The Linux Handbook

Version 1.6, August 2007

Copyright and liability

HP is a trademark of Hewlett-Packard Development Company, L.P.

Linux is a U.S. registered trademark of Linus Torvalds.

Itanium® and Intel® are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Red Hat is a trademark of Red Hat, Inc.

SUSE is a trademark of Novell, Inc.

Mac OS X is a trademark of Apple, Inc.

Microsoft® and Windows® are registered trademarks of Microsoft Corporation in the United States and other countries.

HP shall not be liable for technical or editorial errors or omissions contained in this document or the HP Customer Information DVD. The material contained therein is provided ”as is” without warranty of any kind. To the extent permitted by law, neither HP nor its affiliates will be liable for direct, indirect, incidental, special or consequential damages including downtime cost; damages relating to the procurement of substitute products or services; damages for loss of data; or software restoration. The information on the HP Customer Information DVD is subject to change without notice.

Preface

The Linux Handbook is directed at people who have a basic knowledge of UNIX, and are looking for the major differences between UNIX (especially HP-UX) and Linux. Our intent is not to provide another manual on Linux, or to replace the installation, configuration or administration guides which come with the various Linux distributions. We focus on those things which are common to all the different Linux distributions. Please look at this Linux Handbook as a “get to know” and “how to” guide.

The Linux handbook has been developed and is authored by Martin Körber, Christian Franck, Winfried Knobloch, and Hanul Sieger. Please send any comments and feedback to hanul.sieger@hp.com.

What is Linux?

History

Linus Torvalds started the development of Linux as a university student in 1991 to make advanced use of his new computer equipped with an Intel® i386 processor. His initial goal was not to develop another operating system, not even a new kernel, but to replace the terminal emulator of MINIX. Strictly speaking Linux itself is a free UNIX-like kernel, which has been ported to a huge variety of processor architectures (including x86, PA-RISC, Alpha, SPARC, and PowerPC). Linux on x86 is the most frequently used flavor (see www.kernel.org for the kernel sources).

Distributions

What are distributions? In addition to the free kernel you need a lot more software to get a full-featured UNIX-like operating system. Distributors like SUSE (www.SUSE.com, part of Novell since 2004) or Red Hat (www.redhat.com) take a bunch of mostly free software and bundle it with the Linux kernel to form an entire operating system that can be used right out of the box. This is a so-called Linux distribution (see www.distrowatch.com for reviews of 100+ distributions). Despite their UNIX roots, none of the Linux distributions is registered by The Open Group (the trademark holder for UNIX) as a UNIX certified operating system.[1]

Version numbers

When talking about Linux and the version number, you must specify whether you refer to the version of the distribution, e.g. SUSE Linux Enterprise Server 10 or Red Hat Enterprise Linux 4, or if you are talking about the version of the kernel used for that distribution, e.g. 2.6.16. To identify your Linux distribution, have a look at the output of '`uname`' or at the file `/etc/issue`. For SUSE examine `/etc/SUSE-release`, for Red Hat `/etc/RedHat-release`.

SUSE and Red Hat Enterprise Linux distributions

When using SUSE or Red Hat Linux for business, we must distinguish between enterprise and non-enterprise versions (or Home Office/Small Office [SOHO] editions). What are the differences?

- Release cycle: A typical release cycle for SOHO editions are 2 to 3 versions per year — too many for business use. In contrast the release cycle for the enterprise versions is about 18 months with support guarantees of usually five years.

- Support: Distributors support (if at all) their non-enterprise distributions (including bugfixes) only up to 12 months after the initial release. Both Red Hat and SUSE offer support for their enterprise editions for up to 5 years.

- Certifications: More and more 3rd party software is certified and tested for enterprise editions only, i.e. Oracle and SAP.

- Money: You have to pay a regular maintenance fee for the enterprise editions. You must register your copy of an Enterprise Linux in order to get access to password

[1] There are four UNIX 03 certified operating systems as of August 2007: HP-UX 11i V3 (HP), Mac OS X 10.5 (Apple), Solaris 10 (Sun, Fujitsu), AIX 5L (IBM). See www.opengroup.org/openbrand/register for a more detailed list and older UNIX 95 and UNIX 98 certifications.

restricted sites with the latest updates. These updates (at least in binary form) are no longer available without paying the maintenance fee.

Red Hat canceled its SOHO effort some time ago. It merged the Red Hat consumer products with an Open Source community project called Fedora (fedoraproject.org) and dropped its consumer retail efforts. Red Hat leads and maintains this project, but it is not an official Red Hat product. The goal is to build a cutting-edge Linux distribution on which the enterprise version will be based. Fedora is entirely free. A similar approach can be found for SUSE as openSUSE at www.openSUSE.org.

The current available enterprise editions from SUSE and Red Hat are

- SUSE Linux Enterprise Server (SLES), with versions 8 to 10 currently supported. Version 8 was also distributed as a joint effort with other Linux distributors under the name UnitedLinux 1.0.

- Red Hat Enterprise Linux (RHEL), with version 3 to 5 currently supported. RHEL 3 and 4 are distributed in different packages as AS ("Advanced Server"), ES ("Entry-Level Server"), WS ("Workstation"), and Desktop. RHEL 5 changed the naming convention to "advanced platform", "base server", and Desktop (with or without "workstation option"). The main differences between the packages are number of supported CPUs, RAM ceiling, and virtualization options. A comparison chart can be found at www.redhat.com/rhel/compare.

Both distributions are available for the following architectures:

- Intel® x86 and compatible, both 32-bit and 64-bit (AMD64/EM64T)

- Intel® Itanium®

- IBM Power

Delivery: In 2003 HP has signed contracts with Red Hat and SUSE that provide an easy and convenient way for the customer to order their copy of SUSE or Red Hat Enterprise Linux via HP. This all-in-one package includes the media, the maintenance and support: 1st to 3rd level support is delivered by HP Services and, if necessary, Red Hat's or SUSE's labs can be involved by HP. The collaboration between HP and Red Hat and SUSE is only available for their Linux enterprise versions.

Debian GNU/Linux

HP started to support the Debian GNU/Linux distribution in the fall of 2006. The support is for selected HP ProLiant systems and for installation only. The offer will be expanded to include full OS support in the near future.

Installation

It is not our objective to provide another installation manual. All distributions have good ones, which you should use for specific questions. We would like to focus on general issues (not specific to a certain distribution), questions, and considerations you may encounter when installing a Linux system, especially for x86 and Itanium® architectures.

Installation options

All distributions provide several installation options. Typically you will be able to

- pass special options to the kernel
- choose between text-mode and graphical installation
- choose between default/automatic and expert/manual installation

Warning! Be careful in non-manual installations. The suggested default settings (e.g. partitions, file systems) might not be useful for you. You may even lose all the data on your disks.

Installation medium

A Linux installation can be done

- via local CD-ROM/DVD-ROM or
- over the network (NFS, FTP, HTTP)

Most installations are performed using the local CD-ROM, but there may be several reasons to choose a network based installation instead:

- you do not have the installation media with you
- you do not want to be demoted as “CD-Jockey”
- you want to install several systems simultaneously

The setup of a installation file server is quite simple: Copy the content of all installation CDs into one directory. Afterwards make sure this directory is exported via NFS, or is available for HTTP or (anonymous) FTP access. (Do not forget to copy hidden files, like the .S.u.S.E-disk files stored in the root directory of the SUSE CDs.)

```
# mount /dev/hdc /CD-ROM
# mkdir -p/software/SUSE/10
# mkdir -p/software/SUSE/10
# cp -a/CD-ROM/.S* /CD-ROM/* /software/SUSE/10/
```

On the computer to be installed, make sure to load the network driver during the installation process. Afterwards assign your network parameters (either fixed or obtained via DHCP) and choose your installation server and protocol (NFS, FTP, HTTP).

Boot disks: It might be necessary for you to create boot disks to perform a network based installation without CD-ROMs. All distributions have images of the required boot disks on

their media. You can either use the dd command or rawrite on Microsoft systems to transfer these images to floppies.

Hardware detection

You do not need to worry about hardware detection these days. In a typical scenario the installation routine will detect the hardware during installation and load the appropriate drivers. If there is any trouble you can restart the installation using the manual or expert mode and load the appropriate drivers manually. (See Chapter 6 for details.)

It is sometimes useful to have a so-called "Live-CD" (Knoppix, MandrakeMove, SUSE LiveCD) with you to test hardware compatibility. For HP systems this approach will be seldom used, because our customers have Linux installed on HP-certified hardware.

Centralized deployment

Several software packages from distributors and HP are available to build a deployment repository for managing centralized installation of networked systems. Among those are:

- HP ProLiant Essentials Rapid Deployment Pack - Linux Edition (discontinued in 2005, last version is 1.30)
- HP ProLiant Essentials Rapid Deployment Pack - Windows Edition (can also deploy Linux, but is based on a Windows server)
- HP Insight Control Linux Edition for HP BladeSystem (formerly HP Control Tower for HP BladeSystem)
- HP OpenView Server Configuration Management
- Red Hat Satellite Server
- SUSE AutoYaST

Partitions

Disks are usually divided into smaller pieces, so-called partitions. The partition table found in sector 0 of the disk holds the info about the partition layout on the disk. These partitions are devices used for file system creation or integration of LVM. Current distributions have an "automatic partition" mode, which means you automatically get suggestions for essential settings like "which file system of what size on what disk mounted to what mountpoint".

The advantage is: You can install Linux without much knowledge, just choose a root password and press "Enter" a few times to accept all default values. But often these automatic modes won't fit your needs. You should enter "Expert mode" for those occasions, so you can choose the appropriate settings for your system.

Additionally, you can use the HP Smart Setup CD (provided as part of the HP ProLiant (or Integrity) Essentials Foundation Pack for Linux) to configure the system prior to installing the operating system.

Disk devices files

This is just a short overview, which device files correspond to which SCSI or IDE device. When using a Compaq Smart Array controller the device files are completely different! (See Chapter 6 for a detailed discussion.)

IDE:

`/dev/hda` – Master, primary channel
`/dev/hdb` – Slave, primary channel
`/dev/hdc` – Master, secondary channel
`/dev/hdd` – Slave, secondary channel

SCSI:

`/dev/sda` – first SCSI disk
`/dev/sdb` – second SCSI disk
`/dev/sdc` – third SCSI disk
…

Primary or extended partition, logical drive

The partitions in Linux are labeled with numbers, starting from 1. The partition's device file is the disk device file followed by the partition number, e.g. `/dev/sdb1` for the first partition on the second SCSI disk. The partition table of a PC can consist of up to 4 primary partitions. They are labeled 1 to 4. To bypass this limit of a maximum of 4 partitions per disk, one of these primary partitions can be setup as an so-called extended partition, which may be able to take up several logical drives. These logical drives are numbered starting from 5. Summary: The four primary partitions — present or not — get numbers 1-4. Logical partitions start numbering from 5.

Example: `/dev/hda1` – 1st primary partition on first IDE disk. `/dev/sdb1` – 1st primary partition of second SCSI disk. `/dev/sdb6` – 2nd logical drive on second SCSI disk. For a complete reference of device files, see `/usr/src/linux/Documentation/devices.txt`

Partition type

When creating a partition on your disk, you need to specify

- what kind of partition (primary, extended, logical)
- partition size (start and end cylinder)

The partition type is used to describe its usage, e.g. partition for DOS/Windows, for Linux, for Linux LVM. Most frequently used types are shown in the following table

No.	Type
1	FAT12
5	Extended

6	FAT16
7	HPFS/NTFS
b	Win95 FAT32
c	Win95 FAT32 (LBA)
f	Win95 Extended (LBA)
82	Linux swap
83	Linux
8e	Linux LVM
fd	Linux RAID autodetect

With this theory in mind you should be able to establish a partition table like this:

```
Disk /dev/sda: 34 heads, 62 sectors, 1012 cylinders
Units = cylinders of 2108 * 512 bytes
Device Boot Start End Blocks Id System
/dev/sda1 1 20 21049 83 Linux
/dev/sda4 21 1012 1045568 5 Extended
/dev/sda5 21 145 131719 82 Linux swap
/dev/sda6 146 1012 913787 83 Linux
```

fdisk

The Linux installation programs contain comfortable graphical user interfaces to setup the partions on your disks. Anyway you should know how to manipulate the partition table on the command line. fdisk is an interactive tool to manipulate the partition table. Always remember two things when using fdisk:

- `m` is your best friend, it prints the menu
- None of your changes is written to disk unless you press `w`

```
# fdisk /dev/sda

Command (m for help):

m Command action
a toggle a bootable flag
b edit bsd disklabel
c toggle the dos compatibility flag
d delete a partition
l list known partition types
m print this menu
n add a new partition
```

```
o create a new empty DOS partition table
p print the partition table
q quit without saving changes
s create a new empty Sun disklabel
t change a partition's system id
u change display/entry units
v verify the partition table
w write table to disk and exit
x extra functionality (experts only)
```

Let's demonstrate the usage by an example: Let's wipe out all existing partions and create new ones on `/dev/sda`: A small primary partition and two logical drives in the extended partition, one of them designated as swap partition.

Warning! Following this will wipe out your partitions and delete any data on then. Do not do this on any drive containing important data. Backup first.

```
# fdisk
/dev/sda

Command (m for help): p

Disk /dev/sda: 34 heads, 62 sectors, 1012 cylinders
Units = cylinders of 2108 * 512 bytes

   Device  Boot  Start       End  Blocks     Id   System
/dev/sda1           1      1012 1066617     5   Extended
/dev/sda5           1      1012 1066585+    6   FAT16

Command (m for help): d
Partition number (1-5): 5

Command (m for help): d
Partition number (1-5): 1

Command (m for help): p

Disk /dev/sda: 34 heads, 62 sectors, 1012 cylinders
Units = cylinders of 2108 * 512 bytes

   Device  Boot  Start       End  Blocks     Id   System

Command (m for help): n

Command action
  e extended
  p primary partition (1-4)
p

Partition number (1-4): 1
First cylinder (1-1012, default 1): 1
Last cylinder or +size or +sizeM or +sizeK (1-1012, default
```

```
1012):
+20M

Command (m for help): n

Command action
  e extended
  p primary partition (1-4)
p

Partition number (1-4): 4
First cylinder (21-1012, default 21):
Using default value 21 Last cylinder or +size or +sizeM or +sizeK
(21-1012, default 1012):
Using default value 1012

Command (m for help): n

Command action
  l logical (5 or over)
  p primary partition (1-4)
l
First cylinder (21-1012, default 21):
Using default value 21
Last cylinder or +size or +sizeM or +sizeK (21-1012, default
1012): +128M

Command (m for help): n

Command action
  l logical (5 or over)
  p primary partition (1-4)
First cylinder (146-1012, default 146):
Using default value 146
Last cylinder or +size or +sizeM or +sizeK (146-1012, default
1012):
Using default value 1012

Command (m for help): p

Disk /dev/sda: 34 heads, 62 sectors, 1012 cylinders
Units = cylinders of 2108 * 512 bytes

   Device  Boot  Start       End  Blocks     Id   System

/dev/sda1 1 20 21049 83 Linux
/dev/sda4 21 1012 1045568 5 Extended
/dev/sda5 21 145 131719 83 Linux
/dev/sda6 146 1012 913787 83 Linux

Command (m for help): t
```

```
Partition number (1-6): 5
Hex code (type L to list codes): 82
Changed system type of partition 5 to 82 (Linux swap)

Command (m for help): p

Disk /dev/sda: 34 heads, 62 sectors, 1012 cylinders
Units = cylinders of 2108 * 512 bytes

   Device  Boot  Start       End  Blocks     Id   System

/dev/sda1 1 20 21049 83 Linux
/dev/sda4 21 1012 1045568 5 Extended
/dev/sda5 21 145 131719 83 Linux swap
/dev/sda6 146 1012 913787 83 Linux
```

parted

parted is a disk partitioning and partition resizing program. It allows you to create, destroy, resize, move and copy ext2, ext3, linux-swap, FAT and FAT32 partitions. This is useful for creating space for new operating systems, reorganising disk usage, and copying data to new hard disks. parted can be used interactively like fdisk, but it can also be used non-interactively, passing commands on the command line. In interactive mode press h to get help.

Considerations

Here are some considerations regarding your setup of partitions and file systems. They are based upon our experience with current Linux distributions but may become obsolete in the future.

- 1024 cylinder limit: There was a limitation of the x86 BIOS and the bootloader. The kernel had to be stored on an area of the disk below 1024 cylinders. Therefore we recommend to setup a designated `/boot` partition for the kernel.

- LVM for `/boot`: Not yet possible, because bootloaders do not know anything about LVM.

- LVM for root file system: Possible with current distributions, used to be a problem in earlier days, where LVM was introduced to the distributions. Troubleshooting a non-bootable system gets more complicated. Not recommended for people who are new to LVM.

- Software RAID: See Chapter 8.

- File systems: Use Ext2 for `/boot`. All other parts of the operating system are put into one single root file system. Unlike HP-UX, Linux usually does not implement several partitions for `/home`, `/usr`, `/var`. We recommend using a journaling file system like Ext3 or ReiserFS for those parts.

Bootloader

A bootloader is a program that typically allows you to select, which operating system you want to use, and loads that operating system. For Linux on x86 systems most people will

either use LILO or Grub as the bootloader. Linux on Itanium® uses eLILO. For more details see Chapter 3. During the installation process you have to select a bootloader, and where it should be written to. If Linux is the only operating system on your machine, you will usually write the bootloader to the first sector of your first disk, the so-called Master Boot Record (MBR), which is the place where the BIOS/firmware looks for bootable code to boot the operating system. If you want to have several operating systems on your disk, e.g. both Microsoft Windows and Linux, refer to the appropriate How-Tos to be found at the Linux Documentation Project web site (www.linuxdoc.org).

Itanium® specifics

The most important difference in dealing with Itanium® systems is the use of EFI—Extensible Firmware Interface. EFI provides a shell and a boot manager. EFI is independent of the installed operating systems (be it Linux, HP-UX, or Windows). What you have to know for Linux is how the Linux system is implemented in the shell and the boot manager.

Intel offers a free 60-minute web-based course about the EFI shell (see https://shale.intel.com/softwarecollege/CourseDetails.asp?courseID=172).

The difference to x86-based systems is the use of an EFI partition to boot from. The Linux kernel and accompanying RAM disk have to reside on the EFI partition (with configuration files and others). The partition uses the FAT file system and is usually 100MB in size.

The installation routines from Red Hat and SUSE will create the required EFI partition automatically. If you choose to use the Smart Setup CD to create the partitions prior to the installation, be aware not to overwrite or re-create the EFI partition(s) with the distributor's installation routine or you may end up with some unused space on your disk.

