvmconf      lvmsadc      lvrename
```

7.3. example: using lvm

This example shows how you can use a device (in this case /dev/sdc, but it could have been /dev/sdb or any other disk or partition) with lvm, how to create a volume group (vg) and how to create and use a logical volume (vg/lvol0).

First thing to do, is create physical volumes that can join the volume group with **pvcreate**. This command makes a disk or partition available for use in Volume Groups. The screenshot shows how to present the SCSI Disk device to LVM.

```
root@RHEL4:~# pvcreate /dev/sdc
Physical volume "/dev/sdc" successfully created
```

*Note: lvm **will** work fine when using the complete device, but another operating system on the same computer (or on the same SAN) will not recognize lvm and will mark the block device as being empty! You can avoid this by creating a partition that spans the whole device, then run **pvcreate** on the partition instead of the disk.*

Then **vgcreate** creates a volume group using one device. Note that more devices could be added to the volume group.

```
root@RHEL4:~# vgcreate vg /dev/sdc
Volume group "vg" successfully created
```

The last step **lvcreate** creates a logical volume.

```
root@RHEL4:~# lvcreate --size 500m vg
Logical volume "lvol0" created
```

The logical volume /dev/vg/lvol0 can now be formatted with ext2, and mounted for normal use.

```
root@RHELv4u2:~# mke2fs -m0 -j /dev/vg/lvol0
mke2fs 1.35 (28-Feb-2004)
Filesystem label=
OS type: Linux
Block size=1024 (log=0)
Fragment size=1024 (log=0)
128016 inodes, 512000 blocks
0 blocks (0.00%) reserved for the super user
First data block=1
Maximum filesystem blocks=67633152
63 block groups
8192 blocks per group, 8192 fragments per group
2032 inodes per group
Superblock backups stored on blocks:
8193, 24577, 40961, 57345, 73729, 204801, 221185, 401409

Writing inode tables: done
Creating journal (8192 blocks): done
Writing superblocks and filesystem accounting information: done

This filesystem will be automatically checked every 37 mounts or
180 days, whichever comes first.  Use tune2fs -c or -i to override.
root@RHELv4u2:~# mkdir /home/project10
root@RHELv4u2:~# mount /dev/vg/lvol0 /home/project10/
root@RHELv4u2:~# df -h | grep proj
/dev/mapper/vg-lvol0  485M   11M  474M   3% /home/project10
```

A logical volume is very similar to a partition, it can be formatted with a file system, and can be mounted so users can access it.

7.4. example: extend a logical volume

A logical volume can be extended without unmounting the file system. Whether or not a volume can be extended depends on the file system it uses. Volumes that are mounted as vfat or ext2 cannot be extended, so in the example here we use the ext3 file system.

The fdisk command shows us newly added scsi-disks that will serve our lvm volume. This volume will then be extended. First, take a look at these disks.

```
[root@RHEL5 ~]# fdisk -l | grep sd[bc]
Disk /dev/sdb doesn't contain a valid partition table
Disk /dev/sdc doesn't contain a valid partition table
Disk /dev/sdb: 1181 MB, 1181115904 bytes
Disk /dev/sdc: 429 MB, 429496320 bytes
```

You already know how to partition a disk, below the first disk is partitioned (in one big primary partition), the second disk is left untouched.

```
[root@RHEL5 ~]# fdisk -l | grep sd[bc]
Disk /dev/sdc doesn't contain a valid partition table
Disk /dev/sdb: 1181 MB, 1181115904 bytes
/dev/sdb1               1         143     1148616   83  Linux
Disk /dev/sdc: 429 MB, 429496320 bytes
```

You also know how to prepare disks for lvm with **pvcreate**, and how to create a volume group with **vgcreate**. This example adds both the partitioned disk and the untouched disk to the volume group named **vg2**.

```
[root@RHEL5 ~]# pvcreate /dev/sdb1
  Physical volume "/dev/sdb1" successfully created
[root@RHEL5 ~]# pvcreate /dev/sdc
  Physical volume "/dev/sdc" successfully created
[root@RHEL5 ~]# vgcreate vg2 /dev/sdb1 /dev/sdc
  Volume group "vg2" successfully created
```

You can use **pvdisplay** to verify that both the disk and the partition belong to the volume group.

```
[root@RHEL5 ~]# pvdisplay | grep -B1 vg2
  PV Name               /dev/sdb1
  VG Name               vg2
--
  PV Name               /dev/sdc
  VG Name               vg2
```

And you are familiar both with the **lvcreate** command to create a small logical volume and the **mke2fs** command to put ext2 on it.

```
[root@RHEL5 ~]# lvcreate --size 200m vg2
  Logical volume "lvol0" created
[root@RHEL5 ~]# mke2fs -m20 -j /dev/vg2/lvol0
...
```

As you see, we end up with a mounted logical volume that according to **df** is almost 200 megabyte in size.

```
[root@RHEL5 ~]# mkdir /home/resizetest
[root@RHEL5 ~]# mount /dev/vg2/lvol0 /home/resizetest/
[root@RHEL5 ~]# df -h | grep resizetest
                      194M  5.6M  149M   4% /home/resizetest
```

Extending the volume is easy with **lvextend**.

```
[root@RHEL5 ~]# lvextend -L +100 /dev/vg2/lvol0
  Extending logical volume lvol0 to 300.00 MB
  Logical volume lvol0 successfully resized
```

But as you can see, there is a small problem: it appears that df is not able to display the extended volume in its full size. This is because the filesystem is only set for the size of the volume before the extension was added.

```
[root@RHEL5 ~]# df -h | grep resizetest
                      194M  5.6M  149M   4% /home/resizetest
```

With **lvdisplay** however we can see that the volume is indeed extended.

```
[root@RHEL5 ~]# lvdisplay /dev/vg2/lvol0 | grep Size
  LV Size                300.00 MB
```

To finish the extension, you need **resize2fs** to span the filesystem over the full size of the logical volume.

```
[root@RHEL5 ~]# resize2fs /dev/vg2/lvol0
resize2fs 1.39 (29-May-2006)
Filesystem at /dev/vg2/lvol0 is mounted on /home/resizetest; on-line re\
sizing required
Performing an on-line resize of /dev/vg2/lvol0 to 307200 (1k) blocks.
The filesystem on /dev/vg2/lvol0 is now 307200 blocks long.
```

Congratulations, you just successfully expanded a logical volume.

```
[root@RHEL5 ~]# df -h | grep resizetest
                      291M  6.1M  225M   3% /home/resizetest
[root@RHEL5 ~]#
```

7.5. example: resize a physical Volume

This is a humble demonstration of how to resize a physical Volume with lvm (after you resize it with fdisk). The demonstration starts with a 100MB partition named /dev/sde1. We used fdisk to create it, and to verify the size.

```
[root@RHEL5 ~]# fdisk -l 2>/dev/null | grep sde1
/dev/sde1               1         100      102384   83  Linux
[root@RHEL5 ~]#
```

Now we can use pvcreate to create the Physical Volume, followed by pvs to verify the creation.

```
[root@RHEL5 ~]# pvcreate /dev/sde1
  Physical volume "/dev/sde1" successfully created
[root@RHEL5 ~]# pvs | grep sde1
  /dev/sde1             lvm2 --    99.98M  99.98M
[root@RHEL5 ~]#
```

The next step is ti use fdisk to enlarge the partition (actually deleting it and then recreating /dev/sde1 with more cylinders).

```
[root@RHEL5 ~]# fdisk /dev/sde

Command (m for help): p

Disk /dev/sde: 858 MB, 858993152 bytes
64 heads, 32 sectors/track, 819 cylinders
Units = cylinders of 2048 * 512 = 1048576 bytes

   Device Boot      Start         End      Blocks   Id  System
/dev/sde1               1         100      102384   83  Linux

Command (m for help): d
Selected partition 1

Command (m for help): n
Command action
   e   extended
   p   primary partition (1-4)
p
Partition number (1-4):
Value out of range.
Partition number (1-4): 1
First cylinder (1-819, default 1):
Using default value 1
Last cylinder or +size or +sizeM or +sizeK (1-819, default 819): 200

Command (m for help): w
The partition table has been altered!

Calling ioctl() to re-read partition table.
Syncing disks.
[root@RHEL5 ~]#
```

When we now use fdisk and pvs to verify the size of the partition and the Physical Volume, then there is a size difference. LVM is still using the old size.

```
[root@RHEL5 ~]# fdisk -l 2>/dev/null | grep sde1
/dev/sde1               1         200      204784   83  Linux
[root@RHEL5 ~]# pvs | grep sde1
  /dev/sde1             lvm2 --    99.98M  99.98M
[root@RHEL5 ~]#
```

Executing pvresize on the Physical Volume will make lvm aware of the size change of the partition. The correct size can be displayed with pvs.