Boot process

Understanding the Linux boot process

This chapter explains what happens when a Linux system starts up. Features that are specific to a Linux distribution are separated from those that are generic. Where this isn't possible the Red Hat setup is used as an example. The process consists of four "stages". This is somewhat chosen arbitrarily, but helps to get an overview:

- the "firmware" stage
- the "bootloader" stage
- the "kernel" stage
- the "init" stage

It isn't always easy to separate the firmware stage from the initial operations of the bootloader. On the PC platform, the firmware is so unintelligent that a separate (software) bootloader is required. On other platforms the firmware is quite sophisticated and may be able to load a kernel directly (i.e. PA-RISC or Itanium®). In this chapter we will concentrate on the Intel® x86 platform. The differences of the Itanium® boot process will be mentioned at the end of the chapter.

Stage 1 – firmware stage

The purpose of a bootloader is to get at least part of the operating system kernel into memory and running. After that, the kernel can take over the process. However, unless the bootloader is realized in firmware, to run the bootloader we must first retrieve it, from disk or wherever else it is stored. The purpose of the firmware stage, therefore, is to get a bootloader into memory and run it.

On the PC platform, the firmware stage (which does not depend on the operating system) is governed by the BIOS. Most modern PCs (and other types of computers, of course) can boot from floppy disk, hard disk, or CDROM. The BIOS typically provides a mechanism by which the operator can choose the devices that will be used to boot, and it will probably be prepared to try more than one if necessary. The process is slightly different for the different media types.

Bootloader on floppy disk or hard disk

This is usually the simplest situation. On a floppy disk, the first sector is reserved as the boot sector. It must contain executable program code. The BIOS loads the boot Linux boot sector into memory and then runs it. This process is largely the same whatever the hardware platform. The situation is similar for PC hard disks, except that it is conventional to divide the hard disk into partitions, and to provide a boot sector for each partition.

In the world of DOS, the boot sector was, and remains, combined with the partition table; the partition table controls how much space is allocated to each partition. In addition to the partition boot sectors there is an overall boot sector/partition table called the "master boot record" (MBR). When booting from a hard disk formatted this way, the PC BIOS loads the MBR and executes it as a boot sector; the code in the MBR will then find which partition to boot from, and load and run the boot sector from that partition.

Linux has no need to follow the convention of partitioning that is meaningful to DOS/Windows, but if the hard disk is to be used with more than one operating system then it is a good idea to. So, when booting from a hard disk the Linux bootloader can be placed in the MBR, or in a partition boot sector. In the latter case, it won't be the BIOS that will load the Linux bootloader; it will be the bootloader on the master boot record. Whether the boot disk is a hard disk or a floppy disk, the first stage of the boot process finds a boot sector, which will contain the Linux bootloader, and runs it.

Bootloader on CD-ROM

The ability to boot from a CD-ROM has been commonplace on most platforms for some years. On some platforms a bootable CD-ROM has the same structure as a bootable hard disk: a boot sector followed by a load of data. A structure like this is unworkable for PCs, owing to limitations in the BIOS specification.

Most modern PCs are, however, able to boot from a CD-ROM formatted according to the El Torito specification. This process is far more complex than it ought to be. Because the BIOS cannot cope with a full-sized bootable Linux filesystem on a CD-ROM, El Torito requires that the CD-ROM be provided with an additional bootable filesystem. This filesystem is considered to be outside the normal data area of the CD-ROM, and won't be visible if the CD-ROM is mounted as a filesystem in the usual way. In fact, although the CD-ROM itself will normally be formatted with an ISO9660 filesystem, the El Torito bootable image can be of any filesystem type.

In practice, the bootable image will be formatted as a floppy disk: a boot sector followed by a filesystem. When booting from the CD-ROM, the BIOS finds the bootable filesystem image, loads the boot sector, and makes the rest of the image available through BIOS calls just as it does for a floppy disk. As far as the bootloader is concerned, therefore, the BIOS treats a bootable CD-ROM as an ordinary CD-ROM with an embedded' bootable floppy disk. Booting from CD-ROM is therefore just like booting from a floppy disk in practice. With Linux, this embedded floppy disk is usually formatted with an ext2 filesystem. As with a floppy disk, this filesystem will either become the root filesystem for the next phase of the boot process, or will supply a new, compressed filesystem which will be loaded into memory as a "ramdisk" (see below).

Bootloader from network

Bootloader retrieved from the network are uncommon and are not considered here.

Stage 2 – bootloader stage

After the bootloader is loaded into memory it will be executed. Its job will be to get the kernel into memory and execute it. The bootloader will have to supply various vital pieces of information to the kernel, crucially the location of its root filesystem. There are a number of bootloaders available for Linux: on the PC platform we have essentially LILO and GRUB. LILO is probably the best known, and has been existing since the earliest days of Linux, but GRUB is a much more sophisticated proposition, and has replaced LILO as the default bootloader on both Red Hat and SUSE distributions.

LILO

LILO is a very rudimentary, single-stage bootloader. It has little or no knowledge of Linux, and does not understand the structure of any filesystem. Instead, it reads sectors from the disk using BIOS calls, supplying absolute sector numbers for the locations on the disk of the files it needs. Where does it get these values from? As it has no knowledge of filesystems it has no

way to figure them out at run-time, so the LILO installer has to supply them in the form of a "map" file. The LILO installer is a utility called `lilo`. This utility reads a configuration file typically `/etc/lilo.conf` and builds the map file from it. The location of the map file is then supplied to the boot sector that `lilo` installs.

The boot-loading process with LILO thus looks something like this: The firmware loads the LILO boot sector and executes it. LILO loads its map file using BIOS calls. Using the map file it finds the location of the boot message, which it displays to the console, followed by a prompt. The user selects which kernel to boot – if there's more than one – at the prompt and LILO loads the kernel using BIOS calls, based on information in the map file it loaded earlier. LILO executes the kernel, indicating where it can find its root filesystem and (if necessary) initial ramdisk.

A problem with LILO is that it can be quite tricky to use it for creating a boot sector for a system different to the one running the LILO installer `lilo`. The LILO configuration file (usually `/etc/lilo.conf`) takes the names of files and devices as its inputs, but these names are never passed through to the boot sector being created. The files and devices referenced are simply analyzed for their numerical offsets. For example, if `lilo.conf` contains the line `root=/dev/CD-ROM` and `/dev/CD-ROM` is a symbolic link to the real device file (perhaps `/dev/hdc`), it is important to understand that all `lilo` will store is the major and minor device identifiers of `/dev/hdc`. It is easy to imagine that if the bootable file system you are building contains a file called `/dev/CD-ROM`, and that is a link to, say, `/dev/hdd`, then the root filesystem will be found on `/dev/hdd`. But it won't; LILO does not understand file systems, and the names in the configuration file are simply rendered down to device IDs and file sector locations. Here's a typical configuration file `/etc/lilo.conf`. For more info refer to the man pages and the How-Tos.

```
# Modified by YaST2. Last modification on Thu Oct 3 22:30:35 2002
# LILO configuration file
# Start LILO global Section
# If you want to prevent console users to boot with
init=/bin/bash,
# restrict usage of boot params by setting a passwd and using the
option
# restricted.
#password=bootpwd
#restricted
#append = "enableapic" boot = /dev/had
#Install LILO in MBR
#compact
#faster, but won't work on all systems.
#force sane state
vga = normal
message = /boot/message
menu-scheme = Wg:kw:Wg:Wg
read-only
lba32
prompt
timeout = 80
default = linux
```

```
image = /boot/vmlinuz
      root = /dev/vg00/lvol1
      label = linux
      initrd = /boot/initrd
      append = "hdd=ide-scsi"

image = /boot/vmlinuz-2.4.16
      label = test
      root = /dev/vg00/lvol1
      initrd = /boot/initrd-2.4.16

other = /dev/hda1
      label = dos
      table = /dev/hda
```

GRUB

GRUB is a bootloader that is very different from LILO. It has a two-stage or three-stage operation, and has network boot capabilities (of course, the network boot facilities do not give you a way to get GRUB itself loaded: you'll still need network boot firmware). The additional sophistication of GRUB means that it cannot easily fit into a single boot sector. It therefore uses a multiple-stage process to load successively larger amounts into memory. In doing so it becomes able to understand filesystems, so the kernel itself, and the other files GRUB uses, can be specified dynamically at boot time; there is no need for explicit numerical maps such as the ones that LILO uses. In brief, the GRUB boot process looks like this:

- Stage1: The firmware loads the GRUB boot sector into memory. This is a standard (512 byte) boot sector and, thus far, the process is the same as for `lilo`. Encoded in the boot sector are the numerical disk block addresses of the sectors that make up the implementation of the next stage. GRUB then loads the blocks that are required for the next stage using BIOS calls.

- Stage 1.5: This name reflects the fact that, strictly speaking, it is optional; its purpose is to load the code that recognizes real file systems, and GRUB can be set to use numerical block offsets just like LILO: the code for stage 2 is loaded using BIOS calls, but with knowledge of the filesystem. Typically this code is in the file `/boot/grub/stage2`. On my system this program is about 120 kB in size; clearly we can offer far more sophisticated functionality in a program of this size than in the 5000-or-so bytes of LILO. The fact that GRUB loads its second stage as a file, and not as a list of disk sectors, is the key to its power; LILO cannot do this, so you cannot do much with it at boot time.

- Stage 2 GRUB puts up a menu of defined boot options, and presents a command-line to the operator. The command line can be used to load arbitrary files as kernels and ramdisks (because stage 2 understands file systems). Each boot option in the GRUB configuration file is expressed in terms of GRUB command-line. GRUB executes the commands entered by the operator, either from the configuration file or from the command line prompt. Typical commands are kernel, which loads a kernel into memory, `initrd`, which loads an initial ramdisk from a file, and boot. Here's an example grub configuration file `/boot/grub/menu.lst`.

```
# Modified by YaST2. Last modification on
# Sun Oct 6 12:48:32 2002 timeout 10
```

```
default 1
#0
title Windows
      root (hd0,0)
      chainloader +1
      makeactive

#1
title Linux
      kernel (hd0,1)/vmlinuz root=/dev/vg00/lvol1
      initrd (hd0,1)/initrd
```

Stage 3 – kernel stage

By the time this stage begins, the bootloader will have loaded the kernel into memory, configured it with the location of its root filesystem (and maybe other optional parameters), and loaded the initial ramdisk, if supplied. How we proceed from here depends to a large extent on whether we are using an initial ramdisk or not. So why is an initial ramdisk such a big deal? Well, the concept arose from attempts to solve the problem of fitting a fully bootable Linux system onto a single floppy disk. The problem is that a Linux system that will boot as far as giving a shell, and offering a few basic utilities, needs about 8Mb – far too much to fit onto a floppy. However, such a system will in practice compress down to about 2MB using gzip compression, so if the root filesystem could be compressed, we could get a working system on two standard floppy disks, or a single 2.88MB floppy disk. Another problem that had to be solved was that of booting from a floppy disk and then mounting a root filesystem from a device other than an IDE drive. SCSI drives were particularly problematic: if the kernel was compiled to include all the necessary drivers, it would not fit onto a floppy disk. However, the initial ramdisk technique allows the drivers to be supplied as loadable modules, which can be compressed. In outline, an initial ramdisk is a root filesystem that is unpacked from a compressed file. The boot loader will load the compressed version into memory, and then the kernel decompresses it and mounts it as the root filesystem. In this way we can get an 8MB root filesystem onto a 2.88MB file. Initial ramdisks are also useful on bootable CD-ROMs, because the bootable part of the CD-ROM is typically implemented as an "embedded" floppy disk.

Stage 3a – common kernel stage

Whether or not an initial ramdisk is used, the kernel will begin initializing itself and the hardware devices for which support is compiled in. The process will typically include the following steps.

- detect the CPU and its speed, and calibrate the delay loop
- initialize the display hardware
- probe the PCI bus and build a table of attached peripherals and the resources they have been assigned
- initialize the virtual memory management system, including the swapper `kswapd`
- initialize all compiled-in peripheral drivers; these typically include drivers for IDE hard disks, serial ports, real-time clock, non-volatile RAM, and AGP bus

Other drivers may be compiled in, but it's more and more common use to compile those drivers as stand-alone modules that are not required during this stage of the boot process. Note that drivers must be compiled in if they are needed to support the mounting of the root filesystem. If the root filesystem is an NFS share, for example, then drivers must be compiled in for NFS, TCP/IP, and low-level networking hardware. If we aren't using an initial ramdisk, then the next step is to mount the root filesystem. The kernel can then run the first true process from the root filesystem (strictly speaking, kswapd and its associates are not processes, they are kernel threads). Conventionally this process is `/sbin/init`, although the choice can be overridden by supplying the "`boot=`" parameter to the kernel at boot time. The init process runs with uid zero (i.e., as root) and will be the parent of all other processes. Note that kswapd and the other kernel threads have process IDs but, even though 24 Chapter 4. Linux Bootup they start before init, init still has process ID 1. This is to maintain the Unix convention that init is the first process.

Stage 3b – ramdisk kernel stage

This stage is only relevant if we are using an initial ramdisk. Using `initrd` is set as a boot option to the kernel as you might have guessed from seeing the LILO or GRUB configuration files. So you can turn it on and off using the boot option. But be careful with this. The system may need the modules from initrd to successfully boot.

In the case of using initrd, the kernel won't involve init, but will proceed as follows. The kernel unpacks the compressed ramdisk into a normal, mountable ramdisk. It then mounts the uncompressed ramdisk as a root filesystem. The original ramdisk memory is freed. It should be obvious that the kernel must have drivers compiled in to support whatever filesystem is in the ramdisk, as it won't be able to load any modules until the root filesystem is visible. The kernel then runs an initialization process. This process will, in general, not be the standard UNIX init, but a script that will mount the real root filesystem and then launch the next stage of the boot process.

This script is called `/linuxrc` by convention but it can be specified to the kernel using the init parameter. `/linuxrc` does whatever it needs to, in order to make the real root filesystem available, probably including loading some modules. It then mounts the new root filesystem over the top of the ramdisk filesystem. `/linuxrc` then spawns the "real" init process. It will typically do this using the exec command so that init ends up as process number 1, rather than 2.

`/linuxrc` need not mount a new root filesystem over the top of the ramdisk root, nor need it load init. These activities are simply conventions. For example, in order to boot a full Linux system from a CD-ROM, a workable proposition is to retain the initial ramdisk as the root filesystem, and have `/linuxrc` mount the CD-ROM at, say, `/usr`. This allows the root filesystem to be read-write; if we mounted the CD-ROM at `/`, the root filesystem would be read-only, and we would have to create a separate ramdisk and have a bunch of symbolic links from the CD-ROM to parts of that ramdisk. Similarly, a rescue disk – floppy or CD-ROM – would probably not want to invoke init, but simply put up a root shell.

Stage 4 – init stage

By now the kernel is loaded, memory management is running, some hardware is initialized, and the root filesystem is in place. All subsequent operations are invoked – directly or indirectly – by init. This process takes its instructions – again by default – from the file `/etc/inittab`. inittab specifies at least three important pieces of information:

- the runlevel to enter at startup
- a command to perform basic system initialization (conventionally this is `/etc/rc.sysinit`)
- the commands to run on entry to and exit from particular runlevels

The order of operations is that the initialization command (`rc.sysinit`) is run first, then the runlevel scripts. The division of work between `rc.sysinit` and the runlevel scripts is entirely a convention. If you are building a custom Linux system you do not have to follow this convention. In fact, you do not even have to run init if it does not do what you need.

Stage 4a – rc.sysinit

This script or executable is responsible for all the one-off initialization of the system. Linux distributions differ in the distribution of work between this script and the runlevel scripts but, in general, the following initialization steps are likely to be carried out here:

- Configure the system clock from the hardware clock. Set up key mappings for the console(s), mount the `/proc` filesystem, set up swap space (if there is any), mount and check "local" (i.e., non-network) filesystems, run depmod to initialize the module dependency tree. This is important because it makes it possible for `modprobe` to work. The kernel's module auto-loader refers to modules by name, not by filename. It also expects that when it tries to load a module by name, any modules on which it depends can also be loaded by name. In a custom boot set-up, you may prefer to load all your modules by filename, and not compile in the auto-loader at all. This speeds the boot process considerably. However, you will lose the flexibility of dynamically loading and unloading modules for hot-plug devices.
- Initialize and configure network interfaces. This step usually has to come after the depmod step or its equivalent, because the network drivers are likely to be loaded as modules.
- Load drivers for USB, PCMCIA, sound, etc. Again, these steps probably load or reference modules.

Stage 4b – runlevel scripts

Let's assume that we will be entering runlevel 5 which, by convention, gives us a graphical login prompt on top of the X server. A typical `inittab` will have entries like this:

```
l5:5:wait:/etc/rc.d/rc 5
x:5:respawn:/etc/X11/prefdm -nodaemon
```

The first line says that on entry to runlevel 5, invoke a script called rc, passing the argument "5". The second line says that on entry to runlevel 5, run the script `/etc/X11/prefdm -nodaemon`. `prefdm` is a script inserted by the Red Hat installer. It contains code that will launch the X display manager selected by the user, either at install time or using a configuration utility. The reason it works this way is so that configuration utilities do not have to mess about with `inittab`, which is a bad file to mess up if you want your system to keep working.

The X display manager will typically invoke the X server (the graphical display) on the local machine and give you a login screen. The script `rc` runs the start scripts in a directory for the runlevel given in inittab. Usually, runlevel N will correspond to a directory `/etc/rc.d/rcN.d`. As we've decided to enter runlevel 5, the relevant directory is `/etc/rc.d/rc5.d`. This directory will contain a (possibly large) number of scripts with names beginning with "`S`" or "`K`" followed by two digits, e.g., `S12syslog`. The digits denote the order in which the scripts are executed: The "`S`" scripts are executed in ascending numerical order on entry to the runlevel (i.e., at boot), and the "`K`" scripts are executed in descending order on exit (usually at shutdown). `rc` passes the argument "`start`" to each script at startup, and "`stop`" and shutdown. As a result, we do not really need both "`S`" and "`K`" scripts, because we can use the argument to determine whether we are starting or stopping. Thus it is a convention on Linux systems that the `K` scripts are simply symbolic links to their corresponding `S` scripts, and the `S` scripts do both startup and shutdown operations. So, for example, when entering runlevel 5, somewhere near the beginning of the `rc` process we will execute '`S12syslog start`'. On shutdown, somewhere towards the end of the shutdown process we will do '`K12syslog stop`' which is, in fact, an invocation of '`S12syslog stop`'.

Inside the script `S12syslog` – and most of the other scripts in that directory – you will find both initialization and finalization code. So what do these scripts do? Well, this depends on the runlevel, and the distribution, and any customizations you have made. A typical set of operations will include the following: Apply firewall settings to IP network interfaces, start the system logger, start the NFS portmapper, lock daemon, etc., and mount any NFS shares specified in `/etc/fstab`, start the power management daemon, initialize the auto-mounter, start the X font server etc.

The very last step in the boot process will be to run a script `S99local`. This is the conventional place to put machine-specific initialization. It is not recommended that you customize any of the initialization scripts that are supplied as part of a Linux distribution, simply because other people who may have to manage the system will have expectations about what is in them. Making arbitrary changes here will defeat these expectations. However, everybody expects to see machine-specific configuration in `S99local`.

Itanium® specifics

The bootup process is essentially the same as stage 3 and 4 on x86 systems, but differs in the firmware and bootloader stage, because of the different hardware. As mentioned in the chapter before, Itanium® systems use a different kind of "BIOS" and has the EFI.

EFI shell

Before you start to install Linux on an Itanium® system, you'll need a basic understanding of the EFI shell, what it does, and the information it can provide. The EFI shell is a console interface used to launch applications (such as the Linux installation program), load EFI protocols and device drivers, and execute simple scripts. It is similar to a DOS console and can only access media that is FAT16 (vfat) formatted. The EFI shell also contains common utilities that can be used on the EFI system partition. These utilities include edit, type, cp, rm, and mkdir. To see a list of utilities and other commands, type help at the EFI shell prompt. The EFI shell contains a bootloader called elilo.

The EFI specification defines a new model for the interface between operating systems and platform firmware. The interface consists of data tables that contain platform-related

information, plus boot and runtime service calls that are available to the operating system and its loader. Together, these provide a standard environment for booting an operating system and running pre-boot applications.

EFI Device Names: The map command can be used to list all devices and file systems that EFI can recognize. When your Itanium® system boots into the EFI shell, it probes your system in the following order:

- LS-120 drive (if it contains media)
- IDE hard drives on the primary IDE interface
- IDE hard drives on the secondary IDE interface
- SCSI hard drives on the SCSI interface
- CD-ROM drives on the IDE interface
- CD-ROM drives on the SCSI interface

To view the results of this system poll, type the following command at the EFI Shell prompt:

```
Shell>map
```

The output is listed in the order the system was probed. So, all FAT16 file systems are listed first, then IDE hard drives, then SCSI hard drives, then IDE CD-ROM drives, and finally SCSI CD-ROM drives. For example, output of the map command might look like the following device mapping table:

```
fs0 : VenHw(Unknown Device:00)/HD(Part1,Sig00000000)
fs1 : VenHw(Unknown Device:80)/HD(Part1,Sig00000000)
fs2 : VenHw(Unknown
Device:FF)/CDROM(Entry1)/HD(Part1,Sig00000000)
blk0 : VenHw(Unknown Device:00)
blk1 : VenHw(Unknown Device:00)/HD(Part1,Sig00000000)
blk2 : VenHw(Unknown Device:80)
blk3 : VenHw(Unknown Device:80)/HD(Part1,Sig00000000)
blk4 : VenHw(Unknown Device:80)/HD(Part2,Sig00000000)
blk5 : VenHw(Unknown Device:80)/HD(Part3,Sig00000000)
blk6 : VenHw(Unknown
Device:80)/HD(Part3,Sig00000000)/HD(Part1,Sig725F7772)
blk7 : VenHw(Unknown Device:FF)
blk8 : VenHw(Unknown Device:FF)/CDROM(Entry1)
blk9 : VenHw(Unknown
Device:FF)/CDROM(Entry1)/HD(Part1,Sig00000000)
```

In this example, there is an LS-120 diskette in the LS-120 drive as well as a CD-ROM in the CD-ROM drive. All the listings beginning with fs are FAT16 file systems that EFI can read. All the listings beginning with blk are block devices that EFI recognizes. Both the file systems and block devices are listed in the order they are probed. Therefore, fs0 is the system partition on the LS-120, fs1 is the system partition on the hard drive, and fs2 is the system partition on the CD-ROM.

Additional information on EFI can be found at the following URLs:

- developer.intel.com/technology/efi/index.htm
- developer.intel.com/software/idap/tech/video/efi/efi shell 3.htm

eLILO

EFI System Partition: When partitioning your hard drive for Linux, you must create a system partition that is FAT16 (vfat) formatted and has a mount point of `/boot/efi`. This partition will contain the installed Linux kernel(s) as well as the eLILO configuration file (`elilo.conf`).

The eLILO bootloader itself is in fact some sort of a GRUB loader with LILO syntax. Be aware that you do not call eLILO to have your bootloader configuration become valid and active. After saving the `elilo.conf` all changes are active. Be careful what you do here. Example for a working `elilo.conf`:

```
# cat /boot/efi (efi/redhat/elilo.conf)

default=linux

image=vmlinuz-2.4.18-e.12smp
label=linux
initrd=initrd-2.4.18-e.12smp.img
read-only
root=/dev/sda3
append="hda=ide-scsi"

image=vmlinuz-2.4.18-e.12
label=linux-up
initrd=initrd-2.4.18-e.12.img
read-only
root=/dev/sda3
append="hda=ide-scsi"
```

Managing Software with RPM

Prior to any package management system software on Linux was simply installed by extracting an archive, e.g. a tar file. Obviously this does not give you any chance of

- getting a list of installed software
- removing software easily
- version control

Red Hat introduced the Red Hat package manager (RPM) with Red Hat Linux 2.0 in 1995. RPM was quickly adopted by all major distributions, except for Debian, which has its own solution with major advantages over RPM.

RPM provides a huge number of options. Here are the most frequently used ones.

Query

`rpm -qa` lists all installed packages

`rpm -qi <package>` displays package information like name, version, description

`rpm -ql <package>` lists files in package

`rpm -qlp <package>` lists file in package, which is NOT installed yet

`rpm -qf <file>` shows which package owns <file>

`rpm -V -v <package>` verifies a package by verifying installed files regarding permissions/size etc. against the data these files should have according to the RPM database.

Installation

```
rpm -ihv <file>
```

```
rpm -ihv <ftp://user:passowrd@host/directory/file>
```

`rpm -Uhv <file>` updates the package. Use this if an older version of this package is already installed. `-h` option prints hash marks during the process, `-v` verbose output

Deinstallation

```
rpm -e <package>
```

Further options

Be very careful, when using these options! Some software may fail due to missing dependencies. If important libraries are exchanged, it could even lead to an unstable or unbootable system.

`--force`

`--nodeps` for not doing a dependency check

`--noscripts` does not execute pre- and post-install scripts

Examples

```
# rpm -qa|grep lilo
lilo-21-132

# rpm -qi lilo
Name : lilo Relocations: (not relocateable)
Version : 21 Vendor: SUSE GmbH, Nuernberg, Germany
Release : 132 Build Date: Sat Jul 29 16:29:23 2000
Install date: Wed Jan 3 09:10:06 2001 Build Host: Ohm.SUSE.de
Group : System Environment/Base Source RPM: lilo-21-132.src.rpm
Size : 631344 License: 1992-1997 Werner Almesberger
Packager : feedback@SUSE.de
Summary : LInux LOader
Description :
The LInux-LOader: LILO boots Linux from your hard drive.
It can also boot other operating systems such as MS-DOS and
OS/2, and can even boot DOS from the second hard drive.
The configuration file is /etc/lilo.conf.
The PowerPC variant can be used on new PowerMacs and CHRP
machines.
Authors:
--------
Werner Almesberger <almesber@di.epfl.ch>
PowerPC part:
Paul Mackeras <paulus@linuxcare.com.au>
Cort Dougan <cort@fsmlabs.com>
Benjamin Herrenschmidt <bh40@calva.net>
SUSE series: a

#rpm -ql lilo
/boot
/boot/boot.b
/boot/chain.b
/boot/os2_d.b
/sbin/activate
/sbin/lilo
/usr/sbin/keytab-lilo.pl
/usr/share/doc/packages/lilo
/usr/share/doc/packages/lilo/CHANGES
/usr/share/doc/packages/lilo/COPYING
/usr/share/doc/packages/lilo/INCOMPAT
/usr/share/doc/packages/lilo/README
/usr/share/doc/packages/lilo/tech.dvi
/usr/share/doc/packages/lilo/tech.ps.gz
/usr/share/doc/packages/lilo/user.dvi
/usr/share/doc/packages/lilo/user.ps.gz
/var/adm/setup/setup.liloconfig

#rpm -qf /sbin/lilo lilo-21-132 3.1.6.
```

Source RPM installation

Example of installing a source RPM package:

```
# rpm -ivh bcm5700-<version>.src.rpm
```

Build the binary RPM for the bcm5700 driver: on Red Hat

```
# cd /usr/src/redhat
# rpmbuild -bb SPECS/bcm5700.spec
```

on SUSE SLES 7 and SLES8/UnitedLinux 1.0:

```
# cd /usr/src/packages
# rpm -bb SPECS/bcm5700.spec
```

Some RPM builds (like the example above) require the kernel sources to be installed, too. Install the new RPM package. This installs the bcm5700 driver and man page.

```
# rpm -ivh RPMS/i386/bcm5700-<version>.i386.rpm
```

If an older version of bcm5700 already exists or a conflict occurs, please use the "force" command as shown below.

```
# rpm -ivh RPMS/i386/bcm5700-<version>.i386.rpm --force
```

The bcm5700.o driver will be installed in the following path:

Red Hat AS 2.1:
`/lib/modules/<kernel version>/kernel/drivers/addon/bcm5700/bcm5700.o`

SUSE SLES 8/UnitedLinux 1.0:
`/lib/modules/<kernel version>/kernel/drivers/net/bcm/bcm5700.o`

After installation, you may have to configure something or whatever the installed software requires for post-installation procedures.

Disadvantages and problems

RPM itself provides no way of resolving dependencies automatically. If package A depends on a file which is part of package B you need to install B before A or you need to specify both packages on the install command line:

```
#rpm -ihv A.rpm B.rpm.
```

A problem will now occur if RPM tells you your package cannot be installed because it needs the file XYZ as dependency. So the question is: Which package contains XYZ? Here are ways to resolve this dilemma:

• Check out on another sytem (same distribution): `rpm -qf <path>/XYZ` to get the name of the package containing file XYZ

• If you have connection to the internet simply go to www.rpmfind.net to find the package name, *but* please never download the package from any source other than your distributor. Either use your distribution CD or the ftp servers of your distributor. The name of this game is an obvious one: “dependency hell”. If you use Red Hat’s or SUSE’s built-in software/upgrade tools “up2date” and “YaST” (or YOU for YaST Online Update), you are spared the effort of having to find the dependencies yourself, but you are restricted to the software choices made be the distributors. You can findmore on in these tools below.

Alternatives

There are other package managers — most prominently Debian’s Advanced Packaging Tool APT (`apt-get`), which is also used by Ubuntu, and Gentoo’s `emerge`, which nicely ship around dependency hell by (mostly) solving any open dependencies on their own. Because it is unlikely to encounter Linux distributions at a customer’s site other than Red Hat or SUSE, there is only a very short overview of APT below.

Update and patch management

Both SUSE and Red Hat offer a tool to fetch and install new software and software updates including security patches. SUSE YaST, as expected, takes care of this under the moniker ”YOU” for ”YaST Online Update”. Red Hat has a separate tool named ”up2date”. The tools are available as graphical front ends or text or ncurses-based programs. They find the patches and updates available for your system on the Red Hat Network or SUSE database. An internet connection is required. You can change the repository to CDs or intra-network servers, if an internet connection is not available.

YOU and up2date allow for fine-grained selection of what to update and install. You can restrict them to only install the recommended security patches or to update your whole system. Navigation of the tools is easy.

Centralized software management with Red Hat’s Satellite Server

The Red Hat Satellite Server is a kind of duplication of the Red Hat network inside your company’s network. This is to allow the benefits of using up2date as an easy way to update your systems, without sacrifying required security levels. Not every system should or can have a direct connection to the internet. The Satellite Server replicates the contents of the Red Hat Network (if desired) and the clients get their updates from the Satellite Server. Only this server needs a connection to RHN (and even this can be avoided by feeding the upgrades via CDs). Other than replicating the RHN, the Satellite Server can handle the complete software management of all clients: The clients can be grouped with different update policies for each group. All actions can be scripted, too.

This comes with an extra cost, as the Satellite Server is a commercial product. You have to take a look at how you wish to control update and patch management of your Linux systems. If you have only a few systems, a Satellite Server would be overkill. If you have a good administration team, which is good at scripting, you could setup an equivalent yourself.

Debian APT

Just a few words on Debian’s tool: The Advanced Packaging Tool was introduced in 1998 and included in Debian 2.1 in 1999. The primary advantage over RPM is its ability to do dependency checks on software packages (not only direct dependencies, but in-direct ones, too). The main command-line tools are `apt` for installation, removal, build of packages. Roughly equivalent is `dpkg` (kind of a predecessor to `apt`). Front-end to these tools are `dselect`, `aptitude`, `synaptic` (graphical).

A Kernel called Linux

A kernel is a kernel is a kernel. It is not an operating system. At the heart of every operating system lies a kernel. In our special case the kernel gave the name for the whole operating system: Linux[2]. What distinguishes the kernel from the rest of an operating system? The kernel handles the resource allocation of the hardware parts and the applications that make use of it: CPU, memory, the whole I/O subsystem — hard disks, input devices, video, audio, etc. To make use of the resource allocator, an operating system needs additional tools (so-called userland tools), often a graphical system, and a desktop environment.

Main parts of the most common UNIX flavors

	Linux	*Mac OS X*	*HP-UX*
Desktop	KDE/Gnome	Aqua	CDE
Graphics/Window Manager	X11	Quartz Extreme	X11
Tools	GNU	BSD/GNU	HP/GNU
Kernel	Linux	xnu (Mach)	hpux

Not every kernel is the same

First of all we have to distinguish between plain vanilla kernel to be found on www.kernel.org and so-called distribution kernels which are in fact heavily patched kernels. The distributors (mainly SUSE and Red Hat) put a lot of patches from beta or testing source code trees "back" into the current stable kernel to enhance its features.

Kernel structure

Linux Kernel offers two ways of providing drivers: built-in drivers (like HP-UX — and in fact many other UNIX flavors — Linux can be built as a big monolithic kernel: Everything but the kitchen sink) or with modules to be loaded at runtime.

The kernel's file name is `vmlinuz` for a compressed kernel (most commonly a bzip'ed one) or `vmlinux` for an uncompressed kernel. Distributers sometimes add the version number to the name, e.g. `vmlinuz-2.4.20`.

The kernel itself resides in `/boot`. It is possible to have several kernels in the directory to boot from — often a useful feature when a newly compiled kernel refuses to boot up correctly due to a problem.

Linux allows having driver modules in a virtual file system, which it can mount during boot up. This is called `initrd` (short for initial ramdisk). The `/lib/modules/`\`uname -r\` directory (e.g. `/lib/modules/2.4.20`) contains the driver modules.

---

[2] Although Richard Stallman, founder of the Free Software Foundation and developer of the first GNU tools — gcc and emacs — strongly disagrees. He demands the operating system to be called GNU/Linux.

Modules

Module handling is done by `kmod` (kernel module handler), modules are loaded on demand and unloaded after a certain time when not needed.

Handling and configuration

Modules to be taken into account are listed in `/etc/modules.conf`. As an example we add a network card driver to the kernel. The module is available for the kernel we are currently using and is named is `winbond-840.o`, module `mii.o` is needed also and available, `mii` is the "Media Independent Interface" required to set the speed and duplex mode for network cards. Tasks to be done:

- place the modules into the appropriate modules directory

(here: `/lib/modules/linux-2.4.20/kernel`)

- add the `winbond` module to the configuration file as `eth0`
- create the module dependency file

In detail:

Add `winbond` to the module configuration file, we did not specify `mii` here as we'll see later:

```
# cat /etc/modules.conf
############################################################
# Aliases - specify your hardware
############################################################
alias eth0 winbond-840
# End modules.conf
```

Create the module dependency file:

```
# depmod -av
/lib/modules/2.4.20/kernel/drivers/net/mii.o
/lib/modules/2.4.20/kernel/drivers/net/winbond-840.o
```

Have a look at the module dependencies:

```
# cat /lim/modules/2.4.20/modules.dep
/lib/modules/2.4.20/kernel/drivers/net/winbond-840.o:
/lib/modules/2.4.20/kernel/drivers/net/mii.o
```

We see that `winbond` needs `mii` as specified here. Load the module into the kernel:

```
# modprobe winbond-840
Using /lib/modules/2.4.20-lfs.new/kernel/drivers/net/winbond-
840.o
winbond-840.c:v1.01-d (2.4 port) Nov-17-2001
Donald Becker <becker@scyld.com>
http://www.scyld.com/network/drivers.html
```

```
PCI: Found IRQ 15 for device 00:0a.0
eth0: Compex RL100-ATX at 0xf08af000, 00:80:48:dd:2b:b8, IRQ 15.
eth0: MII PHY 01807731h found at address 1,
status 0x786d advertising 01e1.
```

List all loaded kernel modules:

```
# lsmod
Module Size Used by Not tainted
winbond-840 15088 1 (autoclean)
mii 2560 0 (autoclean) [winbond-840]
loop 9912 1 (autoclean)
```

The `mii` module is used by `winbond-840`. Next we discuss module removal and possible pitfalls. Remove the modules from kernel and relist the modules:

```
# rmmod winbond-840
# rmmod mii
# lsmod
Module Size Used by Not tainted
loop 9912 1 (autoclean)
```

Re-insert the winbond module

```
# insmod winbond-840:
/lib/modules/2.4.20-lfs.new/kernel/drivers/net/winbond-840.o:
unresolved symbol mii_nway_restart_Rsmp_38badc13
/lib/modules/2.4.20-lfs.new/kernel/drivers/net/winbond-840.o:
unresolved symbol mii_ethtool_sset_Rsmp_016c45d9
/lib/modules/2.4.20-lfs.new/kernel/drivers/net/winbond-840.o:
unresolved symbol mii_link_ok_Rsmp_965e2742
/lib/modules/2.4.20-lfs.new/kernel/drivers/net/winbond-840.o:
unresolved symbol mii_ethtool_gset_Rsmp_fa204a84 Using
/lib/modules/2.4.20-lfs.new/kernel/drivers/net/winbond-840.o
```

We get an error and `winbond` is not loaded as we see here:

```
# lsmod
Module Size Used by Not tainted
loop 9912 1 (autoclean)
The problem is insmod only inserts the specified module while
modprobe is
respecting module dependencies, so insmod is complaining about
symbols
which reside in module mii.o not being loaded either.
Query winbond module information:
# modinfo winbond-840
filename: /lib/modules/2.4.22/kernel/drivers/net/winbond-840.o
description: "Winbond W89c840 Ethernet driver"
author: "Donald Becker becker@scyld.com"
license: "GPL"
parm: max_interrupt_work int, description
```

```
"winbond-840 maximum events handled per interrupt"
parm: debug int, description "winbond-840 debug level (0-6)"
parm: rx_copybreak int, description
"winbond-840 copy breakpoint for copy-only-tiny-frames"
parm: multicast_filter_limit int, description
"winbond-840 maximum number
of filtered multicast addresses"
parm: options int array (min = 1, max = 8),
description "winbond-840:
Bits 0-3: media type, bit 17: full duplex"
parm: full_duplex int array (min = 1, max = 8),
description "winbond-840 full duplex setting(s) (1)"
```

You do not have to specify the entire module path as long as the module is known after a run of depmod. Summary of commands for module handling:

`depmod -av` creates module dependencies

`insmod [-f] <module>` inserts module

`modprobe <module>` inserts module including dependencies

`rmmod <module>` removes module

`lsmod` lists loaded modules

Warning! `insmod -f` forces module load even if module does not fit into the kernel, use it with care.