```
[root@RHEL5 ~]# pvresize /dev/sde1
  Physical volume "/dev/sde1" changed
  1 physical volume(s) resized / 0 physical volume(s) not resized
[root@RHEL5 ~]# pvs | grep sde1
  /dev/sde1             lvm2 --   199.98M 199.98M
[root@RHEL5 ~]#
```

7.6. example: mirror a logical volume

We start by creating three physical volumes for lvm. Then we verify the creation and the size with pvs. Three physical disks because lvm uses two disks for the mirror and a third disk for the mirror log!

```
[root@RHEL5 ~]# pvcreate /dev/sdb /dev/sdc /dev/sdd
  Physical volume "/dev/sdb" successfully created
  Physical volume "/dev/sdc" successfully created
  Physical volume "/dev/sdd" successfully created
[root@RHEL5 ~]# pvs
  PV         VG         Fmt  Attr PSize   PFree
  /dev/sdb              lvm2 --   409.60M 409.60M
  /dev/sdc              lvm2 --   409.60M 409.60M
  /dev/sdd              lvm2 --   409.60M 409.60M
```

Then we create the Volume Group and verify again with pvs. Notice how the three physical volumes now belong to vg33, and how the size is rounded down (in steps of the extent size, here 4MB).

```
[root@RHEL5 ~]# vgcreate vg33 /dev/sdb /dev/sdc /dev/sdd
  Volume group "vg33" successfully created
[root@RHEL5 ~]# pvs
  PV         VG         Fmt  Attr PSize   PFree
  /dev/sda2  VolGroup00 lvm2 a-    15.88G      0
  /dev/sdb   vg33       lvm2 a-   408.00M 408.00M
  /dev/sdc   vg33       lvm2 a-   408.00M 408.00M
  /dev/sdd   vg33       lvm2 a-   408.00M 408.00M
[root@RHEL5 ~]#
```

The last step is to create the Logical Volume with **lvcreate**. Notice the **-m 1** switch to create one mirror. Notice also the change in free space in all three Physical Volumes!

```
[root@RHEL5 ~]# lvcreate --size 300m -n lvmir -m 1 vg33
  Logical volume "lvmir" created
[root@RHEL5 ~]# pvs
  PV         VG         Fmt  Attr PSize   PFree
  /dev/sda2  VolGroup00 lvm2 a-    15.88G      0
  /dev/sdb   vg33       lvm2 a-   408.00M 108.00M
  /dev/sdc   vg33       lvm2 a-   408.00M 108.00M
  /dev/sdd   vg33       lvm2 a-   408.00M 404.00M
```

You can see the copy status of the mirror with lvs. It currently shows a 100 percent copy.

```
[root@RHEL5 ~]# lvs vg33/lvmir
  LV    VG   Attr   LSize   Origin Snap%  Move Log        Copy%
  lvmir vg33 mwi-ao 300.00M                lvmir_mlog 100.00
```

7.7. example: snapshot a logical volume

A snapshot is a virtual copy of all the data at a point in time on a volume. A snapshot Logical Volume will retain a copy of all changed files of the snapshotted Logical Volume.

The example below creates a snapshot of the bigLV Logical Volume.

```
[root@RHEL5 ~]# lvcreate -L100M -s -n snapLV vg42/bigLV
  Logical volume "snapLV" created
[root@RHEL5 ~]#
```

You can see with lvs that the snapshot snapLV is indeed a snapshot of bigLV. Moments after taking the snapshot, there are few changes to bigLV (0.02 percent).

```
[root@RHEL5 ~]# lvs
  LV       VG         Attr   LSize   Origin Snap%  Move Log Copy%
  bigLV    vg42       owi-a- 200.00M
  snapLV   vg42       swi-a- 100.00M bigLV    0.02
[root@RHEL5 ~]#
```

But after using bigLV for a while, more changes are done. This means the snapshot volume has to keep more original data (10.22 percent).

```
[root@RHEL5 ~]# lvs | grep vg42
  bigLV    vg42       owi-ao 200.00M
  snapLV   vg42       swi-a- 100.00M bigLV   10.22
[root@RHEL5 ~]#
```

You can now use regular backup tools (dump, tar, cpio, ...) to take a backup of the snapshot Logical Volume. This backup will contain all data as it existed on bigLV at the time the snapshot was taken. When the backup is done, you can remove the snapshot.

```
[root@RHEL5 ~]# lvremove vg42/snapLV
Do you really want to remove active logical volume "snapLV"? [y/n]: y
  Logical volume "snapLV" successfully removed
[root@RHEL5 ~]#
```

7.8. verifying existing physical volumes

7.8.1. lvmdiskscan

To get a list of block devices that can be used with LVM, use **lvmdiskscan**. The example below uses grep to limit the result to SCSI devices.

```
[root@RHEL5 ~]# lvmdiskscan | grep sd
  /dev/sda1               [      101.94 MB]
  /dev/sda2               [       15.90 GB] LVM physical volume
  /dev/sdb                [      409.60 MB]
  /dev/sdc                [      409.60 MB]
  /dev/sdd                [      409.60 MB] LVM physical volume
  /dev/sde1               [       95.98 MB]
  /dev/sde5               [      191.98 MB]
  /dev/sdf                [      819.20 MB] LVM physical volume
  /dev/sdg1               [      818.98 MB]
[root@RHEL5 ~]#
```

7.8.2. pvs

The easiest way to verify whether devices are known to lvm is with the **pvs** command. The screenshot below shows that only /dev/sda2 is currently known for use with LVM. It shows that /dev/sda2 is part of Volgroup00 and is almost 16GB in size. It also shows /dev/sdc and /dev/sdd as part of vg33. The device /dev/sdb is knwon to lvm, but not linked to any Volume Group.

```
[root@RHEL5 ~]# pvs
  PV         VG         Fmt  Attr PSize   PFree
  /dev/sda2  VolGroup00 lvm2 a-    15.88G      0
  /dev/sdb              lvm2 --   409.60M 409.60M
  /dev/sdc   vg33       lvm2 a-   408.00M 408.00M
  /dev/sdd   vg33       lvm2 a-   408.00M 408.00M
[root@RHEL5 ~]#
```

7.8.3. pvscan

The **pvscan** command will scan all disks for existing Physical Volumes. The information is similar to pvs, plus you get a line with total sizes.

```
[root@RHEL5 ~]# pvscan
  PV /dev/sdc    VG vg33         lvm2 [408.00 MB / 408.00 MB free]
  PV /dev/sdd    VG vg33         lvm2 [408.00 MB / 408.00 MB free]
  PV /dev/sda2   VG VolGroup00   lvm2 [15.88 GB / 0    free]
  PV /dev/sdb                    lvm2 [409.60 MB]
  Total: 4 [17.07 GB] / in use: 3 [16.67 GB] / in no VG: 1 [409.60 MB]
[root@RHEL5 ~]#
```

7.8.4. pvdisplay

Use **pvdisplay** to get more information about physical volumes. You can also use **pvdisplay** without an argument to display information about all physical (lvm) volumes.

```
[root@RHEL5 ~]# pvdisplay /dev/sda2
  --- Physical volume ---
  PV Name               /dev/sda2
  VG Name               VolGroup00
  PV Size               15.90 GB / not usable 20.79 MB
  Allocatable           yes (but full)
  PE Size (KByte)       32768
  Total PE              508
  Free PE               0
  Allocated PE          508
  PV UUID               TobYfp-Ggg0-Rf8r-xtLd-5XgN-RSPc-8vkTHD

[root@RHEL5 ~]#
```

7.9. verifying existing volume groups

7.9.1. vgs

Similar to **pvs** is the use of **vgs** to display a quick overview of all volume groups. There is only one volume group in the screenshot below, it is named VolGroup00 and is almost 16GB in size.

```
[root@RHEL5 ~]# vgs
  VG         #PV #LV #SN Attr   VSize  VFree
  VolGroup00   1   2   0 wz--n- 15.88G    0
[root@RHEL5 ~]#
```

7.9.2. vgscan

The **vgscan** command will scan all disks for existing Volume Groups. It will also update the **/etc/lvm/.cache** file. This file contains a list of all current lvm devices.

```
[root@RHEL5 ~]# vgscan
  Reading all physical volumes.  This may take a while...
  Found volume group "VolGroup00" using metadata type lvm2
[root@RHEL5 ~]#
```

LVM will run the vgscan automatically at boot-up, so if you add hot swap devices, then you will need to run vgscan to update /etc/lvm/.cache with the new devices.

7.9.3. vgdisplay

The **vgdisplay** command will give you more detailed information about a volume group (or about all volume groups if you omit the argument).