Kernel build

Each distribution provides the kernel source to build a custom kernel. If the sources are not installed they have to be installed from CD-ROM or online repositories. To compile the kernel sources a `.config` file is required, which determines the components to be included in the kernel as either built into the kernel (_) or module (`M`). Red Hat provides prepared `.config` files in `/usr/src/config`, these files cover different processor types. A suitable configuration file has to be copied to `/usr/src/linux`. SUSE kernels have a patch which allows you to write the running kernel's configuration to a file.

```
# zcat /proc/config.gz > /usr/src/linux/config
```

Details of the kernel build and installation

```
# cd /usr/src/linux/config
# make distclean
```

This removes any debris left from earlier builds from the kernel tree, `.config` files are deleted also. Any `.config` stored here must be saved elsewhere.

```
# make menuconfig
```

This has to be used when configuration changes have to be made. When saving your `.config`, it resides in `/usr/src/linux`, so it may be a good idea to backup an existing `.config` file. Example: add the Tekram SCSI controller as a module NCR53C8 for the built-in SCSI controller (boot controller):

```
<*> NCR53C8XX SCSI support
(4) default tagged command queue depth
(32) maximum number of queued commands
(20) synchronous transfers frequency in MHz
<M> Tekram DC395/U/UW and DC315/U SCSI support

# make oldconfig
```

This is useful when using an existing `.config` without doing any modification to the configuration. If you upgrade a kernel version or simply want to build a custom kernel from an existing `.config` file use the above command.

```
# cp Makefile Makefile.bak
# head -4 Makefile
VERSION = 2 3
PATCHLEVEL = 4
SUBLEVEL = 20
EXTRAVERSION =
```

You should modify the Makefile and add some text to the EXTRAVERSION = entry, e.g. EXTRAVERSION = -custom. This leads to the creation of an independent kernel module custom branch during module installation. The already existing module branch is not touched.

```
# cp Makefile Makefile.bak
# head -4 Makefile
VERSION = 2 3
PATCHLEVEL = 4
SUBLEVEL = 20
EXTRAVERSION = -custom

# make dep
make[1]: Entering directory
/usr/src/linux-2.4.20/arch/i386/boot' make[1]: Nothing to be done
for dep'. make[1]: Leaving directory /usr/src/linux-
2.4.20/arch/i386/boot' scripts/mkdep -- init/*.c >
.depend scripts/mkdep -- find /usr/src/linux-2.4.20/include/asm
/usr/src/linux-2.4.20/include/linux
/usr/src/linux-2.4.20/include/scsi
/usr/src/linux-2.4.20/include/net
/usr/src/linux-2.4.20/include/math-emu \ ( -name SCCS -o -name
.svn \) -prune -o -follow -name \*.h ! -name modversions.h -print
> .hdepend
[...]
```

This step determines the dependencies in the kernel build.

```
# make clean
```

Cleaning up former kernel build files (_.a, _.o, etc.).

```
# make bzImage
```

This makes the kernel itself.

```
# make modules
```

This makes all kernel modules.

```
# make modules_install
```

This installs all kernel modules in the path /lib/modules/2.4.20-custom. The module branch name can be altered by modifying the EXTRAVERSION entry in the kernel Makefile.

```
# cp /usr/src/linux/arch/i386/boot/bzImage /boot/vmlinuz-2.4.20-custom
```

Copy the newly made kernel into the boot directory and rename it. It is also a good idea to name the initrd file accordingly, so kernel and initrd have a common naming scheme. You have to edit the boot loader configuration file to create an entry for the new kernel. When using lilo you have to call /sbin/lilo before rebooting.

Recent changes to the kernel build process

Current distributions using the 2.6 kernel employ a slightly altered build process. In essence it has been shortened to

```
# cd /usr/src/linux/ for SUSE

# cd /usr/src/kernels/<version>/ for Red Hat

# make distclean

# make menuconfig

# make
```

Copy the new kernel (bzImage) to /boot and rename it.

Sometimes the /lib/modules directory is not populated during the make process. Check the directory, if it contains a sub-directory for the new kernel. If it is missing, the modules can be build seperately as before with

```
# make modules modules_install
```

Unfortunately, there is no message indicating this error. If the modules are missing, you will encounter the error at the latest while making the initial RAM disk.

initrd

The advantage of this concept is that the ramdisk file can easily be changed without touching the kernel itself. A good example may be the introduction of a new SCSI controller into a fully setup Linux system that should boot from it. Replacing the SCSI driver in the initrd

with the new module and re-making the initrd file is all that has to be done for the kernel setup. No bloat "generic kernel" is needed which has included everything.

Distributors like SUSE and Red Hat provide initrd files ready to use with their installed stock kernels, so all should work out of the box. But it may be necessary to modify the initrd file. initrd provides the capability to load a RAM disk as a boot option. This RAM disk is mounted as the root file system and programs can be run from it. Afterwards, a new root file system can be mounted from a different device. The previous root (from initrd) is then moved to a directory and can be subsequently unmounted. initrd is mainly designed to allow system startup to occur in two phases, where the kernel comes up with a minimum set of compiled-in drivers, and where additional modules are loaded from initrd. When the initial ramdisk is used the filesystem image is copied into memory and mounted so that the files on it can be accessed. A program on this ramdisk (called /linuxrc) is run and when it is finished a different device is mounted as the root filesystem. Lets have a closer look at the /boot directory (we find more files in there, but let us concentrate on the kernel and ramdisk). Example:

```
#ls /boot
...
vmlinuz-2.4.20
initrd-2.4.20
...
```

When using initrd, the system typically boots as follows, (see also Chapter 3):

- the boot loader loads the kernel and the initial RAM disk
- the kernel converts initrd into a "normal" RAM disk and frees the

memory used by initrd

- initrd is mounted read-write as root
- linuxrc is executed (this can be any valid executable, including shell scripts; it is run with uid 0 and can do basically everything init can do)
- linuxrc mounts the "real" root file system
- linuxrc places the root file system at the root directory using the pivot root system call
- the usual boot sequence (e.g. invocation of /sbin/init) is performed on the root file system
- the initrd file system is removed

Note: changing the root directory does not involve un-mounting it. It is therefore possible to leave processes running on initrd during that procedure.

Also note that file systems mounted under `initrd` continue to be accessible. Let's examine the `initrd` file. Given the fact that `initrd` is a compressed image file we have to uncompress it first.

```
# gzip -d < /boot/initrd-2.4.20 > /boot/initrd-2.4.20uc
# mount -o loop /boot/initrd-2.4.20/uc /mnt
# ls /mnt
. .. bin dev etc lib linuxrc mnt proc
```

With Red Hat Enterprise Linux 4 Update 3 and SUSE Linux Enterprise Server 10 the `initrd` file changed from an image file to a compressed `cpio` archive.

```
# mkdir /tmp/initrd
# cp /boot/initrd-2.6.9-42.0.2.EL.img /tmp/initrd
# cd /tmp/initrd
# gzip -d < initrd-2.6.9-42.0.2.EL.img > initrd-2.6.9-42.0.2.EL
# cpio -i --make-directories < initrd-2.6.9-42.0.2.EL
# ls
. .. bin dev etc init lib loopfs proc sbin sys sysroot
```

All files and directories needed to run `/linuxrc` are available:

```
# cat /mnt/linuxrc
#! /bin/sh export PATH=/bin
echo "Loading module jbd ..." insmod jbd
echo "Loading module ext3 ..." insmod ext3
```

What happens:

- `linuxrc` loads the necessary kernel modules
- `linuxrc` creates and populates the root file system
- `linuxrc` invokes pivot root to change the root file system and executes — via `chroot` — a program that continues the startup.

With the new `initrd` format the script (named `init`) also changed a bit:

```
#!/bin/nash
mount -t proc /proc /proc
setquiet
echo Mounted /proc filesystem
echo Mounting sysfs
mount -t sysfs none /sys
echo Creating /dev
mount -o mode=0755 -t tmpfs none /dev
mknod /dev/console c 5 1
mknod /dev/null c 1 3
mknod /dev/zero c 1 5
mkdir /dev/pts
mkdir /dev/shm
```

```
echo Starting udev
/sbin/udevstart
echo -n "/sbin/hotplug" > /proc/sys/kernel/hotplug
echo "Loading scsi_mod.ko module"
insmod /lib/scsi_mod.ko
echo "Loading sd_mod.ko module"
insmod /lib/sd_mod.ko
echo "Loading cciss.ko module"
insmod /lib/cciss.ko
echo "Loading dm-mod.ko module"
insmod /lib/dm-mod.ko
echo "Loading jbd.ko module"
insmod /lib/jbd.ko
echo "Loading ext3.ko module"
insmod /lib/ext3.ko
/sbin/udevstart
echo Creating root device
mkrootdev /dev/root
umount /sys
echo Mounting root filesystem
mount -o defaults --ro -t ext3 /dev/root /sysroot
mount -t tmpfs --bind /dev /sysroot/dev
echo Switching to new root
switchroot /sysroot
umount /initrd/dev
```

`init` does essentially the same as `linuxrc` before.

Building initrd

Linux distributions provide shell scripts to help making an appropriate `initrd` file.

- SUSE: `/sbin/mk initrd`
- Red Hat: `/sbin/mkinitrd`

SUSE provides a symbolic link `mk_initrd` in `/sbin` pointing to `mkinitrd` to maintain compatibility to Red Hat, but the syntax is different. A `/sbin/mkinitrd -h /boot/initrd` by calling `/sbin/mkinitrd` without any further options. Backup `/boot/initrd` files before.

The Modules have to be specified in `/etc/sysconfig/kernel` for installation in the initrd file.

```
# cat /etc/sysconfig/kernel
#
# This variable contains the list of modules to be added to
# the initial ramdisk by calling the script "mk_initrd"
# (like drivers for scsi-controllers, for lvm or reiserfs)
#
INITRD_MODULES="jbd ext3"
```

Example of a mk initrd call (SUSE). The kernel resides in /boot, mk initrd will put initrd-2.4.20 in the same directory.

```
# mk_initrd -k vmlinuz-2.4.20 -i initrd-2.4.20
using "/dev/sda1" as root device (mounted on "/" as "ext3")

creating initrd "//boot/initrd-2.4.20" for kernel
"//boot/vmlinuz-2.4.20" (2.4.20)
module jbd is "/lib/modules/2.4.20/kernel/fs/jbd/jbd.o"
-> insmod jbd
module ext3 is "/lib/modules/2.4.20/kernel/fs/ext3/ext3.o"
-> insmod ext3
```

Another example of `mk_initrd`, this time with an additional module (SUSE). We add the `aic7xxx` module for an Adaptec SCSI controller.

```
# cat /etc/sysconfig/kernel
#
# This variable contains the list of modules to be added to
# the initial ramdisk by calling the script "mk_initrd"
# (like drivers for scsi-controllers, for lvm or reiserfs)
#
INITRD_MODULES="aic7xxx jbd ext3"

# mk_initrd -k vmlinuz-2.4.22 -i initrd-2.4.22
using "/dev/sda1" as root device (mounted on "/" as "ext3")
creating initrd "//boot/initrd-2.4.22"
for kernel "//boot/vmlinuz-2.4.22" (2.4.22)
module                          aic7xxx                          is
"/lib/modules/2.4.22/kernel/scsi/aic7xxx/aic7xxx.o"
-> insmod aic7xxx
module jbd is "/lib/modules/2.4.22/kernel/fs/jbd/jbd.o"
-> insmod jbd
module ext3 is "/lib/modules/2.4.22/kernel/fs/ext3/ext3.o"
-> insmod ext3
```

Red Hat's `mkinitrd` works a little bit different. When launched it takes into account a set of default module types which are inserted into RAM disk file depending on the current system configuration.

These module types are:

- file system drivers for / file system and dependencies (e.g. ext3, ReiserFS)
- SCSI driver and dependencies for boot controller
- logical volume driver (LVM) if needed

Example of an `mkinitrd` call (Red Hat): The kernel resides in `/boot` with `initrd-2.4.20-20.8bigmem.img`

```
# mkinitrd -v /boot/initrd-2.4.20-20.8bigmem.img 2.4.20-
20.8bigmem
Using modules: ./kernel/fs/jbd/jbd.o ./kernel/fs/ext3/ext3.o
Using loopback device /dev/loop0
/sbin/nash -> /tmp/initrd.qYCBYw/bin/nash
/sbin/insmod.static -> /tmp/initrd.qYCBYw/bin/insmod
‘/lib/modules/2.4.20-20.8bigmem/./kernel/fs/jbd/jbd.o’
-> ‘/tmp/initrd.qYCBYw/lib/jbd.o’
‘/lib/modules/2.4.20-20.8bigmem/./kernel/fs/ext3/ext3.o’
-> ‘/tmp/initrd.qYCBYw/lib/ext3.o’
Loading module jbd Loading module ext3
```

Example of an `mkinitrd` call (Red Hat) with additional module: We add the `aic7xxx` module for an Adaptec SCSI controller.

```
# mkinitrd -v --with=aic7xxx /boot/initrd-2.4.20-20.8bigmem.img \
2.4.20-20.8bigmem

Using modules: ./kernel/fs/jbd/jbd.o ./kernel/fs/ext3/ext3.o
./kernel/drivers/scsi/aic7xxx/aic7xxx.o
Using loopback device /dev/loop0
/sbin/nash -> /tmp/initrd.J0tfIr/bin/nash
/sbin/insmod.static -> /tmp/initrd.J0tfIr/bin/insmod
‘/lib/modules/2.4.20-20.8bigmem/./kernel/fs/jbd/jbd.o’
-> ‘/tmp/initrd.J0tfIr/lib/jbd.o’
‘/lib/modules/2.4.20-20.8bigmem/./kernel/fs/ext3/ext3.o’
-> ‘/tmp/initrd.J0tfIr/lib/ext3.o’
‘/lib/modules/2.4.20-
20.8bigmem/./kernel/drivers/scsi/aic7xxx/aic7xxx.o’
-> ‘/tmp/initrd.J0tfIr/lib/aic7xxx.o’
Loading module jbd Loading module ext3 Loading module aic7xxx
```

Hardware components

This section explains how to detect and access typical hardware components of your Linux system.

Detecting Hardware

If you are familiar with HP-UX you will miss having a Linux command like ioscan in HP-UX, which lists all the hardware attached to a system. But do not worry, in most cases detecting and configuring hardware is quite easy with Linux, because all major distributions have a good auto-detection routine during installation. For an already installed system exists a startup script to look for new hardware and to configure it if required. (Red Hat: kudzu, SUSE: hwinfo). If manual interaction is required you should be aware of these general facts:

- A low-level hardware driver is required to see attached devices, e.g. without the appropriate driver for your SCSI controller, you will not see attached devices to this SCSI bus.
- If you try to load a kernel module, which does not detect hardware supported by this module, you will receive the error message: `init module: No such device` Hint: `insmod` errors can be caused by incorrect module parameters, including invalid I/O or IRQ parameters. You may find more information in `syslog` or the output from `dmesg`
- Information about detected devices are provided by low-level drivers and can typically be found in kernel messages (`dmesg`) or in the `/proc` filesystem.
- PCI devices can be listed using the command `lspci` or by executing `cat /proc/pci`.

```
# lspci
00:00.0 Host bridge: ServerWorks CNB20LE Host Bridge (rev 06)
00:00.1 Host bridge: ServerWorks CNB20LE Host Bridge (rev 06)
00:02.0 RAID bus controller: American Megatrends Inc. MegaRAID (rev 20)
00:03.0 Ethernet controller: Intel Corp. 82557/8/9 [Ethernet Pro 100] (rev 08)
00:04.0 Ethernet controller: Intel Corp. 82557/8/9 [Ethernet Pro 100] (rev 08)
00:05.0 VGA compatible controller: ATI Technologies Inc 3D Rage IIC (rev 7a)
0:0f.0 ISA bridge: ServerWorks OSB4 South Bridge (rev 50)
00:0f.1 IDE interface: ServerWorks OSB4 IDE Controller
03:06.0 SCSI storage controller: LSI Logic / Symbios Logic (formerly NCR) \
53c1010 Ultra3 SCSI Adapter (rev 01)
03:06.1 SCSI storage controller: LSI Logic / Symbios Logic (formerly NCR) \
53c1010 Ultra3 SCSI Adapter (rev 01)
```

The `/proc` filesystem is a pseudo or virtual filesystem used as an interface to kernel data structures. Most of it is read-only, some parts are write-able to perform online changes. The filesystem doesn't need any real space on your disk.

Typical hardware components

CPU

The Linux kernel can be built with or without support for symmetric multiprocessing (SMP). For a one-CPU system use the non-SMP kernel, otherwise select the SMP kernel. The numbers and types of your CPUs can be seen at `/proc/cpuinfo`. Example:

```
# cat /proc/cpuinfo
processor : 0
vendor_id : AuthenticAMD
cpu family : 6
model : 4
model name : AMD Athlon(tm)
Processor stepping : 4
cpu MHz : 1327.702
cache size : 256 KB
fdiv_bug : no
hlt_bug : no
f00f_bug : no
coma_bug : no
fpu : yes
fpu_exception : yes
cpuid level : 1
wp : yes
flags : fpu vme de pse tsc msr pae mce cx8 sep mtrr pge mca cmov pat pse36 mmx
fxsr syscall mmxext 3dnowext 3dnow
bogomips : 2647.65
```

Memory

The amount of memory detected is shown as one of the first kernel messages during the bootup process.

```
#dmesg | less
Linux version 2.4.19-4GB (root@Athlon.SUSE.de) gcc version 3.2)
#1 Fri Sep 13 13:19:15 UTC 2002
BIOS-provided physical RAM map:
BIOS-e820: 0000000000000000 - 000000000009fc00 (usable)
BIOS-e820: 000000000009fc00 - 00000000000a0000 (reserved)
BIOS-e820: 00000000000f0000 - 0000000000100000 (reserved)
BIOS-e820: 0000000000100000 - 000000001fff0000 (usable) BIOS-e820:
000000001fff0000 - 000000001fff8000 (ACPI data) BIOS-e820:
000000001fff8000 - 0000000020000000 (ACPI NVS) BIOS-e820:
00000000fec00000 - 00000000fec01000 (reserved) BIOS-e820:
00000000fee00000 - 00000000fee01000 (reserved) BIOS-e820:
00000000ffee0000 - 00000000fff00000 (reserved) BIOS-e820:
00000000fffc0000 - 0000000100000000 (reserved)
511MB LOWMEM available.
...
Memory: 513644k/524224k available (1685k kernel code, 10192k reserved,
413k data, 164k init, 0k (highmem)
...
```

`/proc/meminfo` brings up details regarding memory and its usage. **Note**: The kernel can be forced to use less memory than automatically detected. This is a useful option to simulate a system with less memory without removing a memory module physically. In those cases the option `mem=nn[KMG]` was passed to the kernel, which can be checked by executing `cat /proc/cmdline`.

IDE

Typically the IDE driver is a static part of the kernel, not used as a module. This is due to the fact that nearly every system uses an IDE device, at least an IDE CD-ROM. As already shown in Chapter 2 the device files are:

`/dev/hda` – Master, primary channel

/dev/hdb – Slave, primary channel

/dev/hdc – Master, secondary channel

/dev/hdd – Slave, secondary channel

Information about the attached devices and its settings are found as part of the kernel messages during bootup (dmesg) and in /proc/ide/. Settings: The current Linux distributions typically have the DMA mode enabled for the IDE channels. But some combinations of IDE chip-sets and their drivers may experience problems when operating in DMA-mode. To disable DMA mode, which means to use PIO, you can

- run hdparm
- pass an option to the kernel: ide=nodma

SCSI

For details refer to the SCSI-2.4-HOWTO as part of the LDP (www.linuxdoc.org) and take a look at the files /usr/src/linux/Documentation/devices.txt and /usr/src/linux/Documentation/cciss.txt.

As soon as the hardware specific low-level driver for your SCSI controller is loaded the attached SCSI devices show up in the kernel messages (dmesg). In addition /proc/scsi is populated.

Tape support: To access a SCSI-Tape you need to enable the SCSI tape support in the kernel. Usually this will be done using the kernel module called st.

Device files: The scan algorithm scans all host adapters, all channels, all SCSI IDs and all LUNs in a nested loop. SCSI device files (/dev/sd for disks) are assigned in the order of detecting the device. So there is a good chance to mix up your device files when doing changes to your SCSI bus, like adding another disk, changing IDs, adding another adapter.

Driver cciss for Compaq Smart Array controllers: This driver is a block device driver, thus the attached devices do not show up in the SCSI layer!

/dev/sda – First SCSI disk

/dev/sdb – Second SCSI disk

/dev/sdc – Third SCSI disk

. . .

/dev/st0 – First SCSI tape (rewinding)

/dev/nst0 – First SCSI tape (non-rewinding)

. . .

/dev/cciss/c0d0 – Controller 0, disk 0, whole device

`/dev/cciss/c0d0p1` – Controller 0, disk 0, partition 1

`/dev/cciss/c1d2p3` – Controller 1, disk 2, partition 3

Example, excerpts taken from `dmesg` output:

```
scsi0 : LSI Logic MegaRAID H01.08 254 commands 15 targs 4 chans 7 luns
blk: queue c35efa18, I/O limit 4095Mb (mask 0xffffffff)
scsi0: scanning virtual channel 0 for logical drives.
Vendor: MegaRAID Model: LD0 RAID0 34730R Rev: H
Type: Direct-Access ANSI SCSI revision: 02
blk: queue c35efc18, I/O limit 4095Mb (mask 0xffffffff)
scsi0: scanning virtual channel 1 for logical drives.
scsi0: scanning virtual channel 2 for logical drives. #
scsi0: scanning physical channel 0 for devices.
Vendor: HP Model: SAFTE; U160/M BP Rev: 1023
Type: Processor ANSI SCSI revision: 02
blk: queue f79c9618, I/O limit 4095Mb (mask 0xffffffff)
Attached scsi disk sda at scsi0, channel 0, id 0, lun 0
SCSI device sda: 71127040 512-byte hdwr sectors (36417 MB)
Partition check:
sda: sda1 sda2 sda4 < sda5 >
...
sym53c8xx: at PCI bus 3, device 6, function 0
sym53c8xx: 53c1010-33 detected with Symbios NVRAM
sym53c8xx: at PCI bus 3, device 6, function 1
sym53c8xx: setting PC_COMMAND_MASTER...(fix-up)
sym53c8xx: 53c1010-33 detected with Symbios NVRAM
sym53c8xx: 53c1010-33 state OFF thus not attached
sym53c1010-33-0: rev 0x1 on pci bus 3 device 6 function 0 irq 24
sym53c1010-33-0: Symbios format NVRAM, ID 7, Fast-80, Parity Checking
sym53c1010-33-0: on-chip RAM at 0xfd010000
sym53c1010-33-0: restart (scsi reset).
sym53c1010-33-0: handling phase mismatch from SCRIPTS.
sym53c1010-33-0: Downloading SCSI SCRIPTS.
scsi1 : sym53c8xx-1.7.3c-20010512
blk: queue f7992618, I/O limit 1048575Mb (mask 0xfffffffffff)
Vendor: HP Model: C5683A Rev: C005
Type: Sequential-Access ANSI SCSI revision: 02
blk: queue f7992018, I/O limit 1048575Mb (mask 0xfffffffffff)
Vendor: HP Model: DVD-ROM 305 Rev: 1.01
Type: CD-ROM ANSI SCSI revision: 02
...
st: Version 20020205, bufsize 32768, wrt 30720, max init. bufs 4, s/g segs 16
Attached scsi tape st0 at scsi1, channel 0, id 2, lun 0
sym53c1010-33-0-<2,*>: FAST-20 WIDE SCSI 40.0 MB/s (50.0 ns, offset 31)
```

Ethernet

Ethernet devices are labeled `eth0`, `eth1`, ... Their setup can be done using `ifconfig` command.

USB

USB devices are rareley used on server systems, except for the new Itanium® systems with USB keyboard and mouse, and for the occasional USB stick for backup or boot purposes. `lsusb` and again `/proc` provide useful information about attached devices.

Serial and parallel ports

Serial ports may get interesting if you would like to attach a serial console. As usual the kernel driver for serial ports will detect the ports, results can be checked by `dmesg`. The first serial port can be accessed via `/dev/ttyS0`, the second one via `/dev/ttyS1`.

Most frequently used components

This section lists the most frequently used SCSI and Ethernet controller types and their drivers. For a comprehensive list please refer to the HPHOWTO as part of the Linux Documentation Project (www.linuxdoc.org).

HP specific tools

In addition to the above described generic information HP provides free additional software to inspect and monitor components of HP systems, like ProLiants or the obsolete NetServers. For NetServers use the Instant Top Tools, for ProLiant systems the health agents (System Insight Manager). Both are available from the HP Linux Website (linux.hp.com).

Frequently used components

Category	chip, output of lspci command	Driver
SCSI	SCSI storage controller: LSI Logic/Symbios Logic (formerly NCR) 53c1010 Ultra3 SCSI Adapter	sym53c8xx
SCSI	SCSI storage controller: LSI Logic /Symbios Logic 53c895	sym53c8xx
SCSI	SCSI storage controller: Adaptec AIC-7880U	aic7xxx
SCSI	RAID bus controller: American Megatrends Inc. MegaRAID	megaraid
SCSI	RAID bus controller: Compaq Computer Corp. Smart Array 5i/532	cciss
Ethernet	Ethernet controller: Intel Corp. 82557/8/9 [Ethernet Pro 100]	e100, eepro100
Ethernet	Ethernet controller: BROADCOM Corp. NetXtreme BCM5701 Gigabit Ethernet	bcm5700, tg3

Memory failure

Memory failure can be seen probably by the crash of an application or a crash of the operating system. In the former case, a error message indicating a memory problem is likely to be seen in `/var/log/messages`. If the amount of memory in `/proc/meminfo` is shown to be below the RAM built into the server, it is very likely that a RAM module has failed.

```
# more /proc/meminfo
MemTotal: 4119792 kB
```

The missing memory is in most cases ½ or ¼ of the total memory, as most servers have two or four RAM slots occupied. It cannot be determined from within Linux, which RAM module is affected. The right slot has to be determined by either using offline diagnostics tools, or by removing one module after another and trying to boot.

Harddisk failure

The SCSI bus is only scanned by the kernel at boot-up. Changes in the SCSI layer are not recognized until reboot. Thus, any harddisk failure is not immediately visible in `/proc/scsi/` or `/var/log/messages`. A failure is usually seen by the user while trying to read or write to the disk by accessing a file. Linux will issue an I/O error message then. The failed disk can be determined by

- knowing on which disk the filesystem resides that issued the error
- by rebooting and looking in `/proc/scsi/scsi` which disk is missing
- by using a SCSI rescan utility (i.e. `hp_rescan` from the fibreutils suite) and again looking into `proc/scsi/scsi`

Example

```
Host: scsi0 Channel: 00 Id: 00 Lun: 00
Vendor: HP 73.4G Model: MAX3073NC Rev: HPC1
Type: Direct-Access ANSI SCSI revision: 03
Host: scsi0 Channel: 00 Id: 02 Lun: 00
Vendor: HP 73.4G Model: MAX3073NC Rev: HPC1
Type: Direct-Access ANSI SCSI revision: 03
The ”second” disk is missing with Host: scsi0 Channel: 00 Id: 01
Lun: 00.
```

ProLiant Support Pack

The ProLiant Support Pack for Linux is a software bundle containing

- drivers for NICs, Fibre Channel HBAs, SCSI adapters, and RAID controllers
- HP System Management Homepage
- HP Lights-Out Drivers and Agents
- HP System Health Application and Insight Management Agents
- HP Array Configuration Utility
- Fibre Channel Utilities for HP StorageWorks
- HP Insight Diagnostics Online Edition

Fibre channel

Basics

Devices connected via fibre channel are handled by the standard SCSI layer. So disks are treated as normal SCSI disks, using the usual device files `/dev/sda`, `/dev/sdb` and so on. The same is true for tape drives.

Supported configurations

Make sure you check the support matrix for your connectivity. FC-Adapters from Qlogic and Emulex are supported from their respective vendors. There are no Linux drivers for FC cards of HP-UX systems using an Agilent chipset. As of August 2007 HP officially supports selected Fibre Channel HBAs:

For ProLiants:

- PCI-X-based FCA2214 and FCA2214DC (Qlogic)
- PCI-Express-based FC1142SR and FC1242SR, FC2142SR, FC2142SR

For Integrity

- PCI-X-based FC1143, FC2143

Example:

```
# dmesg
Host: scsi2 Channel: 00 Id:
02 Lun: 00
Vendor: HP Model: A6189A Rev: HP14
Type: Direct-Access ANSI SCSI revision: 03
Host: scsi2 Channel: 00 Id: 02 Lun: 01
Vendor: HP Model: A6189A Rev: HP14
Type: Direct-Access ANSI SCSI revision: 03
Host: scsi2 Channel: 00 Id: 02 Lun: 02
Vendor: HP Model: A6189A Rev: HP14
Type: Direct-Access ANSI SCSI revision: 03
Host: scsi2 Channel: 00 Id: 02 Lun: 03
Vendor: HP Model: A6189A Rev: HP14
Type: Direct-Access ANSI SCSI revision: 03
Host: scsi2 Channel: 00 Id: 02 Lun: 05
Vendor: HP Model: A6189A Rev: HP14
Type: Direct-Access ANSI SCSI revision: 03
```

Details, limitations, known problems

The way Linux assigns SCSI device files has severe limitations if you implement changes in your SAN environment, e.g. adding LUNs, because the device files may change! The LUN that used to be `/dev/sdc` may now become `/dev/sde`. The official Red Hat advice is to reboot the system, to make changes in a SAN effective. The HP fibreutils feature a tool called "`hp_rescan`" to avoid this. It is recommended to install the utilities for working with Fibre Channel devices.

Support for fabric rediscovery through `sysfs` is now available in Red Hat Enterprise Linux 4 Update 3 and later. For the Qlogic (`qla2xxx`) and Emulex (`lpfc`) Fibre Channel HBA drivers, run the following commands to perform rediscovery and rescan for new storage:

```
#echo "1" > /sys/class/fc_host/hostXYZ/issue_lip

#echo "- - -" > /sys/class/scsi_host/hostXYZ/scan
```

XYZ is the SCSI host number of your HBA.

For Red Hat Enterprise Linux 2.1 and 3 rebooting the system is still the only supported method for adding a new SCSI (and Fibre Channel belongs to the SCSI layer) device to a running system (according to Red Hat's knowledgebase article 4011). This said, you can try the following commands at your own risk:

```
#echo "scsi add-single-device <H> <C> <I> <L>" > /proc/scsi/scsi
```

H, C, I, L represent Host, Channel, ID, and LUN. This will add the device specified so that it is accessible to the system.

```
#echo "scsi remove-single-device <H> <C> <I> <L>" >
/proc/scsi/scsi
```

This will remove the device specified so that it is no longer accessible to the system.

```
#echo "scsi scan-new-devices" > /proc/scsi/scsi
```

This will scan all host adapters again to see if there are any new devices. Novell is not as strict as Red Hat about the rescan issue and offers its own `rescan-scsi-bus.sh` with its Linux distribution.

With the introduction of the device-mapper with RHEL4 U2 and SLES9 SP2 this awkward behavior will slowly sink into being a thing of the past. See the LVM chapter's LVM2 section for more details on the device-mapper.

You do not see all LUNs

- First of all make sure your Fibre Channel card detects all LUNs of your devices. For Qlogic based cards you can check that either by the Qlogic BIOS, or within the OS by having a look at `/proc/scsi/qla...` If not all LUNs are detected here you do not have to proceed to the next item. You must check your SAN settings.

- Let's assume the FC driver detects all LUNs, but there maybe problems integrating these LUNs to the Linux SCSI layer. How is this mapping done? Linux scans all adapters, all channels, all targets (SCSI IDs), all LUNs looking for a device which should be passed on to the Linux SCSI (and thus disk or tape) layer.

  ```
  for adapter=0 to nr_adapters
  for channel=0 to nr_channel
  for target=0 to nr_targets
  for lun=0 to max_scsi_luns
  check for device: abort if no answer, otherwise continue
  }
  ```

```
}
}
}
```

- What does that mean? LUN 0 must always be mapped, otherwise there will be no chance of seeing any subsequent LUN on your device. The SCSI layer stops scanning for further LUNs if the first one queried (LUN0) doesn't respond. The driver has to assume there is no such device with that SCSI id.

- MULTI LUN support, up to what LUN will be scanned? Linux kernel has a general switch called "multi LUN support". SUSE has this switched on, RH off. Additionally there is a black/whitelist of systems which are known to need special handling, one of these special handlings is the multi LUN feature. The whitelist is part of the `/usr/src/linux-2.4/drivers/scsi/scsi scan.c` file.

- In order to get multi LUN support, you must either have the general MULTI LUN support flag or make sure your device (the string reported upon the SCSI inquiry command) is marked as device with multi-LUN support in the SCSI black/whitelist. In rare cases where none of these conditions are met, there is still a chance to get access to the LUNs. Install the HP fibreutils (www.hp.com) and run `hp rescan -a`, or the SUSE `scsi_rescan.sh` shell script. This will add the devices via the command

  ```
  # echo add single-device <H> <C> <I> <L> > /proc/scsi/scsi
  ```

  as described in the SCSI-2.4 Howto.

• What about sparse LUNs? Sparse LUNs means the LUNs mapped to your device are not contiguous, e.g. 0, 1, 2, 6 If you have a look at the algorithm described above, you will notice that the scan usually stops at the first LUN queried which does not respond. So in this example it would stop with LUN 3 not answering, LUN 6 won't be detected. BUT there is a special flag called SPARSE LUN. If a device has this flag in the white/blacklist the scan is not stopped, but the scan runs to a limit (kernel parameter `max_scsi_luns`) and thus sparse LUNs are handled correctly. **Note**: make sure you set the kernel parameter `max_scsi_luns` to a reasonable value otherwise the scan will take a very long time.

• What about LUNs beyond 7? As of writing this the current Linux implementations only scans SCSI-2 devices (e.g. HP XP systems) up to LUN 7. You will probably guess the solution: There is a flag BLIST LARGE LUN in the SCSI black/whitelist for these devices to scan LUNs greater than 7. Once again if this flag is not set for your storage device the `hp_rescan` utility (or SUSE `rescan_scsi.sh`) will solve the problem.

SANsurfer

SANsurfer is a management application used to install, configure, and deploy Fibre Channel HBAs (Qlogic). HP recommends using the SANsurfer utility for use with the FCA2214(DC). It even features an Java-based graphical front-end.

LUN Persistence

The quest for LUN persistence affects both SCSI and fibre channel devices as they are using the same kernel mechanism. One of the quirks of Linux usage in the enterprise environment is the non-existence of automatic persistence of device files especially for disks and LUNs. The Linux kernel assigns device files during boot-up depending on the order the devices are found during the scan of the bus. Changing disks or LUNs can change the device file for the

same physical disk on the next reboot, when a disk was added or removed, which is scanned earlier.

To avoid device file confusion several tools have been developped to overcome this. Among these are udev, device-mapper, and the HP's LUN Persistence and Migration Utility for Linux, which is an implementation of udev. Please refer to the HP StorageWorks LUN migration and persistence utilities 1.1 Application notes available at www.hp.com.

LVM and LVM2

LVM

LVM is a Logical Volume Manager implemented by Heinz Mauelshagen for the Linux operating system. As of kernel version 2.4, LVM is incorporated in the main kernel source tree. The Linux LVM is designed to closely resemble the LVM used on HP-UX. The commands and options are identical or at least very similar, although the Linux LVM lacks some of the features found in HP-UX's LVM. For an experienced HP-UX user switching to Linux LVM requires only a very flat learning curve.

Logical volume management provides a higher-level view of the disk storage on a computer system than the traditional view of disks and partitions. This gives the system administrator much more flexibility in allocating storage to applications and users. Storage volumes created under the control of the logical volume manager can be resized and moved around almost at will, although this may require some upgrading of file system tools.

The logical volume manager also allows management of storage volumes in user-defined groups, allowing the system administrator to deal with sensibly named volume groups such as "development" and "sales" rather than physical disk names such as sda and sdb. Note that there is a much wider choice of different file systems you can use with Linux LVM than on HP-UX.