```
[root@RHEL5 ~]# vgdisplay VolGroup00
  --- Volume group ---
  VG Name               VolGroup00
  System ID
  Format                lvm2
  Metadata Areas        1
  Metadata Sequence No  3
  VG Access             read/write
  VG Status             resizable
  MAX LV                0
  Cur LV                2
  Open LV               2
  Max PV                0
  Cur PV                1
  Act PV                1
  VG Size               15.88 GB
  PE Size               32.00 MB
  Total PE              508
  Alloc PE / Size       508 / 15.88 GB
  Free  PE / Size       0 / 0
  VG UUID               qsXvJb-71qV-9l7U-ishX-FobM-qptE-VXmKIg

[root@RHEL5 ~]#
```

7.10. verifying existing logical volumes

7.10.1. lvs

Use **lvs** for a quick look at all existing logical volumes. Below you can see two logical volumes named LogVol00 and LogVol01.

```
[root@RHEL5 ~]# lvs
  LV       VG         Attr   LSize  Origin Snap%  Move Log Copy%
  LogVol00 VolGroup00 -wi-ao 14.88G
  LogVol01 VolGroup00 -wi-ao  1.00G
[root@RHEL5 ~]#
```

7.10.2. lvscan

The **lvscan** command will scan all disks for existing Logical Volumes.

```
[root@RHEL5 ~]# lvscan
  ACTIVE            '/dev/VolGroup00/LogVol00' [14.88 GB] inherit
  ACTIVE            '/dev/VolGroup00/LogVol01' [1.00 GB] inherit
[root@RHEL5 ~]#
```

7.10.3. lvdisplay

More detailed information about logical volumes is available through the **lvdisplay(1)** command.

```
[root@RHEL5 ~]# lvdisplay VolGroup00/LogVol01
  --- Logical volume ---
  LV Name                /dev/VolGroup00/LogVol01
  VG Name                VolGroup00
  LV UUID                RnTGK6-xWsi-t530-ksJx-7cax-co5c-A1KlDp
  LV Write Access        read/write
  LV Status              available
  # open                 1
  LV Size                1.00 GB
  Current LE             32
  Segments               1
  Allocation             inherit
  Read ahead sectors     0
  Block device           253:1

[root@RHEL5 ~]#
```

7.11. manage physical volumes

7.11.1. pvcreate

Use the **pvcreate** command to add devices to lvm. This example shows how to add a disk (or hardware RAID device) to lvm.

```
[root@RHEL5 ~]# pvcreate /dev/sdb
  Physical volume "/dev/sdb" successfully created
[root@RHEL5 ~]#
```

This example shows how to add a partition to lvm.

```
[root@RHEL5 ~]# pvcreate /dev/sdc1
  Physical volume "/dev/sdc1" successfully created
[root@RHEL5 ~]#
```

You can also add multiple disks or partitions as target to pvcreate. This example adds three disks to lvm.

```
[root@RHEL5 ~]# pvcreate /dev/sde /dev/sdf /dev/sdg
  Physical volume "/dev/sde" successfully created
  Physical volume "/dev/sdf" successfully created
  Physical volume "/dev/sdg" successfully created
[root@RHEL5 ~]#
```

7.11.2. pvremove

Use the **pvremove** command to remove physical volumes from lvm. The devices may not be in use.

```
[root@RHEL5 ~]# pvremove /dev/sde /dev/sdf /dev/sdg
  Labels on physical volume "/dev/sde" successfully wiped
  Labels on physical volume "/dev/sdf" successfully wiped
  Labels on physical volume "/dev/sdg" successfully wiped
[root@RHEL5 ~]#
```

7.11.3. pvresize

When you used fdisk to resize a partition on a disk, then you must use **pvresize** to make lvm recognize the new size of the physical volume that represents this partition.

```
[root@RHEL5 ~]# pvresize /dev/sde1
  Physical volume "/dev/sde1" changed
  1 physical volume(s) resized / 0 physical volume(s) not resized
```

7.11.4. pvchange

With **pvchange** you can prevent the allocation of a Physical Volume in a new Volume Group or Logical Volume. This can be useful if you plan to remove a Physical Volume.

```
[root@RHEL5 ~]# pvchange -xn /dev/sdd
  Physical volume "/dev/sdd" changed
  1 physical volume changed / 0 physical volumes not changed
[root@RHEL5 ~]#
```

To revert your previous decision, this example shows you how te re-enable the Physical Volume to allow allocation.

```
[root@RHEL5 ~]# pvchange -xy /dev/sdd
  Physical volume "/dev/sdd" changed
  1 physical volume changed / 0 physical volumes not changed
[root@RHEL5 ~]#
```

7.11.5. pvmove

With **pvmove** you can move Logical Volumes from within a Volume Group to another Physical Volume. This must be done before removing a Physical Volume.

```
[root@RHEL5 ~]# pvs | grep vg1
  /dev/sdf   vg1        lvm2 a-   816.00M      0
  /dev/sdg   vg1        lvm2 a-   816.00M 816.00M
[root@RHEL5 ~]# pvmove /dev/sdf
  /dev/sdf: Moved: 70.1%
  /dev/sdf: Moved: 100.0%
[root@RHEL5 ~]# pvs | grep vg1
  /dev/sdf   vg1        lvm2 a-   816.00M 816.00M
  /dev/sdg   vg1        lvm2 a-   816.00M      0
```

7.12. manage volume groups

7.12.1. vgcreate

Use the **vgcreate** command to create a volume group. You can immediately name all the physical volumes that span the volume group.

```
[root@RHEL5 ~]# vgcreate vg42 /dev/sde /dev/sdf
  Volume group "vg42" successfully created
[root@RHEL5 ~]#
```

7.12.2. vgextend

Use the **vgextend** command to extend an existing volume group with a physical volume.

```
[root@RHEL5 ~]# vgextend vg42 /dev/sdg
  Volume group "vg42" successfully extended
[root@RHEL5 ~]#
```

7.12.3. vgremove

Use the **vgremove** command to remove volume groups from lvm. The volume groups may not be in use.

```
[root@RHEL5 ~]# vgremove vg42
  Volume group "vg42" successfully removed
[root@RHEL5 ~]#
```

7.12.4. vgreduce

Use the **vgreduce** command to remove a Physical Volume from the Volume Group.

The following example adds Physical Volume /dev/sdg to the vg1 Volume Group using vgextend. And then removes it again using vgreduce.

```
[root@RHEL5 ~]# pvs | grep sdg
  /dev/sdg              lvm2 --   819.20M 819.20M
[root@RHEL5 ~]# vgextend vg1 /dev/sdg
  Volume group "vg1" successfully extended
[root@RHEL5 ~]# pvs | grep sdg
  /dev/sdg   vg1        lvm2 a-   816.00M 816.00M
[root@RHEL5 ~]# vgreduce vg1 /dev/sdg
  Removed "/dev/sdg" from volume group "vg1"
[root@RHEL5 ~]# pvs | grep sdg
  /dev/sdg              lvm2 --   819.20M 819.20M
```

7.12.5. vgchange

Use the **vgchange** command to change parameters of a Volume Group.

This example shows how to prevent Physical Volumes from being added or removed to the Volume Group vg1.

```
[root@RHEL5 ~]# vgchange -xn vg1
  Volume group "vg1" successfully changed
[root@RHEL5 ~]# vgextend vg1 /dev/sdg
  Volume group vg1 is not resizable.
```

You can also use vgchange to change most other properties of a Volume Group. This example changes the maximum number of Logical Volumes and maximum number of Physical Volumes that vg1 can serve.

```
[root@RHEL5 ~]# vgdisplay vg1 | grep -i max
  MAX LV                0
  Max PV                0
[root@RHEL5 ~]# vgchange -l16 vg1
  Volume group "vg1" successfully changed
[root@RHEL5 ~]# vgchange -p8 vg1
  Volume group "vg1" successfully changed
[root@RHEL5 ~]# vgdisplay vg1 | grep -i max
  MAX LV                16
  Max PV                8
```

7.12.6. vgmerge

Merging two Volume Groups into one is done with **vgmerge**. The following example merges vg2 into vg1, keeping all the properties of vg1.

```
[root@RHEL5 ~]# vgmerge vg1 vg2
  Volume group "vg2" successfully merged into "vg1"
[root@RHEL5 ~]#
```

7.13. manage logical volumes

7.13.1. lvcreate

Use the **lvcreate** command to create Logical Volumes in a Volume Group. This example creates an 8GB Logical Volume in Volume Group vg42.

```
[root@RHEL5 ~]# lvcreate -L5G vg42
  Logical volume "lvol0" created
[root@RHEL5 ~]#
```

As you can see, lvm automatically names the Logical Volume **lvol0**. The next example creates a 200MB Logical Volume named MyLV in Volume Group vg42.

```
[root@RHEL5 ~]# lvcreate -L200M -nMyLV vg42
  Logical volume "MyLV" created
[root@RHEL5 ~]#
```

The next example does the same thing, but with different syntax.

```
[root@RHEL5 ~]# lvcreate --size 200M -n MyLV vg42
  Logical volume "MyLV" created
[root@RHEL5 ~]#
```

This example creates a Logical Volume that occupies 10 percent of the Volume Group.

```
[root@RHEL5 ~]# lvcreate -l 10%VG -n MyLV2 vg42
  Logical volume "MyLV2" created
[root@RHEL5 ~]#
```

This example creates a Logical Volume that occupies 30 percent of the remaining free space in the Volume Group.

```
[root@RHEL5 ~]# lvcreate -l 30%FREE -n MyLV3 vg42
  Logical volume "MyLV3" created
[root@RHEL5 ~]#
```

7.13.2. lvremove

Use the **lvremove** command to remove Logical Volumes from a Volume Group. Removing a Logical Volume requires the name of the Volume Group.

```
[root@RHEL5 ~]# lvremove vg42/MyLV
Do you really want to remove active logical volume "MyLV"? [y/n]: y
  Logical volume "MyLV" successfully removed
[root@RHEL5 ~]#
```

Removing multiple Logical Volumes will request confirmation for each individual volume.

```
[root@RHEL5 ~]# lvremove vg42/MyLV vg42/MyLV2 vg42/MyLV3
Do you really want to remove active logical volume "MyLV"? [y/n]: y
  Logical volume "MyLV" successfully removed
Do you really want to remove active logical volume "MyLV2"? [y/n]: y
  Logical volume "MyLV2" successfully removed
Do you really want to remove active logical volume "MyLV3"? [y/n]: y
  Logical volume "MyLV3" successfully removed
[root@RHEL5 ~]#
```

7.13.3. lvextend

Extending the volume is easy with **lvextend**. This example extends a 200MB Logical Volume with 100 MB.

```
[root@RHEL5 ~]# lvdisplay /dev/vg2/lvol0 | grep Size
  LV Size                200.00 MB
[root@RHEL5 ~]# lvextend -L +100 /dev/vg2/lvol0
  Extending logical volume lvol0 to 300.00 MB
  Logical volume lvol0 successfully resized
[root@RHEL5 ~]# lvdisplay /dev/vg2/lvol0 | grep Size
  LV Size                300.00 MB
```

The next example creates a 100MB Logical Volume, and then extends it to 500MB.

```
[root@RHEL5 ~]# lvcreate --size 100M -n extLV vg42
  Logical volume "extLV" created
[root@RHEL5 ~]# lvextend -L 500M vg42/extLV
  Extending logical volume extLV to 500.00 MB
  Logical volume extLV successfully resized
[root@RHEL5 ~]#
```

This example doubles the size of a Logical Volume.

```
[root@RHEL5 ~]# lvextend -l+100%LV vg42/extLV
  Extending logical volume extLV to 1000.00 MB
  Logical volume extLV successfully resized
[root@RHEL5 ~]#
```

7.13.4. lvrename

Renaming a Logical Volume is done with **lvrename**. This example renames extLV to bigLV in the vg42 Volume Group.

```
[root@RHEL5 ~]# lvrename vg42/extLV vg42/bigLV
  Renamed "extLV" to "bigLV" in volume group "vg42"
[root@RHEL5 ~]#
```

7.14. practice : lvm

1. Create a volume group that contains a complete disk and a partition on another disk.

2. Create two logical volumes (a small one and a bigger one) in this volumegroup. Format them wih ext3, mount them and copy some files to them.

3. Verify usage with fdisk, mount, pvs, vgs, lvs, pvdisplay, vgdisplay, lvdisplay and df. Does fdisk give you any information about lvm?

4. Enlarge the small logical volume by 50 percent, and verify your work!

5. Take a look at other commands that start with vg* , pv* or lv*.

6. Create a mirror and a striped Logical Volume.

7. Convert a linear logical volume to a mirror.

8. Convert a mirror logical volume to a linear.

9. Create a snapshot of a Logical Volume, take a backup of the snapshot. Then delete some files on the Logical Volume, then restore your backup.

10. Move your volume group to another disk (keep the Logical Volumes mounted).

11. If time permits, split a Volume Group with vgsplit, then merge it again with vgmerge.

7.15. solution : lvm

1. Create a volume group that contains a complete disk and a partition on another disk.

step 1: select disks:

```
root@rhel65:~# fdisk -l | grep Disk
Disk /dev/sda: 8589 MB, 8589934592 bytes
Disk identifier: 0x00055ca0
Disk /dev/sdb: 1073 MB, 1073741824 bytes
Disk identifier: 0x00000000
Disk /dev/sdc: 1073 MB, 1073741824 bytes
Disk identifier: 0x00000000
...
```

I choose /dev/sdb and /dev/sdc for now.

step 2: partition /dev/sdc

```
root@rhel65:~# fdisk /dev/sdc
Device contains neither a valid DOS partition table, nor Sun, SGI or OSF disk\
label
Building a new DOS disklabel with disk identifier 0x94c0e5d5.
Changes will remain in memory only, until you decide to write them.
After that, of course, the previous content won't be recoverable.

Warning: invalid flag 0x0000 of partition table 4 will be corrected by w(rite)

WARNING: DOS-compatible mode is deprecated. It's strongly recommended to
         switch off the mode (command 'c') and change display units to
         sectors (command 'u').

Command (m for help): n
Command action
   e   extended
   p   primary partition (1-4)
p
Partition number (1-4): 1
First cylinder (1-130, default 1):
Using default value 1
Last cylinder, +cylinders or +size{K,M,G} (1-130, default 130):
Using default value 130

Command (m for help): w
The partition table has been altered!

Calling ioctl() to re-read partition table.
Syncing disks.
```

step 3: pvcreate and vgcreate

```
root@rhel65:~# pvcreate /dev/sdb /dev/sdc1
  Physical volume "/dev/sdb" successfully created
  Physical volume "/dev/sdc1" successfully created
root@rhel65:~# vgcreate VG42 /dev/sdb /dev/sdc1
  Volume group "VG42" successfully created
```

2. Create two logical volumes (a small one and a bigger one) in this volumegroup. Format them wih ext3, mount them and copy some files to them.

```
root@rhel65:~# lvcreate --size 200m --name LVsmall VG42
  Logical volume "LVsmall" created
root@rhel65:~# lvcreate --size 600m --name LVbig VG42
  Logical volume "LVbig" created
root@rhel65:~# ls -l /dev/mapper/VG42-LVsmall
lrwxrwxrwx. 1 root root 7 Apr 20 20:41 /dev/mapper/VG42-LVsmall -> ../dm-2
root@rhel65:~# ls -l /dev/VG42/LVsmall
lrwxrwxrwx. 1 root root 7 Apr 20 20:41 /dev/VG42/LVsmall -> ../dm-2
root@rhel65:~# ls -l /dev/dm-2
brw-rw----. 1 root disk 253, 2 Apr 20 20:41 /dev/dm-2
```

```
root@rhel65:~# mkfs.ext3 /dev/mapper/VG42-LVsmall
mke2fs 1.41.12 (17-May-2010)
Filesystem label=
OS type: Linux
Block size=1024 (log=0)
Fragment size=1024 (log=0)
Stride=0 blocks, Stripe width=0 blocks
51200 inodes, 204800 blocks
10240 blocks (5.00%) reserved for the super user
First data block=1
Maximum filesystem blocks=67371008
25 block groups
8192 blocks per group, 8192 fragments per group
2048 inodes per group
Superblock backups stored on blocks:
 8193, 24577, 40961, 57345, 73729

Writing inode tables: done
Creating journal (4096 blocks): done
Writing superblocks and filesystem accounting information: done

This filesystem will be automatically checked every 39 mounts or
180 days, whichever comes first.  Use tune2fs -c or -i to override.
```

```
root@rhel65:~# mkfs.ext3 /dev/VG42/LVbig
mke2fs 1.41.12 (17-May-2010)
Filesystem label=
OS type: Linux
Block size=4096 (log=2)
Fragment size=4096 (log=2)
Stride=0 blocks, Stripe width=0 blocks
38400 inodes, 153600 blocks
7680 blocks (5.00%) reserved for the super user
First data block=0
Maximum filesystem blocks=159383552
5 block groups
32768 blocks per group, 32768 fragments per group
7680 inodes per group
Superblock backups stored on blocks:
 32768, 98304

Writing inode tables: done
Creating journal (4096 blocks): done
Writing superblocks and filesystem accounting information: done

This filesystem will be automatically checked every 25 mounts or
180 days, whichever comes first.  Use tune2fs -c or -i to override.
```

The mounting and copying of files.