LVM2

Red Hat and SUSE introduced device-mapper and LVM2 with RHEL4 U2 and SLES9 SP2. The Linux Kernel Device Mapper is the LVM team's implementation of a minimalistic kernel-space driver that handles volume management, while keeping knowledge of the underlying device layout in user-space. This makes it useful for not only LVM, but EVMS, software raid, and other drivers that create "virtual" block devices.

The changes for userspace are mostly transparent. The commands still do the same as before, although some commands have been added to LVM2.

The device-mapper creates a `/dev/dm-` or `/dev/device-mapper` directory.The device files are mapped directly to a specific hardware device. Before the introduction of the device-mapper Linux created the device files for hard disks and LUNs in the order it received from the SCSI layer. The SCSI layer itself only responded to input from the bus, without "knowing" which specific device was acting. Thus, it was possible, that a new hard disk or LUN configuration could lead to a re-shuffling of the device files. The device-mapper does it differently by mapping a drive or LUN ID to the device file. So, you can see device files with long names deriving from a Fibre Channel WWN. It is possible to make alias files in order to have more recognizable names.

LVM2 acts (and can act only) on device-mapper generated device files. The device-mapper is accompanied by a new RAID and multipath tool aptly named "multipath". More on this in the RAID chapter.

Definitions

- **Volume Group** (VG): The Volume Group is the highest level abstraction used within the LVM. It gathers together a collection of Logical Volumes and Physical Volumes into one administrative unit.

- **Physical Volume** (PV): A physical volume is typically a hard disk, or a disk partition.

- **Logical Volume** (LV): The equivalent of a disk partition in a non- LVM system. The LV is visible as a standard block device; as such the LV can contain a file system (eg. `/home`).

- **Physical Extent** (PE): Each physical volume is divided chunks of data, known as physical extents, these extents have the same size as the logical extents for the volume group.

- **Logical Extent** (LE): Each logical volume is split into chunks of data, known as logical extents. The extent size is the same for all logical volumes in the volume group.

Example: Lets suppose we have a volume group called VG1, this volume group has a physical extent size of 4MB. Into this volume group we introduce two hard disk partitions, `/dev/hda1` and `/dev/hdb1`. These partitions will become physical volumes PV1 and PV2 (more meaningful names can be given at the administrators discretion). The PVs are divided up into 4MB chunks, since this is the extent size for the volume group. The disks are different sizes and we get 99 extents in PV1 and 248 extents in PV2. We now can create ourselves a logical volume, this can be any size between 1 and 347 (248 + 99) extents. When the logical volume is created a mapping is defined between logical extents and physical extents, eg. logical extent 1 could map onto physical extent 51 of PV1, data written to the first 4 MB of the logical volume in fact be written to the 51st extent of PV1.

Features

The Linux LVM can distribute data as linear or striped chunks. It also has a "snapshot" function, that can create an exact copy of a logical volume. If your kernel was compiled with the `/proc` file system support (most are, i.e. SUSE and Red Hat) then you can verify that LVM is present by looking for a `/proc/lvm` directory. When LVM is active you will see entries in `/proc/lvm` for all your physical volumes, volume groups and logical volumes. In addition there is a "file" called `/proc/lvm/global` which gives a summary of the LVM status and also shows just which version of the LVM kernel you are using. LVM can be used for the root disk as long as `/boot` is on a separate partition (and of course the kernel has LVM built-in).

Commands

Initializing disks:

```
# pvcreate /dev/hdb
```

Initializing disk partitions:

```
# pvcreate /dev/hdb1
```

Creating a volume group:

```
# vgcreate <name of volume group> /dev/hda1 /dev/hdb1
```

Activating a volume group:

```
# vgchange -a y <name of volume group>
```

Removing a volume group:

```
# vgchange -a n <name of volume group>

# vgremove <name of volume group>
```

Extending a volume group:

`# vgextend <name of volume group> /dev/hdc1` (where /dev/hdc1 is the "new" physical volume)

Removing a physical volume from a volume group:

```
# vgreduce <name of volume group> /dev/hda1
```

Creating a logical volume: To create a 1500MB linear LV named `testlv` and its block device special `/dev/testvg/testlv`:

```
# lvcreate -L1500 -ntestlv testvg
```

To create a 100 LE large logical volume with 2 stripes and stripesize 4 KB.

```
# lvcreate -i2 -I4 -l100 -nanothertestlv testvg
```

Removing a logical volume: A logical volume must be closed before it can be removed:

```
# umount /dev/myvg/homevol

# lvremove /dev/myvg/homevol
```

Extending a logical volume:

`# lvextend -L12G /dev/myvg/homevol` (extends to 12GB) or

`# lvextend -L+1G /dev/myvg/homevol` (adds 1GB to the logical volume, this is slightly different from HP-UX)

The file systems can be resized offline (requires unmount) or online (ReiserFS, XFS, ext2 with online patch).

Extending a logical volume and file system: For ext2 file systems:

```
# umount /home

# e2fsadm -L+1G /dev/myvg/homevol

# mount /home
```

For ReiserFS:

```
# umount /home

# resize_reiserfs -s+1G /dev/myvg/homevol

# lvreduce -L+1G /dev/myvg/homevol

# mount -treiserfs /dev/myvg/homevol /home
```

There is no way to shrink XFS file systems. Commands to get info about LVM are the same as on HP-UX: `pvdisplay`, `vgdisplay`, `lvdisplay`, `vgscan`.

LVM mirroring

LVM mirroring (LVM2 only) is implemented on the logical volume level and is done by simply adding the -m option to the lvcreate command:

```
# lvcreate -L1500 -m 1 -ntestlv testvg
```

This function is only available in Red Hat Enterprise Linux 4 Update 4 or later.

Limitations and Warnings

These warnings mostly apply to the old LVM (or LVM1).

There is no built-in mirroring in LVM, but it is available with LVM2 (see above). The raidtools (in particular md) are used to mirror LVM structures. This can lead to unsatisfying behavior in some environments, because the underlying structure of how Linux handles SCSI devices is not as sophisticated as in other flavors of UNIX such as HP-UX or Tru64. A note about usage of LVM with ServiceGuard for Linux: LVM is not cluster-aware. Be very careful doing this, LVM is not currently cluster-aware and it is very easy to lose all your data. If you have a fibre-channel or shared-SCSI environment here more than one machine has physical access to a set of disks then you can use LVM to divide these disks up into logical volumes.

The maximum number of VGs are 99 with a maximum of 256 LVs all together (not for each VG). Maximum LV size is 256GB and a maximum of 256 PVs (disks or partitions) can be used. SUSE's YaST Administration tool can configure LVM, but the implementation is still buggy as of this writing, so be careful.

Red Hat Advanced Server 2.1 does not come with LVM and it is only supported with LVM in a ServiceGuard environment. The ServiceGuard software provides the kernel patch and LVM user-land tools. As of August 2007 ServiceGuard alone is not supported with devicemapper and LVM2, although both are supported, if you use a combination of SG/LX and Red Hat Cluster Suite with Global File System (GFS).

Software RAID md and multipath

This section describes the very basics of the Linux Software-RAID-driver md in the 2.4 and 2.6 kernels. This description does not apply to the old RAID functionality of the 2.2 and 2.0 kernels. To give you an idea of how md works this section will show how to setup a software mirror using md and how to replace a faulty disk. For an in-depth discussion of md please refer to the software RAID-HOWTO available in the Linux Documentation project (www.linuxdoc.org) and the documentation on your system, e.g. `/usr/share/doc/packages/raidtools/`

RAID levels

Linux SW-RAID can work on most block devices. It does not matter if you use IDE or SCSI disks, or even a mixture. A Linux RAID device again is a block device, so on top of the RAID device you have LVM. Available RAID levels:

- Linear mode: 2 or more devices are combined into one large device.
- RAID-0 Striping
- RAID-1 Mirror
- RAID-4 parity on one drive, rarely used
- RAID-5 distributed parity information
- Multi-path For those cases where a physical device is available via alternate paths (e.g. via two Fibre Channel links)

RAID setup

All you need is a kernel 2.4.x or 2.6.x with RAID support and the userspace raidtools. The file `/proc/mdstat` delivers information and you should also see a device called `md` in `/proc/devices`. The RAID setup is done via `/etc/raidtab`.

Example: RAID-1

This example will show how to setup a mirror, and how to replace one of the two disks in case it has a hardware fault. For other RAID levels please refer to the documentation mentioned above

Setup: We run SUSE Enterprise Server 8 and have two disks. We would like to have a 1GB ext3 file system mirrored between these two physical disks.

```
# cat /etc/SUSE-release
SUSE SLES-8 (i386) VERSION = 8.1

# cat /proc/scsi/scsi
Attached devices:
Host: scsi0 Channel: 00 Id: 05 Lun: 00
Vendor: HP Model: SAFTE; U160/M BP Rev: 1023
Type: Processor ANSI SCSI revision: 02
Host: scsi0 Channel: 01 Id: 00 Lun: 00
Vendor: MegaRAID Model: LD0 RAID0 17365R Rev: H
```

```
Type: Direct-Access ANSI SCSI revision: 02
Host: scsi0 Channel: 01 Id: 01 Lun: 00 50
Vendor: MegaRAID Model: LD1 RAID0 17365R Rev: H
Type: Direct-Access ANSI SCSI revision: 02
Let's save the partition table of both disks.
#sfdisk -d /dev/sda > /tmp/partitions.sda
#sfdisk -d /dev/sdb > /tmp/partitions.sdb
```

Now let's setup the partitions which are supposed to be mirrored.

fdisk -l
Disk /dev/sda: 255 heads, 63 sectors, 2213 cylinders
Units = cylinders of 16065 * 512 bytes
Device Boot Start End Blocks Id System
/dev/sda1 * 1 7 56196 83 Linux
/dev/sda2 8 40 265072+ 82 Linux swap
/dev/sda3 41 2213 17454622+ f Win95 Ext'd
9.3. EXAMPLE: RAID-1 69
(LBA) /dev/sda5 41 432 3148708+ 83 Linux
/dev/sda6 433 687 2048256 8e Linux LVM
Disk /dev/sdb: 255 heads, 63 sectors, 2213 cylinders
Units = cylinders of 16065 * 512 bytes
Device Boot Start End Blocks Id System

```
# fdisk /dev/sda
The number of cylinders for this disk is set to 2213.
There is nothing wrong with that, but this is larger than 1024,
and could in certain setups cause problems with:
1) software that runs at boot time (e.g., old versions of LILO)
2) booting and partitioning software from other OSs
(e.g., DOS FDISK, OS/2 FDISK)
Command (m for help): n
Command action
l logical (5 or over)
p primary partition (1-4)
l
First cylinder (688-2213, default 688):
Using default value 688
Last cylinder or +size or +sizeM or +sizeK (688-2213, default
2213): +1024M
Command (m for help): t
Partition number (1-7): 7
Hex code (type L to list codes): fd
Changed system type of partition 7 to fd (Linux raid autodetect)
Command (m for help): p
Disk /dev/sda: 255 heads, 63 sectors, 2213 cylinders
Units = cylinders of 16065 * 512 bytes
Device Boot Start End Blocks Id System
/dev/sda1 * 1 7 56196 83 Linux
/dev/sda2 8 40 265072+ 82 Linux swap
/dev/sda3 41 2213 17454622+ f Win95 Ext'd
(LBA) /dev/sda5 41 432 3148708+ 83 Linux
/dev/sda6 433 687 2048256 8e Linux LVM
```

```
/dev/sda7 688 818 1052226 fd Linux raid autodetect
Command (m for help): w
The partition table has been altered!
Calling ioctl() to re-read partition table.
WARNING: Re-reading the partition table failed with error 16:
Device or resource busy.
The kernel still uses the old table.
The new table will be used at the next reboot. Syncing disks.
```

We create a partition of same size and type of `sdb` (not shown in details here.).

```
# fdisk -l
Disk /dev/sda: 255 heads, 63 sectors, 2213 cylinders
Units = cylinders of 16065 * 512 bytes
Device Boot Start End Blocks Id System
/dev/sda1 * 1 7 56196 83 Linux
/dev/sda2 8 40 265072+ 82 Linux swap
/dev/sda3 41 2213 17454622+ f Win95 Ext'd (LBA)
/dev/sda5 41 432 3148708+ 83 Linux
/dev/sda6 433 687 2048256 8e Linux LVM
/dev/sda7 688 818 1052226 fd Linux raid autodetect
Disk /dev/sdb: 255 heads, 63 sectors, 2213 cylinders
Units = cylinders of 16065 * 512 bytes
Device Boot Start End Blocks Id System
/dev/sdb1 1 131 1052226 fd Linux raid autodetect
```

As the partition table could not be re-read we need to perform a reboot!

```
# reboot
```

Create /etc/raidtab.

```
# cat /etc/raidtab
raiddev /dev/md0
raid-level 1
nr-raid-disks 2
nr-spare-disks 0
chunk-size 4
persistent-superblock 1
device /dev/sda7
raid-disk 0
device /dev/sdb1
raid-disk 1
```

Now let's initialize the RAID.

```
# mkraid /dev/md0
handling MD device /dev/md0
analyzing super-block
disk 0: /dev/sda7, 1052226kB, raid superblock at 1052160kB
disk 1: /dev/sdb1, 1052226kB, raid superblock at 1052160kB
# cat /proc/mdstat
```

```
Personalities : [raid1]
read_ahead 1024 sectors
md0 : active raid1 sdb1[1] sda7[0]
1052160 blocks [2/2] [UU]
[===>.................] resync = 16.6\% (175260/1052160)
finish=1.9min \
speed=7620K/sec
unused devices: <none>
```

A little bit later, the synchronization between the 2 partitions is done, everything is ok.

```
# cat /proc/mdstat
Personalities : [raid1]
read_ahead 1024 sectors
md0 : active raid1 sdb1[1] sda7[0]
1052160 blocks [2/2] [UU]
unused devices: <none>
```

We're now ready to create a filesystem on /dev/md0.

```
# mkfs.ext3 /dev/md0
mke2fs 1.28 (31-Aug-2002)
Filesystem label=
OS type: Linux
Block size=4096 (log=2)
Fragment size=4096 (log=2)
131616 inodes, 263040 blocks
13152 blocks (5.00\%) reserved for the super user
First data block=0
9 block groups
32768 blocks per group, 32768 fragments per group
14624 inodes per group S
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 33 mounts or
180 days, whichever comes first. Use tune2fs -c or -i to
override.
```

Let's assume disk /dev/sdb needs to be replaced.

```
# raidhotremove /dev/md0 /dev/sdb1
/dev/md0: can not hot-remove disk: disk busy!

# umount /dev/md0

# raidstop /dev/md0
```

Usually raidhotremove should be able to remove the drive from the array. Obviously this wasn't possible here for an unknown reason. So we needed to stop the RAID in order to hot-

swap the faulty disk. A reboot might be required in some cases! The next step is to re-create partitions on the new empty disk so that the partition layout is the same as on the replaced disk.

A backup of the old partition table becomes quite useful here. Afterwards the RAID can be started again to synchronize data.

```
# fdisk -l /dev/sdb
Disk /dev/sdb: 255 heads, 63 sectors, 2213 cylinders
Units = cylinders of 16065 * 512 bytes
Device Boot Start End Blocks Id System
# sfdisk /dev/sdb </tmp/partitions.sdb
Checking that no-one isusing this disk right now ...
OK
Disk /dev/sdb: 2213 cylinders, 255 heads, 63 sectors/track
Old situation:
Units = cylinders of 8225280 bytes, blocks of 1024 bytes,
counting
from 0
Device Boot Start End #cyls #blocks Id System
/dev/sdb1 0 - 0 0 0 Empty
/dev/sdb2 0 - 0 0 0 Empty
/dev/sdb3 0 - 0 0 0 Empty
/dev/sdb4 0 - 0 0 0 Empty
New situation:
Units = sectors of 512 bytes, counting from 0
Device Boot Start End #sectors Id System
/dev/sdb1 63 2104514 2104452 fd Linux raid autodetect
/dev/sdb2 0 - 0 0 Empty
/dev/sdb3 0 - 0 0 Empty
/dev/sdb4 0 - 0 0 Empty
Warning: no primary partition is marked bootable (active)
This does not matter for LILO, but the DOS MBR will not boot this
disk.
Successfully wrote the new partition table
Re-reading the partition table ...
```

If you created or changed a DOS partition, `/dev/foo7`, say, then use `dd` to zero the first 512 bytes: `dd if=/dev/zero of=/dev/foo7 bs=512 count=1` (See fdisk(8).)

```
# fdisk -l /dev/sdb
Disk /dev/sdb: 255 heads, 63 sectors, 2213 cylinders
Units = cylinders of 16065 * 512 bytes
Device Boot Start End Blocks Id System
/dev/sdb1 1 131 1052226 fd Linux raid autodetect

# raidstart /dev/md0

# cat /proc/mdstat
Personalities : [raid1]
read_ahead 1024 sectors
md0 : active raid1 sda7[0]
```

```
1052160 block [2/1] [U_]
unused devices: <none>

# raidhotadd /dev/md0 /dev/sdb1

# cat /proc/mdstat
Personalities : [raid1]
read_ahead 1024 sectors
md0 : active raid1 sdb1[2] sda7[0]
1052160 blocks [2/1] [U_]
[>.....................] recovery = 0.9\% (11008/1052160)
finish=1.5min \
speed=11008K/sec
unused devices: <none>
/dev/md0: 11/131616 files (0.0\% non-contiguous), 12348/263040
blocks

# cat /proc/mdstat
Personalities : [raid1]
read_ahead 1024 sectors
md0 : active raid1 sdb1[1] sda7[0]
1052160 blocks [2/2] [UU]
unused devices: <none>
```

Booting on RAID?

The simple answer is: It should be avoided, if possible. There are several ways to setup a system to boot from a software RAID, but they are all complicated, and add another level of software. Considering the cost of a low-end hardware RAID controller (nearly every HP ProLiant model ships with a built-in Smart Array Controller), the need for a software RAID is no longer given.

Multipathing with md

You can implement a rudimentary version of multipathing with md using `raidtab`:

```
raiddev /dev/md0
raid-level multipath
nr-raid-disks 2
device /dev/sdb
raid-disk 0
device /dev/sdf
raid-disk 1
raiddev /dev/md1
raid-level multipath
nr-raid-disks 2
device /dev/sdg
raid-disk 0
device /dev/sdc
raid-disk 1
```

Another way is to use `mdadm`:

```
# mdadm create /dev/md0 level multipath raid-devices=2 /dev/sdc
/dev/sdd
```

This multipathing device will failover in an event of failure. It uses one path only, load balancing is not possible. The original path will not be monitored and `md` will not failback to the original path, if it comes back online. You have to monitor the path and switch back to it manually. Should the second path fail, the system will prompt an I/O error, and access to the device will be lost. You have to decide whether to use the device in multipath mode or as an ordinary RAID, a combination does not work.