```
root@rhel65:~# mkdir /srv/LVsmall
root@rhel65:~# mkdir /srv/LVbig
root@rhel65:~# mount /dev/mapper/VG42-LVsmall /srv/LVsmall
root@rhel65:~# mount /dev/VG42/LVbig /srv/LVbig
root@rhel65:~# cp -r /etc /srv/LVsmall/
root@rhel65:~# cp -r /var/log /srv/LVbig/
```

3. Verify usage with fdisk, mount, pvs, vgs, lvs, pvdisplay, vgdisplay, lvdisplay and df. Does fdisk give you any information about lvm?

Run all those commands (only two are shown below), then answer 'no'.

```
root@rhel65:~# df -h
Filesystem            Size  Used Avail Use% Mounted on
/dev/mapper/VolGroup-lv_root
                      6.7G  1.4G  5.0G  21% /
tmpfs                 246M     0  246M   0% /dev/shm
/dev/sda1             485M   77M  383M  17% /boot
/dev/mapper/VG42-LVsmall
                      194M   30M  154M  17% /srv/LVsmall
/dev/mapper/VG42-LVbig
                      591M   20M  541M   4% /srv/LVbig
root@rhel65:~# mount | grep VG42
/dev/mapper/VG42-LVsmall on /srv/LVsmall type ext3 (rw)
/dev/mapper/VG42-LVbig on /srv/LVbig type ext3 (rw)
```

4. Enlarge the small logical volume by 50 percent, and verify your work!

```
root@rhel65:~# lvextend VG42/LVsmall -l+50%LV
  Extending logical volume LVsmall to 300.00 MiB
  Logical volume LVsmall successfully resized
root@rhel65:~# resize2fs /dev/mapper/VG42-LVsmall
resize2fs 1.41.12 (17-May-2010)
Filesystem at /dev/mapper/VG42-LVsmall is mounted on /srv/LVsmall; on-line res\
izing required
old desc_blocks = 1, new_desc_blocks = 2
Performing an on-line resize of /dev/mapper/VG42-LVsmall to 307200 (1k) blocks.
The filesystem on /dev/mapper/VG42-LVsmall is now 307200 blocks long.

root@rhel65:~# df -h | grep small
/dev/mapper/VG42-LVsmall
                      291M   31M  246M  12% /srv/LVsmall
root@rhel65:~#
```

5. Take a look at other commands that start with vg* , pv* or lv*.

6. Create a mirror and a striped Logical Volume.

7. Convert a linear logical volume to a mirror.

8. Convert a mirror logical volume to a linear.

9. Create a snapshot of a Logical Volume, take a backup of the snapshot. Then delete some files on the Logical Volume, then restore your backup.

10. Move your volume group to another disk (keep the Logical Volumes mounted).

11. If time permits, split a Volume Group with vgsplit, then merge it again with vgmerge.

Chapter 8. iSCSI devices

This chapter teaches you how to setup an **iSCSI target server** and an **iSCSI initiator client**.

8.1. iSCSI terminology

iSCSI is a protocol that enables SCSI over IP. This means that you can have local SCSI devices (like /dev/sdb) without having the storage hardware in the local computer.

The computer holding the physical storage hardware is called the **iSCSI Target**. Each individual addressable iSCSI device on the target server will get a **LUN number**.

The iSCSI client computer that is connecting to the Target server is called an **Initiator**. An initiator will send SCSI commands over IP instead of directly to the hardware. The Initiator will connect to the Target.

8.2. iSCSI Target in RHEL/CentOS

This section will describe iSCSI Target setup on RHEL6, RHEL7 and CentOS.

Start with installing the **iSCSI Target** package.

```
yum install scsi-target-utils
```

We configure three local disks in **/etc/tgt/targets.conf** to become three LUN's.

```
<target iqn.2008-09.com.example:server.target2>
    direct-store /dev/sdb
    direct-store /dev/sdc
    direct-store /dev/sdd
    incominguser paul hunter2
</target>
```

Restart the service.

```
[root@centos65 ~]# service tgtd start
Starting SCSI target daemon:                               [  OK  ]
```

The standard local port for iSCSI Target is 3260, in case of doubt you can verify this with **netstat**.

```
[root@server1 tgt]# netstat -ntpl | grep tgt
tcp    0   0 0.0.0.0:3260       0.0.0.0:*          LISTEN      1670/tgtd
tcp    0   0 :::3260            :::*               LISTEN      1670/tgtd
```

The **tgt-admin -s** command should now give you a nice overview of the three LUN's (and also LUN 0 for the controller).

```
[root@server1 tgt]# tgt-admin -s
Target 1: iqn.2014-04.be.linux-training:server1.target1
    System information:
        Driver: iscsi
        State: ready
    I_T nexus information:
    LUN information:
        LUN: 0
            Type: controller
            SCSI ID: IET     00010000
            SCSI SN: beaf10
            Size: 0 MB, Block size: 1
            Online: Yes
            Removable media: No
            Prevent removal: No
            Readonly: No
            Backing store type: null
            Backing store path: None
            Backing store flags:
        LUN: 1
            Type: disk
            SCSI ID: IET     00010001
            SCSI SN: VB9f23197b-af6cfb60
            Size: 1074 MB, Block size: 512
            Online: Yes
            Removable media: No
            Prevent removal: No
            Readonly: No
            Backing store type: rdwr
            Backing store path: /dev/sdb
            Backing store flags:
        LUN: 2
            Type: disk
            SCSI ID: IET     00010002
            SCSI SN: VB8f554351-a1410828
            Size: 1074 MB, Block size: 512
            Online: Yes
            Removable media: No
            Prevent removal: No
            Readonly: No
            Backing store type: rdwr
            Backing store path: /dev/sdc
            Backing store flags:
        LUN: 3
            Type: disk
            SCSI ID: IET     00010003
            SCSI SN: VB1035d2f0-7ae90b49
            Size: 1074 MB, Block size: 512
            Online: Yes
            Removable media: No
            Prevent removal: No
            Readonly: No
            Backing store type: rdwr
            Backing store path: /dev/sdd
            Backing store flags:
    Account information:
    ACL information:
        ALL
```

8.3. iSCSI Initiator in RHEL/CentOS

This section will describe iSCSI Initiator setup on RHEL6, RHEL7 and CentOS.

Start with installing the **iSCSI Initiator** package.

```
[root@server2 ~]# yum install iscsi-initiator-utils
```

Then ask the **iSCSI target server** to send you the target names.

```
[root@server2 ~]# iscsiadm -m discovery -t sendtargets -p 192.168.1.95:3260
Starting iscsid:                                           [  OK  ]
192.168.1.95:3260,1 iqn.2014-04.be.linux-training:centos65.target1
```

We received **iqn.2014-04.be.linux-training:centos65.target1**.

We use this iqn to configure the username and the password (paul and hunter2) that we set on the target server.

```
[root@server2 iscsi]# iscsiadm -m node --targetname iqn.2014-04.be.linux-tra\
ining:centos65.target1 --portal "192.168.1.95:3260" --op=update --name node.\
session.auth.username --value=paul
[root@server2 iscsi]# iscsiadm -m node --targetname iqn.2014-04.be.linux-tra\
ining:centos65.target1 --portal "192.168.1.95:3260" --op=update --name node.\
session.auth.password --value=hunter2
[root@server2 iscsi]# iscsiadm -m node --targetname iqn.2014-04.be.linux-tra\
ining:centos65.target1 --portal "192.168.1.95:3260" --op=update --name node.\
session.auth.authmethod --value=CHAP
```

RHEL and CentOS will store these in **/var/lib/iscsi/nodes/**.

```
[root@server2 iscsi]# grep auth /var/lib/iscsi/nodes/iqn.2014-04.be.linux-tr\
aining\:centos65.target1/192.168.1.95\,3260\,1/default
node.session.auth.authmethod = CHAP
node.session.auth.username = paul
node.session.auth.password = hunter2
node.conn[0].timeo.auth_timeout = 45
[root@server2 iscsi]#
```

A restart of the **iscsi** service will add three new devices to our system.

```
[root@server2 iscsi]# fdisk -l | grep Disk
Disk /dev/sda: 42.9 GB, 42949672960 bytes
Disk identifier: 0x0004f229
Disk /dev/sdb: 1073 MB, 1073741824 bytes
Disk identifier: 0x00000000
Disk /dev/sdc: 1073 MB, 1073741824 bytes
Disk identifier: 0x00000000
Disk /dev/sdd: 1073 MB, 1073741824 bytes
Disk identifier: 0x00000000
Disk /dev/sde: 2147 MB, 2147483648 bytes
Disk identifier: 0x00000000
Disk /dev/sdf: 2147 MB, 2147483648 bytes
Disk identifier: 0x00000000
Disk /dev/sdg: 2147 MB, 2147483648 bytes
Disk identifier: 0x00000000
Disk /dev/mapper/VolGroup-lv_root: 41.4 GB, 41448112128 bytes
Disk identifier: 0x00000000
Disk /dev/mapper/VolGroup-lv_swap: 973 MB, 973078528 bytes
Disk identifier: 0x00000000
[root@server2 iscsi]# service iscsi restart
Stopping iscsi:                                            [  OK  ]
Starting iscsi:                                            [  OK  ]
[root@server2 iscsi]# fdisk -l | grep Disk
Disk /dev/sda: 42.9 GB, 42949672960 bytes
Disk identifier: 0x0004f229
Disk /dev/sdb: 1073 MB, 1073741824 bytes
Disk identifier: 0x00000000
Disk /dev/sdc: 1073 MB, 1073741824 bytes
Disk identifier: 0x00000000
Disk /dev/sdd: 1073 MB, 1073741824 bytes
Disk identifier: 0x00000000
Disk /dev/sde: 2147 MB, 2147483648 bytes
Disk identifier: 0x00000000
Disk /dev/sdf: 2147 MB, 2147483648 bytes
Disk identifier: 0x00000000
Disk /dev/sdg: 2147 MB, 2147483648 bytes
Disk identifier: 0x00000000
Disk /dev/mapper/VolGroup-lv_root: 41.4 GB, 41448112128 bytes
Disk identifier: 0x00000000
Disk /dev/mapper/VolGroup-lv_swap: 973 MB, 973078528 bytes
Disk identifier: 0x00000000
Disk /dev/sdh: 1073 MB, 1073741824 bytes
Disk identifier: 0x00000000
Disk /dev/sdi: 1073 MB, 1073741824 bytes
Disk identifier: 0x00000000
Disk /dev/sdj: 1073 MB, 1073741824 bytes
Disk identifier: 0x00000000
```

You can verify iscsi status with:

```
service iscsi status
```

8.4. iSCSI target on Debian

Installing the software for the target server requires **iscsitarget** on Ubuntu and Debian, and an extra **iscsitarget-dkms** for the kernel modules only on Debian.

```
root@debby6:~# aptitude install iscsitarget
The following NEW packages will be installed:
  iscsitarget
0 packages upgraded, 1 newly installed, 0 to remove and 0 not upgraded.
Need to get 69.4 kB of archives. After unpacking 262 kB will be used.
Get:1 http://ftp.belnet.be/debian/ squeeze/main iscsitarget i386 1.4.20.2-1\
 [69.4 kB]
Fetched 69.4 kB in 0s (415 kB/s)
Selecting previously deselected package iscsitarget.
(Reading database ... 36441 files and directories currently installed.)
Unpacking iscsitarget (from .../iscsitarget_1.4.20.2-1_i386.deb) ...
Processing triggers for man-db ...
Setting up iscsitarget (1.4.20.2-1) ...
iscsitarget not enabled in "/etc/default/iscsitarget", not starting...(warning).
```

On Debian 6 you will also need **aptitude install iscsitarget-dkms** for the kernel modules, on Debian 5 this is **aptitude install iscsitarget-modules-`uname -a`**. Ubuntu includes the kernel modules in the main package.

The iSCSI target server is disabled by default, so we enable it.

```
root@debby6:~# cat /etc/default/iscsitarget
ISCSITARGET_ENABLE=false
root@debby6:~# vi /etc/default/iscsitarget
root@debby6:~# cat /etc/default/iscsitarget
ISCSITARGET_ENABLE=true
```

8.5. iSCSI target setup with dd files

You can use LVM volumes (/dev/md0/lvol0), physical partitions (/dev/sda) ,raid devices (/dev/md0) or just plain files for storage. In this demo, we use files created with **dd**.

This screenshot shows how to create three small files (100MB, 200MB and 300MB).

```
root@debby6:~# mkdir /iscsi
root@debby6:~# dd if=/dev/zero of=/iscsi/lun1.img bs=1M count=100
100+0 records in
100+0 records out
104857600 bytes (105 MB) copied, 0.315825 s, 332 MB/s
root@debby6:~# dd if=/dev/zero of=/iscsi/lun2.img bs=1M count=200
200+0 records in
200+0 records out
209715200 bytes (210 MB) copied, 1.08342 s, 194 MB/s
root@debby6:~# dd if=/dev/zero of=/iscsi/lun3.img bs=1M count=300
300+0 records in
300+0 records out
314572800 bytes (315 MB) copied, 1.36209 s, 231 MB/s
```

We need to declare these three files as iSCSI targets in **/etc/iet/ietd.conf** (used to be /etc/ietd.conf).

```
root@debby6:/etc/iet# cp ietd.conf ietd.conf.original
root@debby6:/etc/iet# > ietd.conf
root@debby6:/etc/iet# vi ietd.conf
root@debby6:/etc/iet# cat ietd.conf
Target iqn.2010-02.be.linux-training:storage.lun1
 IncomingUser isuser hunter2
 OutgoingUser
 Lun 0 Path=/iscsi/lun1.img,Type=fileio
 Alias LUN1

Target iqn.2010-02.be.linux-training:storage.lun2
 IncomingUser isuser hunter2
 OutgoingUser
 Lun 0 Path=/iscsi/lun2.img,Type=fileio
 Alias LUN2

Target iqn.2010-02.be.linux-training:storage.lun3
 IncomingUser isuser hunter2
 OutgoingUser
 Lun 0 Path=/iscsi/lun3.img,Type=fileio
 Alias LUN3
```

We also need to add our devices to the **/etc/initiators.allow** file.

```
root@debby6:/etc/iet# cp initiators.allow initiators.allow.original
root@debby6:/etc/iet# >initiators.allow
root@debby6:/etc/iet# vi initiators.allow
root@debby6:/etc/iet# cat initiators.allow
iqn.2010-02.be.linux-training:storage.lun1
iqn.2010-02.be.linux-training:storage.lun2
iqn.2010-02.be.linux-training:storage.lun3
```

Time to start the server now:

```
root@debby6:/etc/iet# /etc/init.d/iscsitarget start
Starting iSCSI enterprise target service:.
.
root@debby6:/etc/iet#
```

Verify activation of the storage devices in **/proc/net/iet**:

```
root@debby6:/etc/iet# cat /proc/net/iet/volume
tid:3 name:iqn.2010-02.be.linux-training:storage.lun3
 lun:0 state:0 iotype:fileio iomode:wt blocks:614400 blocksize:\
512 path:/iscsi/lun3.img
tid:2 name:iqn.2010-02.be.linux-training:storage.lun2
 lun:0 state:0 iotype:fileio iomode:wt blocks:409600 blocksize:\
512 path:/iscsi/lun2.img
tid:1 name:iqn.2010-02.be.linux-training:storage.lun1
 lun:0 state:0 iotype:fileio iomode:wt blocks:204800 blocksize:\
512 path:/iscsi/lun1.img
root@debby6:/etc/iet# cat /proc/net/iet/session
tid:3 name:iqn.2010-02.be.linux-training:storage.lun3
tid:2 name:iqn.2010-02.be.linux-training:storage.lun2
tid:1 name:iqn.2010-02.be.linux-training:storage.lun1
```

8.6. ISCSI initiator on ubuntu

First we install the iSCSi client software (on another computer than the target).

```
root@ubu1104:~# aptitude install open-iscsi
Reading package lists... Done
Building dependency tree
Reading state information... Done
Reading extended state information
Initializing package states... Done
The following NEW packages will be installed:
  open-iscsi open-iscsi-utils{a}
```

Then we set the iSCSI client to start automatically.

```
root@ubu1104:/etc/iscsi# cp iscsid.conf iscsid.conf.original
root@ubu1104:/etc/iscsi# vi iscsid.conf
root@ubu1104:/etc/iscsi# grep ^node.startup iscsid.conf
node.startup = automatic
```

Or you could start it manually.

```
root@ubu1104:/etc/iscsi/nodes# /etc/init.d/open-iscsi start
 * Starting iSCSI initiator service iscsid                        [ OK ]
 * Setting up iSCSI targets                                       [ OK ]
root@ubu1104:/etc/iscsi/nodes#
```

Now we can connect to the Target server and use **iscsiadm** to discover the devices it offers:

```
root@ubu1104:/etc/iscsi# iscsiadm  -m discovery -t st -p 192.168.1.31
192.168.1.31:3260,1 iqn.2010-02.be.linux-training:storage.lun2
192.168.1.31:3260,1 iqn.2010-02.be.linux-training:storage.lun1
192.168.1.31:3260,1 iqn.2010-02.be.linux-training:storage.lun3
```

We can use the same **iscsiadm** to edit the files in **/etc/iscsi/nodes/**.