Multipathing with multipath

Device-mapper, LVM2 and multipath are all parts of an general update to the volume management handling of Linux. While the device-mapper is a significant improvement to the old way of device file generation, multipath is the equivalent for doing multipathing on Linux.

Multipathing with multipath tools consists of defining the multipath devices in the configuration file `/etc/multipath.conf` and starting the daemon `mulitpathd` with

```
# /etc/init.d/multipathd start
```

The `multipath.conf` template file looks like this:

```
##
## This is a template multipath-tools configuration file
## Uncomment the lines relevent to your environment
##
#defaults {
# udev_dir /dev
# polling_interval 10
# selector "round-robin 0"
# path_grouping_policy multibus
# getuid_callout "/sbin/scsi_id -g -u -s /block/\%n"
# prio_callout /bin/true
# path_checker readsector0
# rr_min_io 100
# rr_weight priorities
# failback immediate
# no_path_retry fail
# user_friendly_names no
#}
#devnode_blacklist {
# wwid 26353900f02796769
# devnode "^(ram|raw|loop|fd|md|dm-|sr|scd|st)[0-9]*"
# devnode "^hd[a-z][[0-9]*]"
# devnode "^cciss!c[0-9]d[0-9]*[p[0-9]*]"
# device {
# vendor DEC.*
# product MSA[15]00
# }
#}
#multipaths {
```

```
# multipath {
# wwid 3600508b4000156d700012000000b0000
# alias yellow
# path_grouping_policy multibus
# path_checker readsector0
# path_selector "round-robin 0"
# failback manual
# rr_weight priorities
# no_path_retry 5
# rr_min_io 100
# }
# multipath {
# wwid 1DEC_____321816758474
# alias red
# }
#}
#devices {
# device {
# vendor "COMPAQ "
# product "HSV110 (C)COMPAQ"
# path_grouping_policy multibus
# getuid_callout "/sbin/scsi_id -g -u -s /block/\%n"
# path_checker readsector0
# path_selector "round-robin 0"
# hardware_handler "0"
# failback 15
# rr_weight priorities
# no_path_retry queue
# rr_min_io 100
# product_blacklist LUN_Z
# }
# device {
# vendor "COMPAQ "
# product "MSA1000 "
# path_grouping_policy multibus
# }
#}
```

The ”selector” set to ”round-robin” indicates load-balancing, ”multibus” stands for failover in the ”path grouping policy”. In the “devices” section you have to group all devices according to the failover policy.

If `pvscan` reports duplicate physical volumes, LVM2 must be instructed not to look at the paths but only at the multipaths. Add this line to the devices block in `/etc/lvm/lvm.conf`:

```
filter = [ "r/sd.*/", "a/.*/" ]
```

You can mix HBA models from different vendors with different speed and you can mix storage controllers. Root file system on a multi-path’d SAN is supported.

File Systems

When Linux was first developed, it supported only one filesystem - the Minix filesystem. Today, Linux has the ability to support several file systems concurrently. This was done by the introduction of another layer between the kernel and the filesystem code - The Virtual File System (VFS).

The kernel "speaks" to the VFS layer. The VFS layer passes the kernel's request to the proper filesystem management code. After the integration of the VFS in the kernel, a new filesystem, called the "Extended File System" was implemented in April 1992 and added to Linux 0.96c. This new filesystem removed the two big Minix limitations: its maximum size was 2GB and the maximum file name size was 255 characters. It was an improvement over the Minix filesystem but some problems were still present in it.

As a response to these problems, the Second Extended File System (ext2) was released in January 1993. It is a stable and well-tested file system now, but lacks journaling capabilities. Journaled file system recover much faster - no long `fsck` required - and are better protected against meta-data corruption. In 2000 journaling was added to ext2 and the Third Extended File System (ext3) was released. ext2 and ext3 are something like Linux' "native" file systems. They are robust and widely used. The next iteration, aptly named ext4, is in beta testing.

There are a lot of other file systems available for Linux. To list a few: IBM's Journaling File System (JFS), The Naming System Venture's Reiser File System (ReiserFS), and SGI's XFS.

Linux can also read and write to a lot of file systems used on other operating systems: FAT and NTFS (Microsoft DOS and Windows), HFS+ (MacOS X), and legacy file systems like HFFS (Amiga), BeFS (BeOS), and HPFS (OS/2).

Unfortunately Linux does not support the file systems used on HP-UX (vxfs, hfs) and vice versa. You cannot mount a HP-UX partition on a Linux system, say, on a double-boot Itanium® system. We concentrate on the file systems which most distributions use. We will discuss the virtual file system /proc, and the "real" file systems ext2, ext3 and ReiserFS.

The `/proc` file system

The `/proc` file system is a "virtual" file system. That means none of the "files" that it contains are stored on the hard disk, and only exists in memory. Rather, it is a way to easily access dynamic information about the system at any time. Working with a Linux system sometimes requires a more in-depth knowledge of your system than other operating systems, such as, "what PCI devices do I currently have in my system? What interrupts are currently in use?" The `/proc` file system allows both reading and writing data from and to the kernel to influence system behavior. Given the fact that the `/proc` file system is virtual, it does not cover disk space and of course cannot be formatted. It can also be used to pass kernel data to programs (e.g shell scripts). Example:

```
# cat /proc/interrupts
CPU0
0: 1319556 XT-PIC timer
1: 64602 XT-PIC keyboard
2: 0 XT-PIC cascade
8: 132 XT-PIC rtc
9: 0 XT-PIC acpi
```

```
10: 45783 XT-PIC ncr53c8xx
11: 274161 XT-PIC nvidia
12: 209994 XT-PIC PS/2 Mouse
14: 0 XT-PIC EMU10K1
15: 19490 XT-PIC eth0
NMI: 0
LOC: 1319560
ERR: 330
MIS: 0

# echo "1" > /proc/sys/kernel/sysrq
```

This enables the “sysrq” function which allows us to complete several operations even if the kernel seems to locked up completely. The documentation about the handling of /proc in the kernel can be found in `/usr/src/linux/Documentation`.

The ext2 file system

This is the second version of the original Linux file systems. Because it does not provide any journaling, a file system check of a large ext2 partitions can take a very long time. The ext2 file system is considered as “rock solid”.

Creating and mounting ext2

```
# mkfs.ext2 /dev/lvtest/var
mke2fs 1.26 (3-Feb-2002)
Filesystem label=
OS type: Linux
Block size=4096 (log=2)
Fragment size=4096 (log=2)
221760 inodes, 443392 blocks
22169 blocks (5.00\%) reserved for the super user
First data block=0
14 block groups
32768 blocks per group, 32768 fragments per group
15840 inodes per group
Superblock backups stored on blocks:
32768, 98304, 163840, 229376, 294912
Writing inode tables: done
Writing superblocks and filesystem accounting information: done
This filesystem will be automatically checked every 30 mounts or
180 days, whichever comes first. Use tune2fs -c or -i to
override.

# mount /dev/lvtest/var /mnt

# mount
/dev/lvtest/var on /mnt type ext2 (rw)
10.2.2 Checking ext2
# fsck -f /dev/lvtest/var
fsck 1.26 (3-Feb-2002)
e2fsck 1.26 (3-Feb-2002)
Pass 1: Checking inodes, blocks, and sizes
```

```
Pass 2: Checking directory structure
Pass 3: Checking directory connectivity
Pass 4: Checking reference counts
Pass 5: Checking group summary information
/dev/lvtest/var:   11/221760   files   (0.0\%   non-contiguous),
6975/443392 blocks

# mount /dev/lvtest/var /mnt

# mount
/dev/lvtest/var on /mnt type ext2 (rw)
```

The ext3 file system

To overcome the disadvantages of a long file checking routine the ext3 file system was developed. Journaling allows checking only the modified files leading to faster file checking. The ext3 file system is in fact ext2 with a journaling facility. With ext3 being backward compatible to ext2 and ext3 file system can be mounted as ext2 (without the journaling)

Creating and mounting ext3

There are two ways to create an ext3 file system: Converting ext2 to ext3 or the direct creation.

```
# tune2fs -j /dev/lvtest/var
tune2fs 1.26 (3-Feb-2002)
Creating journal inode: done
This filesystem will be automatically checked every 22 mounts or
180 days, whichever comes first. Use tune2fs -c or -i to
override.

# mount /dev/lvtest/var /mnt

# mount
/dev/lvtest/var on /mnt type ext3 (rw)
```

or

```
# mkfs.ext3 /dev/lvtest/var
```

Checking ext3

```
# fsck -f /dev/lvtest/var
fsck 1.26 (3-Feb-2002)
e2fsck 1.26 (3-Feb-2002)
Pass 1: Checking inodes, blocks, and sizes
Pass 2: Checking directory structure
Pass 3: Checking directory connectivity
Pass 4: Checking reference counts
Pass 5: Checking group summary information
/dev/lvtest/var: 11/221760 files (0.0\% non-contiguous),
15177/443392 blocks
```

ReiserFS

ReiserFS is another journaling file system providing fast journaling based on balanced trees. It is not compatible with ext2 or ext3. ReiserFS is developed by Hans Reiser and is the default root file system on SUSE.

Creating and mounting ReiserFS

```
# mkreiserfs /dev/lvtest/var
mkreiserfs 3.x.1b (2002)
mkreiserfs: Guessing about desired format..
mkreiserfs: Kernel 2.4.19-A7 is running.
Format 3.6 with standard journal
Count of blocks on the device: 443392
Number of blocks consumed by mkreiserfs formatting process: 8225
Blocksize: 4096
Hash function used to sort names: "r5"
Journal Size 8193 blocks (first block 18)
Journal Max transaction length 1024
inode generation number: 0
UUID: 5cdaaef1-3e41-406d-a366-23ec7d385732
ATTENTION: YOU SHOULD REBOOT AFTER FDISK!
ALL DATA WILL BE LOST ON '/dev/lvtest/var'!
Continue (y/n):y
Initializing journal -
0\%....20\%....40\%....60\%....80\%....100\%
Syncing..ok
```

Please visit www.namesys.com for information about ReiserFS sponsors

```
# mount /dev/lvtest/var /mnt

# mount
/dev/lvtest/var on /mnt type reiserfs (rw)
```

Checking ReiserFS

```
# /sbin/fsck.reiserfs /dev/lvtest/var
reiserfsck 3.x.1b (2002)
Will read-only check consistency of the filesystem on
/dev/lvtest/var Will put log info to 'stdout'
Do you want to run this program? [No/Yes] (note need to type
Yes):Yes
###########
reiserfsck --check started at Sun Mar 30 21:54:51 2003
###########
Replaying journal..
0 transactions replayed
Checking S+tree..ok
Comparing bitmaps..ok
Checking Semantic tree... ok
No corruptions found
82 CHAPTER 10. FILE SYSTEMS
```

```
There are on the filesystem:
Leaves 1
Internal nodes 0
Directories 1
Other files 0
Data block pointers 0 (zero of them 0)
Safe links 0
###########
reiserfsck finished at Sun Mar 30 21:54:55 2003
###########
```

Networking

Linux distributions may have some differences where certain files are placed. If you are unsure a quick lookup with the locate command can be helpful.

Network card drivers

Network card drivers are provided either as a module or they are built statically into the kernel. For typical drivers see Chapter 5. It depends on the network card driver, if settings like speed and mode (Half-duplex, Full-Duplex) are logged and if these parameters can be set via options. For details please refer to Donald Becker's Network Homepage (www.scyld.com).

Network startup scripts

Script	Config files	Distro
`/etc/init.d/network`	`/etc/sysconfig/network/`	SUSE
`/etc/init.d/network`	`/etc/sysconfig/network-scripts`	Red Hat
`/etc/init.d/named`	`/etc/named.conf`	both
`/etc/init.d/nfsserver`	`/etc/sysconfig/nfs`	SUSE
`/etc/init.d/nfs`	`/etc/sysconfig/network`	Red Hat
`/etc/init.d/inetd`	`/etc/inetd.conf`	SUSE
`/etc/init.d/xinetd`	`/etc/xinetd.conf, /etc/xinet.d/`	Red Hat
`/etc/init.d/ypserv`	`/etc/ypserv.conf, /var/yp/securenets`	both
`/etc/init.d/nscd`	`/etc/nscd.conf`	both
`/etc/init.d/xntpd`	`/etc/ntpd.conf`	SUSE
`/etc/init.d/ntpd`	`/etc/ntpd.conf`	Red Hat

Explanations:

- `/etc/init.d/network` — Basic network startup
- `/etc/init.d/named` — Name Server startup (BIND)
- `/etc/init.d/nfsserver` — NFS Server startup
- `/etc/init.d/nfs` — NFS Server startup
- `/etc/init.d/inetd` — Network Services
- `/etc/init.d/xinetd` — Network Services

- `/etc/init.d/ypserv` — NIS-Server startup
- `/etc/init.d/nscd` — Name service cache daemon
- `/etc/init.d/xntpd` — time distribution daemon
- `/etc/init.d/ntpd` — time distribution daemon

Man pages are available also, so a `man <server>` is always helpful.

Location of files

Configuration files

- `/etc/hosts` — IP to name resolution
- `/etc/networks` — netname-to-address mapping
- `/etc/resolv.conf` — domain name and DNS server IP
- `/etc/services` — maps services to UDP/TCP ports
- `/etc/inetd.conf` — list of services controlled by inetd
- `/etc/xinetd.conf` — list of services controlled by xinetd

Security related files

- `/etc/exports` — NFS export
- `/etc/netgroups` — for NFS
- `/etc/hosts.equiv` — for `remsh`, `rcp`, `rlogin`
- `/etc/hosts.allow`—access control for inetd services started through tcpd
- `/etc/hosts.deny` — access control for inetd services started through tcpd
- `$HOME/.rhosts` — for `remsh`, `rcp`, `rlogin`
- `/etc/ftpusers` — for `ftp`, users who may not login via `ftp`
- `/etc/securetty` — specifies tty from which root can login

Root login: By default `/etc/securetty` will prevent root-login via network!

Log files

- `/var/log/messages` — central Linux log file
- `/var/log/log` — log files of all kind

- `/var/log/boot.msg` — boot log file
- `/var/XFree86.log` — X-Server log file(s)

Basic Network and Services Diagnostics

Linux as a UNIX-like operating system provides similar means to check and troubleshoot network as HP-UX does.

Example: Verify ratio of RX-OK packets versus RX-ERR, same for outbound (TX-)

```
# netstat -i
Kernel Interface table
Iface MTU Met RX-OK RX-ERR RX-DRP RX-OVR TX-OK TX-ERR TX-DRP TX-
OVR Flg
eth0 1500 0 581046 0 0 0 469446 0 0 0 BMRU
lo 16436 0 66340 0 0 0 66340 0 0 0 LRU
ppp0 1492 0 7544 0 0 0 5445 0 0 0 MOPRU
Check kernel routing table

# netstat -rn
Kernel IP routing table
Destination Gateway Genmask Flags MSS Window irtt Iface
195.158.139.241 0.0.0.0 255.255.255.255 UH 40 0 0 ppp0
0.0.0.0 195.158.139.241 0.0.0.0 UG 40 0 0 ppp0
```

This reflects the actual kernel routing table. If routes should be present after system startup, these should be entered into the appropriate config file `/etc/sysconfig/network` resp. `/etc/sysconfig/network/`scripts.

Ping another host's IP:

```
# ping 195.158.139.241
64 bytes from 195.158.139.241: icmp_seq=2 ttl=127 time=16.7 ms
--- 195.158.139.241 ping statistics ---
2 packets transmitted, 2 received, 0% loss, time 1012ms
rtt min/avg/max/mdev = 16.713/19.607/22.501/2.894 ms
```

Ping another host's name, but ping will resolve the other host's name. If any problems like this occur use nslookup or dig.

```
# ping www.hp.com
```

PING www.hpgtm.speedera.net (192.6.118.97) from 213.128.125.212

Record route (-o in HP UX):

```
# ping -R 195.158.139.241
64 bytes from 195.158.139.241: icmp_seq=1 ttl=127 time=21.0 ms
NOP
RR: 213.128.125.212
213.128.125.212
```

Printout actual status of all interfaces:

```
# ifconfig
eth0 Link encap:Ethernet HWaddr 00:80:48:DD:2B:B8
UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
RX packets:581391 errors:0 dropped:0 overruns:0 frame:0
TX packets:469865 errors:0 dropped:0 overruns:0 carrier:0
collisions:95 txqueuelen:100
RX bytes:472061566 (450.1 Mb) TX bytes:40069024 (38.2 Mb)
Interrupt:15 Base address:0x2000
lo Link encap:Local Loopback
inet addr:127.0.0.1 Mask:255.0.0.0
UP LOOPBACK RUNNING MTU:16436 Metric:1
RX packets:66340 errors:0 dropped:0 overruns:0 frame:0
TX packets:66340 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:0
RX bytes:111386722 (106.2 Mb) TX bytes:111386722 (106.2 Mb)
ppp0 Link encap:Point-to-Point Protocol
inet addr:213.128.125.212 P-t-P:195.158.139.241
Mask:255.255.255.255
UP POINTOPOINT RUNNING NOARP MULTICAST MTU:1492 Metric:1
RX packets:7760 errors:0 dropped:0 overruns:0 frame:0
TX packets:5735 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:3
RX bytes:8151207 (7.7 Mb) TX bytes:316545 (309.1 Kb)
```

Check arp cache content:

```
# arp -a
hprtdu14.grc.hp.com (15.140.8.172) at 08:00:09:18:4C:6B [ether]
on eth0
grcdg437.grc.hp.com (15.140.11.136) at 08:00:20:F0:F2:47 [ether]
on eth0
rtgdg501.grc.hp.com (15.140.11.292) at 00:30:6E:2C:04:37 [ether]
on eth0
```

Check name resolution order:

```
# cat /etc/nsswitch.conf
passwd: compat
group: compat
hosts: files dns
networks: files dns
localhost should always work:

# grep localhost /etc/hosts
127.0.0.1 localhost
```

For DNS-client operation check `/etc/resolv.conf`:

```
# cat /etc/resolv.conf
nameserver 141.1.1.12
```

```
nameserver 195.158.131.2
```

To verify correct name resolution:

```
# dig www.hp.com
[...]
;; ANSWER SECTION:
www.hp.com. 4367 IN CNAME www.hpgtm.speedera.net.
www.hpgtm.speedera.net. 111 IN A 192.6.118.44
www.hpgtm.speedera.net. 111 IN A 192.6.118.97
[...]
```

To check open files and port usage:

```
# lsof -i :111
COMMAND PID USER FD TYPE DEVICE SIZE NODE NAME
portmap 17558 bin 3u IPv4 2820905 UDP *:sunrpc
portmap 17558 bin 4u IPv4 2820906 TCP *:sunrpc (LISTEN)
```

As an example the portmapper presence is verified. `lsof` is a very useful program to check for open files and ports.