```
root@ubu1104:/etc/iscsi# iscsiadm -m node --targetname "iqn.2010-02.be.linu\
x-training:storage.lun1" --portal "192.168.1.31:3260" --op=update --name no\
de.session.auth.authmethod --value=CHAP
root@ubu1104:/etc/iscsi# iscsiadm -m node --targetname "iqn.2010-02.be.linu\
x-training:storage.lun1" --portal "192.168.1.31:3260" --op=update --name no\
de.session.auth.username --value=isuser
root@ubu1104:/etc/iscsi# iscsiadm -m node --targetname "iqn.2010-02.be.linu\
x-training:storage.lun1" --portal "192.168.1.31:3260" --op=update --name no\
de.session.auth.password --value=hunter2
```

Repeat the above for the other two devices.

Restart the initiator service to log in to the target.

```
root@ubu1104:/etc/iscsi/nodes# /etc/init.d/open-iscsi restart
 * Disconnecting iSCSI targets                                    [ OK ]
 * Stopping iSCSI initiator service                               [ OK ]
 * Starting iSCSI initiator service iscsid                        [ OK ]
 * Setting up iSCSI targets
```

Use **fdisk -l** to enjoy three new iSCSI devices.

```
root@ubu1104:/etc/iscsi/nodes# fdisk -l 2> /dev/null | grep Disk
Disk /dev/sda: 17.2 GB, 17179869184 bytes
Disk identifier: 0x0001983f
Disk /dev/sdb: 209 MB, 209715200 bytes
Disk identifier: 0x00000000
Disk /dev/sdd: 314 MB, 314572800 bytes
Disk identifier: 0x00000000
Disk /dev/sdc: 104 MB, 104857600 bytes
Disk identifier: 0x00000000
```

The Target (the server) now shows active sessions.

```
root@debby6:/etc/iet# cat /proc/net/iet/session
tid:3 name:iqn.2010-02.be.linux-training:storage.lun3
 sid:5348024611832320 initiator:iqn.1993-08.org.debian:01:8983ed2d770
  cid:0 ip:192.168.1.35 state:active hd:none dd:none
tid:2 name:iqn.2010-02.be.linux-training:storage.lun2
 sid:4785074624856576 initiator:iqn.1993-08.org.debian:01:8983ed2d770
  cid:0 ip:192.168.1.35 state:active hd:none dd:none
tid:1 name:iqn.2010-02.be.linux-training:storage.lun1
 sid:5066549618344448 initiator:iqn.1993-08.org.debian:01:8983ed2d770
  cid:0 ip:192.168.1.35 state:active hd:none dd:none
root@debby6:/etc/iet#
```

8.7. using iSCSI devices

There is no difference between using SCSI or iSCSI devices once they are connected : partition, make filesystem, mount.

```
root@ubu1104:/etc/iscsi/nodes# history | tail -13
   94  fdisk /dev/sdc
   95  fdisk /dev/sdd
   96  fdisk /dev/sdb
   97  mke2fs /dev/sdb1
   98  mke2fs -j /dev/sdc1
   99  mkfs.ext4 /dev/sdd1
  100  mkdir /mnt/is1
  101  mkdir /mnt/is2
  102  mkdir /mnt/is3
  103  mount /dev/sdb1 /mnt/is1
  104  mount /dev/sdc1 /mnt/is2
  105  mount /dev/sdd1 /mnt/is3
  106  history | tail -13
root@ubu1104:/etc/iscsi/nodes# mount | grep is
/dev/sdb1 on /mnt/is1 type ext2 (rw)
/dev/sdc1 on /mnt/is2 type ext3 (rw)
/dev/sdd1 on /mnt/is3 type ext4 (rw)
```

8.8. practice: iSCSI devices

1. Set up a target (using an LVM and a SCSI device) and an initiator that connects to both.

8.9. solution: iSCSI devices

1. Set up a target (using an LVM and a SCSI device) and an initiator that connects to both.

This solution was done on **Debian/ubuntu/Mint**. For RHEL/CentOS check the theory.

Decide (with a partner) on a computer to be the Target and another computer to be the Initiator.

On the Target computer:

First install iscsitarget using the standard tools for installing software in your distribution. Then use your knowledge from the previous chapter to setup a logical volume (/dev/vg/lvol0) and use the RAID chapter to setup /dev/md0. Then perform the following step:

```
vi /etc/default/iscsitarget (set enable to true)
```

Add your devices to /etc/iet/ietf.conf

```
root@debby6:/etc/iet# cat ietd.conf
Target iqn.2010-02.be.linux-training:storage.lun1
 IncomingUser isuser hunter2
 OutgoingUser
 Lun 0 Path=/dev/vg/lvol0,Type=fileio
 Alias LUN1
Target iqn.2010-02.be.linux-training:storage.lun2
 IncomingUser isuser hunter2
 OutgoingUser
 Lun 0 Path=/dev/md0,Type=fileio
 Alias LUN2
```

Add both devices to /etc/iet/initiators.allow

```
root@debby6:/etc/iet# cat initiators.allow
iqn.2010-02.be.linux-training:storage.lun1
iqn.2010-02.be.linux-training:storage.lun2
```

Now start the iscsitarget daemon and move over to the Initiator.

On the Initiator computer:

Install open-iscsi and start the daemon.

Then use **iscsiadm -m discovery -t st -p 'target-ip'** to see the iscsi devices on the Target.

Edit the files **/etc/iscsi/nodes/** as shown in the book. Then restart the iSCSI daemon and rund **fdisk -l** to see the iSCSI devices.

Chapter 9. introduction to multipathing

9.1. install multipath

RHEL and CentOS need the **device-mapper-multipath** package.

```
yum install device-mapper-multipath
```

This will create a sample multipath.conf in **/usr/share/doc/device-mapper-multipath-0.4.9/multipath.conf**.

There is no **/etc/multipath.conf** until you initialize it with **mpathconf**.

```
[root@server2 ~]# mpathconf --enable --with_multipathd y
Starting multipathd daemon:                                [  OK  ]
[root@server2 ~]# wc -l /etc/multipath.conf
99 /etc/multipath.conf
```

9.2. configure multipath

You can now choose to either edit **/etc/multipath.conf** or use **mpathconf** to change this file for you.

```
[root@server2 ~]# grep user_friendly_names /etc/multipath.conf
 user_friendly_names yes
# user_friendly_names yes
[root@server2 ~]# mpathconf --enable --user_friendly_names n
[root@server2 ~]# grep user_friendly_names /etc/multipath.conf
 user_friendly_names no
# user_friendly_names yes
[root@server2 ~]# mpathconf --enable --user_friendly_names y
[root@server2 ~]# grep user_friendly_names /etc/multipath.conf
 user_friendly_names yes
# user_friendly_names yes
```

9.3. network

This example uses three networks, make sure the iSCSI Target is connected to all three networks.

```
[root@server1 tgt]# ifconfig | grep -B1 192.168
eth1      Link encap:Ethernet  HWaddr 08:00:27:4E:AB:8E
          inet addr:192.168.1.98  Bcast:192.168.1.255  Mask:255.255.255.0
--
eth2      Link encap:Ethernet  HWaddr 08:00:27:3F:A9:D1
          inet addr:192.168.2.98  Bcast:192.168.2.255  Mask:255.255.255.0
--
eth3      Link encap:Ethernet  HWaddr 08:00:27:94:52:26
          inet addr:192.168.3.98  Bcast:192.168.3.255  Mask:255.255.255.0
```

The same must be true for the multipath Initiator:

```
[root@server2 ~]# ifconfig | grep -B1 192.168
eth1      Link encap:Ethernet  HWaddr 08:00:27:A1:43:41
          inet addr:192.168.1.99  Bcast:192.168.1.255  Mask:255.255.255.0
--
eth2      Link encap:Ethernet  HWaddr 08:00:27:12:A8:70
          inet addr:192.168.2.99  Bcast:192.168.2.255  Mask:255.255.255.0
--
eth3      Link encap:Ethernet  HWaddr 08:00:27:6E:99:9B
          inet addr:192.168.3.99  Bcast:192.168.3.255  Mask:255.255.255.0
```

9.4. start multipathd and iscsi

Time to start (or restart) both the multipathd and iscsi services:

```
[root@server2 ~]# service multipathd restart
Stopping multipathd daemon:                                [  OK  ]
Starting multipathd daemon:                                [  OK  ]
[root@server2 ~]# service iscsi restart
Stopping iscsi:                                            [  OK  ]
Starting iscsi:                                            [  OK  ]
```

This shows **fdisk** output when leaving the default friendly_names option to yes. The bottom three are the multipath devices to use.

```
[root@server2 ~]# fdisk -l | grep Disk
Disk /dev/sda: 42.9 GB, 42949672960 bytes
Disk identifier: 0x0004f229
Disk /dev/sdb: 1073 MB, 1073741824 bytes
Disk identifier: 0x00000000
Disk /dev/sdc: 1073 MB, 1073741824 bytes
Disk identifier: 0x00000000
Disk /dev/sdd: 1073 MB, 1073741824 bytes
Disk identifier: 0x00000000
Disk /dev/sde: 2147 MB, 2147483648 bytes
Disk identifier: 0x00000000
Disk /dev/sdf: 2147 MB, 2147483648 bytes
Disk identifier: 0x00000000
Disk /dev/sdg: 2147 MB, 2147483648 bytes
Disk identifier: 0x00000000
Disk /dev/mapper/VolGroup-lv_root: 41.4 GB, 41448112128 bytes
Disk identifier: 0x00000000
Disk /dev/mapper/VolGroup-lv_swap: 973 MB, 973078528 bytes
Disk identifier: 0x00000000
Disk /dev/sdh: 1073 MB, 1073741824 bytes
Disk identifier: 0x00000000
Disk /dev/sdi: 1073 MB, 1073741824 bytes
Disk identifier: 0x00000000
Disk /dev/sdj: 1073 MB, 1073741824 bytes
Disk identifier: 0x00000000
Disk /dev/sdl: 1073 MB, 1073741824 bytes
Disk identifier: 0x00000000
Disk /dev/sdn: 1073 MB, 1073741824 bytes
Disk identifier: 0x00000000
Disk /dev/sdk: 1073 MB, 1073741824 bytes
Disk identifier: 0x00000000
Disk /dev/sdm: 1073 MB, 1073741824 bytes
Disk identifier: 0x00000000
Disk /dev/sdp: 1073 MB, 1073741824 bytes
Disk identifier: 0x00000000
Disk /dev/sdo: 1073 MB, 1073741824 bytes
Disk identifier: 0x00000000
Disk /dev/mapper/mpathh: 1073 MB, 1073741824 bytes
Disk identifier: 0x00000000
Disk /dev/mapper/mpathi: 1073 MB, 1073741824 bytes
Disk identifier: 0x00000000
Disk /dev/mapper/mpathj: 1073 MB, 1073741824 bytes
Disk identifier: 0x00000000
[root@server2 ~]#
```

9.5. multipath list

You can list the multipath connections and devices with **multipath -ll**.

```
[root@server2 ~]# multipath -ll
mpathj (1IET     00010001) dm-4 Reddy,VBOX HARDDISK
size=1.0G features='0' hwhandler='0' wp=rw
|-+- policy='round-robin 0' prio=1 status=active
| `- 13:0:0:1 sdh 8:112 active ready running
|-+- policy='round-robin 0' prio=1 status=enabled
| `- 12:0:0:1 sdi 8:128 active ready running
`-+- policy='round-robin 0' prio=1 status=enabled
  `- 14:0:0:1 sdm 8:192 active ready running
mpathi (1IET     00010003) dm-3 Reddy,VBOX HARDDISK
size=1.0G features='0' hwhandler='0' wp=rw
|-+- policy='round-robin 0' prio=1 status=active
| `- 13:0:0:3 sdk 8:160 active ready running
|-+- policy='round-robin 0' prio=1 status=enabled
| `- 12:0:0:3 sdn 8:208 active ready running
`-+- policy='round-robin 0' prio=1 status=enabled
  `- 14:0:0:3 sdp 8:240 active ready running
mpathh (1IET     00010002) dm-2 Reddy,VBOX HARDDISK
size=1.0G features='0' hwhandler='0' wp=rw
|-+- policy='round-robin 0' prio=1 status=active
| `- 12:0:0:2 sdl 8:176 active ready running
|-+- policy='round-robin 0' prio=1 status=enabled
| `- 13:0:0:2 sdj 8:144 active ready running
`-+- policy='round-robin 0' prio=1 status=enabled
  `- 14:0:0:2 sdo 8:224 active ready running
[root@server2 ~]#
```

The IET (iSCSI Enterprise Target) ID should match the ones you see on the Target server.

```
[root@server1 ~]# tgt-admin -s | grep -e LUN -e IET -e dev
    LUN information:
        LUN: 0
            SCSI ID: IET     00010000
        LUN: 1
            SCSI ID: IET     00010001
            Backing store path: /dev/sdb
        LUN: 2
            SCSI ID: IET     00010002
            Backing store path: /dev/sdc
        LUN: 3
            SCSI ID: IET     00010003
            Backing store path: /dev/sdd
```

9.6. using the device

The rest is standard mkfs, mkdir, mount:

```
[root@server2 ~]# mkfs.ext4 /dev/mapper/mpathi
mke2fs 1.41.12 (17-May-2010)
Filesystem label=
OS type: Linux
Block size=4096 (log=2)
Fragment size=4096 (log=2)
Stride=0 blocks, Stripe width=0 blocks
65536 inodes, 262144 blocks
13107 blocks (5.00%) reserved for the super user
First data block=0
Maximum filesystem blocks=268435456
8 block groups
32768 blocks per group, 32768 fragments per group
8192 inodes per group
Superblock backups stored on blocks:
 32768, 98304, 163840, 229376

Writing inode tables: done
Creating journal (8192 blocks): done
Writing superblocks and filesystem accounting information: done

This filesystem will be automatically checked every 38 mounts or
180 days, whichever comes first.  Use tune2fs -c or -i to override.
[root@server2 ~]# mkdir /srv/multipath
[root@server2 ~]# mount /dev/mapper/mpathi /srv/multipath/
[root@server2 ~]# df -h /srv/multipath/
Filesystem          Size  Used Avail Use% Mounted on
/dev/mapper/mpathi 1008M   34M  924M   4% /srv/multipath
```

9.7. practice: multipathing

1. Find a partner and decide who will be iSCSI Target and who will be iSCSI Initiator and Multipather. Set up Multipath as we did in the theory.

2. Uncomment the big 'defaults' section in /etc/multipath.conf and disable friendly names. Verify that multipath can work. You may need to check the manual for **/lib/dev/scsi_id** and for **multipath.conf**.

9.8. solution: multipathing

1. Find a partner and decide who will be iSCSI Target and who will be iSCSI Initiator and Multipather. Set up Multipath as we did in the theory.

```
Look in the theory...
```

2. Uncomment the big 'defaults' section in /etc/multipath.conf and disable friendly names. Verify that multipath can work. You may need to check the manual for **/lib/dev/scsi_id** and for **multipath.conf**.

```
vi multipath.conf
```

```
remove # for the big defaults section
add # for the very small one with friendly_names active
add the --replace-whitespace option to scsi_id.

defaults {
        udev_dir                /dev
        polling_interval        10
        path_selector           "round-robin 0"
        path_grouping_policy    multibus
        getuid_callout          "/lib/udev/scsi_id --whitelisted --replace\
-whitespace --device=/dev/%n"
        prio                    const
        path_checker            readsector0
        rr_min_io               100
        max_fds                 8192
        rr_weight               priorities
        failback                immediate
        no_path_retry           fail
        user_friendly_names     no
}
```

The names now (after service restart) look like:

```
root@server2 etc]# multipath -ll
1IET_00010001 dm-8 Reddy,VBOX HARDDISK
size=1.0G features='0' hwhandler='0' wp=rw
`-+- policy='round-robin 0' prio=1 status=active
  |- 17:0:0:1 sdh 8:112 active ready running
  |- 16:0:0:1 sdi 8:128 active ready running
  `- 15:0:0:1 sdn 8:208 active ready running
1IET_00010003 dm-10 Reddy,VBOX HARDDISK
size=1.0G features='0' hwhandler='0' wp=rw
`-+- policy='round-robin 0' prio=1 status=active
  |- 17:0:0:3 sdl 8:176 active ready running
  |- 16:0:0:3 sdm 8:192 active ready running
  `- 15:0:0:3 sdp 8:240 active ready running
1IET_00010002 dm-9 Reddy,VBOX HARDDISK
size=1.0G features='0' hwhandler='0' wp=rw
`-+- policy='round-robin 0' prio=1 status=active
  |- 17:0:0:2 sdj 8:144 active ready running
  |- 16:0:0:2 sdk 8:160 active ready running
  `- 15:0:0:2 sdo 8:224 active ready running
```

Did you blacklist your own devices ?

```
 vi multipath.conf
--> search for blacklist:
add
        devnode "^sd[a-g]"
```

Index

Symbols

A

B

C

D

E

F

H

I

J

L

M

N

P

R

S

T

U

V

Z