When certain services (e.g. telnet) cannot be used (message "connection refused") check whether inetd/xinetd processes are running.

```
# ps -ef | grep inetd
root 1664 1 0 Mar23 ? 00:00:00 /usr/sbin/inetd
```

SUSE or Red Hat may not startup inetd/xinetd automatically as a security feature, start inetd/xinetd manually if necessary. Also check whether the requested services are activated in the config files, they may be commented out for security reasons.

NFS server

Linux provides kernel NFS supporting both v2 and v3. This has to be activated in the kernel.

NFS Configuration files

There are three main configuration files you have to edit to set up an NFS server: `/etc/exports`, `/etc/hosts.allow`, and `/etc/hosts.deny`. Strictly speaking, you only need to edit `/etc/exports` to get NFS to work.

This file contains a list of entries; each entry indicates a volume that is shared and how it is shared. Check the man pages (`man exports`) for a complete description of all the setup options for the file, although the description here will probably satisfy most people's needs. An entry in `/etc/exports` will typically look like this:

```
/usr/local 192.168.0.1(ro) 192.168.0.2(ro)
/home 192.168.0.1(rw) 192.168.0.2(rw)
```

or

```
/usr/local 192.168.0.0/255.255.255.0(ro)
```

```
/home 192.168.0.0/255.255.255.0(rw)
```

NFS Processes

NFS depends on the portmapper daemon, either called portmap or rpc.portmap. It will need to be started first.

```
#ps -ef | grep portmap
```

NFS serving is taken care of by five daemons: rpc.nfsd, which does most of the work; rpc.lockd and rpc.statd, which handle file locking; rpc.mountd, which handles the initial mount requests, and rpc.rquotad, which handles user file quotas on exported volumes. Verifying that NFS is running: To do this, query the portmapper with the command `rpcinfo -p` to find out what services it is providing. You should get something like this:

```
program vers proto port
100000 2 tcp 111 portmapper
100000 2 udp 111 portmapper
11.5. NFS SERVER 89
100011 1 udp 749 rquotad
100011 2 udp 749 rquotad
100005 1 udp 759 mountd
100005 1 tcp 761 mountd
100005 2 udp 764 mountd
100005 2 tcp 766 mountd
100005 3 udp 769 mountd
100005 3 tcp 771 mountd
100003 2 udp 2049 nfs
100003 3 udp 2049 nfs
300019 1 tcp 830 amd
300019 1 udp 831 amd
100024 1 udp 944 status
100024 1 tcp 946 status
100021 1 udp 1042 nlockmgr
100021 3 udp 1042 nlockmgr
100021 4 udp 1042 nlockmgr
100021 1 tcp 1629 nlockmgr
100021 3 tcp 1629 nlockmgr
100021 4 tcp 1629 nlockmgr
```

So we have NFS versions 2 and 3, rpc.statd version 1, network lock manager (the service name for rpc.lockd) versions 1, 3, and 4. Some of the services can be used via UDP or TCP. Linux systems use UDP by default unless TCP is explicitly requested

Dump Devices

Red Hat offers two dump device configurations, Netdump and Diskdump.

Netdump

Netdump requires the setup of a netdump server and additional clients. Server Configuration:

1. Verify that the netdump server is installed: `rpm -q netdump-server`. If it is not installed, install it by running the command: `up2date netdump-server`

2. After the netdump server package is installed change the password for the "netdump" user: `passwd netdump`

3. Enable the netdump server: `chkconfig netdump-server on`

4. Start the netdump server: `service netdump-server start`

Client Configuration:

1. Verify that the netdump client is installed: `rpm -q netdump`. Otherwise, install it by running the command: `up2date netdump`.

2. Edit `/etc/sysconfig/netdump` and add the following line: `NETDUMPADDR=<IP address of the netdump server>`

3. Enter the following command and type the netdump password when prompted: `service netdump propagate`

4. Enable the netdump client: `chkconfig netdump on`

5. Start the netdump client: `service netdump start`

When the setup is done all sysrq commands entered should be sent across the network to the netdump server, where they should be stored in `/var/crash`. Dump files generated by netdump are named `vmcore` by default. You should test the setup by entering any sysrq command. Not all hardware is supported.

Diskdump

The diskdump utility offers the ability to collect kernel dumps without needing to be connected to a network. Diskdump creates files in an identical format to the netdump core files and can be analyzed with the same tools.

Like netdump, diskdump will only work with certain hardware. Diskdump on i386 (including AMD64 and EMT64) and Itanium® currently supports aic7xxx, aic79xx, mpt fusion, megaraid (i386 only), and SATA devices. For HP hardware support for diskdump is therefore limited. To configure diskdump, the first step is to load the diskdump module.

```
# modprobe diskdump
```

Check that the module is loaded, the output of the command should look similar to the output below.

```
# dmesg | tail
hader blocks: 3
bitmap blocks: 8
total number of memory blocks: 261999
total blocks written: 262010
```

The diskdump daemons reserved partition can be specified in the file `/etc/sysconfig/diskdump`.

DEVICE=/dev/sde1

Next step is to initialize the partition for use. **Warning**: Any data will be lost.

```
# service diskdump initialformat
```

And finally, add the service to run on startup and then start the service.

```
# chkconfig diskdump on

# service diskdump start
```

The diskdump service is now configured. At the next system crash, the data should be saved to the specified partition. This may take some time if the system has large amounts of RAM.

Linux Kernel Crash Dump `lkcd`

On SUSE Linux systems SLES8, 9, an 10 you can install the LKCD utility to configure a dump device. LKCD generates a crash dump for every kernel panic or oops. The system memory is then saved to the dump device and upon reboot the dump will be created in the dump directory for analysis with the tool `lcrash`. LKCD has the option to dump on the system itself or across the network.

Install the software via YaST (SLES 8 requires SP3 at least). Select the "lkcdutils" package (and "lkcdutils-netdump-server"). After installation configure the dump device in `/etc/sysconfig/dump`. Set DUMPACTIVE to 1, and specify the dump device with DUMPDEV. SLES8 default is `/dev/vmdump`, which is linked to the first swap partition. On SLES9 it is eth0. The dump device should be big enough to hold the content of the memory. Next, you need to specify a dump directory with DUMPDIR. Default is `/var/log/dump`.

DUMP LEVEL determines the verbosity of the dump. Valid options include:

- 0 - do nothing
- 1 - dump the dump header and first 128 bytes
- 2 - dump the dump header and only the kernel pages (only SLES9)
- 4 - everything except free kernel pages (only SLES9)
- 8 - everything

The default is SLES8 is 8. The default in SLES9 is 2. Other options include DUMP COMPRESS (1=RLE, 2=GZIP) and PANIC TIMEOUT (time in seconds for reboot after panic). Run the configuration script and verify it:

```
# lkcd config

# lkcd_config -q
Configured dump device: 0x801
Configured dump flags: KL_DUMP_FLAGS_DISKDUMP
Configured dump level: KL_DUMP_LEVEL_HEADER|KL_DUMP_LEVEL_KERN
Configured dump compression method: KL_DUMP_COMPRESS_GZIP
```

The "_" is used in the second command only. LKCD's default behavior after a panic is to dump to the dump device and then reboot the system. Saving the dump to the dump directory and reconfiguring have to be done manually when the system comes back up by doing the following: Open a terminal and `su` to root; save the dump to the dump directory by entering `lkcd save`, and configure LKCD by entering `lkcd config`.

LKCD comes with a boot script that will do the save and config when the system boots. To use the boot script, first verify the `boot.lkcd` file exists, for example:

```
# ls /etc/init.d/boot.lkcd
/etc/init.d/boot.lkcd
```

A message indicating no file or directory means the file is not installed. The `boot.lkcd` script will not run at boot unless the service is inserted, so enter the following at the prompt:

```
# insserv /etc/init.d/boot.lkcd
```

With `boot.lkcd` configured, `lkcd` will run and check for a dump on the dump device. If one is found a save will occur and then a configuration.

You can test LKCD functionality by pressing Alt-SysRq-d on SLES9 to create a panic. On SLES8, the key combination is Alt-SysRq-c. If SysRq is turned off, turn it on with

```
# echo 1 > /proc/sys/kernel/sysrq.
```

After the reboot check in `/var/log/dump` for a saved dump.

System Administration Tools

This chapter gives an overview about the most common tools for Linux system administration. It neither will go into detail for every product mentioned nor will it name all the tools available. There are many system administration tools for Linux both for graphical and command line environments.

Every distribution has its own approach to system administration. For example SUSE favors its own (non-GPL'd) YaST2, a graphical frontend to nearly all common system adminstration tasks — similar to HP-UX' SAM, not in its usage, but in its aim. Red Hat has a loosely grouped bunch of tools which can be started from the same folder (aptly named "Start here").

The graphical tools all more or less try to resemble Microsoft Windows' Control Panel. There are of course equivalents to all the graphical tools on the command line and more. The command line is still more powerful, but that's the UNIX way and how it should be.

SUSE YaST and YaST2

SUSE has developed its own administration tool for a long time. YaST (Yet another Setup Tool) can be accessed as YaST for non-X-environments and YaST2 for X-environments (QT libraries are also required, the text mode of YaST2 is not very user friendly).

The goal of YaST/YaST2 is to get everything running and to make changes easily with a few keystrokes and some mouse clicks. The tool is self-explanatory and is very easy to navigate. It features everything from user account management, network configuration, to software updates. YaST development has somewhat crumpled in the past, so YaST2 is the more powerful tool.

Red Hat System and Server Settings

"Start Here" was designed to hold all of the tools and applications you need to access when using your system. From your favorite applications to system and configuration tools, the Start Here window provides a central location for using and customizing your system. The System Settings icon includes tools that help you set up your system for personal everyday use (date and time, sound-card detection, users and groups, printing).

The Server Settings includes tools for example for the ApacheWeb Server and DNS/Bind. You must have those server applications installed before these tools appear in this section.

The various GUI-Tools can be accessed from the command line. They all start with "`redhat-config-`". Others are `netconfig`, `checkconfig`, `ntsysv`.

Webmin

Webmin is a web-based interface for system administration of UNIX and Unix-like operating systems. Using any browser that supports tables and forms (and Java for the File Manager module), you can setup user accounts, Apache, DNS, file sharing and so on.

Webmin consists of a simple web server, and a number of CGI programs which directly update system files like `/etc/inetd.conf` and `/etc/passwd`. The web server and all CGI programs are written in Perl version 5, and use no non-standard Perl modules. Supported operation systems are nearly all flavors of Linux and UNIX (including HP-UX, Solaris, and MacOS X).

Webmin is divided into a number of modules that each allow you to administer a single aspect of your system. Modules exist for most common, and many uncommon, system administration tasks. The standard modules provide a graphical interface for: Apache, Squid, Bind, NFS, man pages, Sendmail, Postfix, Samba, and much more. There also exist a wide array of third party modules that provide even more extensive functionality. This book focuses on the standard modules, but may expand to encompass other modules in time.

Upon first logging in, you'll see a row of tabs and a number of icons. The tabs are labeled Webmin, System, Servers, Hardware, Cluster, and Others. Webmin provides a number of configurable options, access control features, and flexible action logging that provides you with maximum flexibility and security of the Webmin server and the various Webmin system administration modules. These features are accessed through the Webmin tab on the index page of Webmin. When you display the Webmin tab, you see icons for Usermin Configuration, Webmin Actions Log, Webmin Configuration, Webmin Servers Index, and Webmin Users. Keep in mind that the modules located under the Webmin tab are for configuring Webmin itself, not the underlying system. So, for example, creating a user in the Webmin Users module will not create a system user, only a Webmin user. Likewise, the Webmin Actions Log module allows you to search and view the Webmin log, not any system or service log that might exist. We'll get to those kinds of options later. For the moment, we're going to skip over Usermin Configuration because Usermin receives full coverage in the next chapter.

Command line tools

Command line tools are most often similar or identical to the ones found in other flavors of Unix. You can find for example `useradd` or `adduser` for adding users. Usually these tools reside in `/usr/bin`. Also the direct manipulation of configuration files is possible using an editor of choice.

Useful stuff

Tips and Tricks for bash

Commands for Moving: Cursor keys should work to navigate.

- Ctrl-a — move to the start of the current line
- Ctrl-e — move to the end of line
- Ctrl-f — move forward one character
- Ctrl-b — Move backward one character

Shell history: Use Cursor Up and Cursor down to navigate through the shell history, and Ctrl-R to search backwards in the shell history Pathname and shell command expansion: Use the Tabulator key for pathname and command expansion. If several possibilities are given, the shell shows them in a list after pressing Tab two times. If no possible expansion exists the shell either signals it with a beep or does nothing.

Tools and Commands

- Midnight Commander (`mc`): A Norton Commander clone, useful to navigate into RPM files.
- Boot floppy: Useful for recovery. Use mkbootdisk or sysadmin tools provided by the distribution to generate a boot floppy.
- `locate`: Are you searching for a file in your filesystem? `locate` is your best friend. It simply searches in a database created by `updatedb`. `updatedb` should be run regularly by cron to have an up-to-date database.
- strace: trace system calls
- `dig` and `host`: (better) alternatives to `nslookup`.
- `tcpdump` and `ethereal` are available for Linux and have the same function as the ports for HP-UX.
- `cfg2html`: The HP internal tool to gather information for a system overview has been ported to Linux. Similar tools are `snapshot` and `sysreport`
- OmniBack: There is an Omniback client for Linux available.

Measuring performance

Linux on-board tools are `vmstat`, `sar`, `free`, and `top`. You can take a look into `/proc/meminfo` for memory information. Measureware was available for Linux, but is discontinued now, as is GlancePlus.

Documentation

Linux HOWTOs: Linux Documentation Projekt (tldp.org).

Sysadmin's Universal Translator

You're looking for a table listing the sysadmin commands of the various UNIX flavours? Rosetta Stone's website answers your question: Rosetta Stone for Unix (bhami.com/rosetta.html)

ServiceGuard for Linux

HP has ported its high availability cluster solution to Linux beginning with version 11.13 in 2002. The now current version is SeviceGuard for Linux (SG/LX) 11.17 running on SUSE Linux Enterprise Server (SLES) 8, 9, and 10 and Red Hat Enterprise Linux (RHEL) AS 3, 4, and 5 on various HP ProLiant and Blade systems, and accompanying storage devices. Also, SG/LX is supported on third-party hardware.

ServiceGuard for Linux is developed in parallel with the HP-UX version. The differences between the two ports are diminishing since the first Linux version and the capabilities are very similar now. Configuration and administration are the same.

Since LVM and LVM2 are not cluster-aware, you can activate volume groups on all nodes and you can write to a file system mounted on more than one node with the devastating effect of file corruption. Be careful with this.

HP supports the combination of ServiceGuard and Red Hat Cluster Suite with Global File System (GFS). GFS is a cluster-aware file system, which is active on all nodes all the time. It uses a distributed locking mechanism to prevent corruption.

ServiceGuard Troubleshooting

Installation problems

- The installation of the RPM is usually a very easy procedure with little effort and problems. You may encounter some dependency issues, which should be solved accordingly (see chapter about installing software)

- One of the prerequisites before installing the software is to configure "bonding" of two or more LAN cards. Bonding involves configuring a bond device and attaching the LAN cards to it. This serves as a fallback for network failures.

- After installing a ServiceGuard patch (of the form SGLX 000xx), check the version of the deadman driver (see `$SGROOT/drivers/README`).

Configuration problems

- The most common problems for configuration involve the access between the nodes. Look for the right and complete entries in `/etc/hosts` and `cmclnodelist` (if used).

- If Lock LUN problems occur, check and test access to the storage device, read/write permissions, and configuration files.

Runtime problems

- Problems during runtime such as a cluster rebuild, node hangs/freezes, and reboots are often caused by software bugs or network problems.

- Check the `/etc/messages` file and `dmesg` outputs for error messages.

- Check the kernel and SG/LX patch versions and compare them to the currently available patches at HP's ITRC website.

SAP and Oracle on Linux

Linux finds its place on a lot of systems doing tasks like web serving, file serving, print serving, acting as firewalls, or e-mail servers. In recent years more and more business-critical application like databases or application servers are installed on Linux.

SAP and Oracle are the most prominent commercial offerings in their fields. Both vendors support Linux as a "normal" platform now, as they do with Windows, HP-UX, Solaris, or AIX (and others). Deploying Oracle databases on Linux is not exotic anymore, it is daily business.

SAP

SAP supports SUSE and Red Hat Enterprise versions from SLES 7 to 10 and RHEL 2.1 to 5. Previously, SAP supported a special kernel version only, but changed this to generally support all original and official update kernels of any supported distribution. There are still recommended kernel versions, but any other kernel version released by SUSE or Red Hat should do ok. Recommended kernel versions look like this:

- RHEL 4 for x86: 2.6.9-22.EL (U2)
- SLES 9 for x86: 2.6.5-7.244 (SP3)

Linux support and recommended kernels can be found on the SAP website at: www.sap.com/linux/platforms/index.asp. You can find a list of supported HP hardware (32-bit and 64-bit ProLiant, and 64-bit Integrity systems) here: www.sap.com/linux/platforms/hp.asp.

SAP supports the following databases on Linux:

- IBM DB2/UDB
- Informix
- Oracle
- SAP DB/MaxDB

SAP claims to use Linux standard APIs only, so no conflicts with other applications should arise over what library is installed. The mySAP Business Suite supports Database Server, Application Server and the graphical frontend for Linux.

Oracle

Oracle supports the Red Hat, SUSE, and Asianux distributions for use with Oracle databases. Like SAP, Oracle does recommend specific kernels, but supports all kernels officially released by the distributors. There are no specific hardware certifications, Oracle supports Linux on the platforms that are certified by the hardware vendor for Linux.

You should change kernel parameters as recommended by Oracle (as on every other operating system). You can find a FAQ about Oracle on Linux here: www.oracle.com/technology/tech/linux/htdocs/oracleonlinuxfaq.html.

Bibliography

Linus Torvalds (with David Diamond): Just for Fun: The Story of an Accidental Revolutionary, HarperBusiness (2002)

Daniel P. Bovet, Marco Cesati: Understanding the Linux Kernel, 2nd Edition, O'Reilly (2002) — covers Linux kernel v.2.4

Daniel P. Bovet, Marco Cesati: Understanding the Linux Kernel, 3rd Edition, O'Reilly (2005) — covers Linux kernel v.2.6

Martin F. Krafft: The Debian System—Concepts and Techniques, No Starch Press (2005)