LPI exam 201 prep: Hardware

Intermediate Level Administration (LPIC-2) topic 204

Skill Level: Intermediate

David Mertz, Ph.D. (mertz@gnosis.cx)
Developer
Gnosis Software

Brad Huntting (huntting@glarp.com)
Mathematician
University of Colorado

02 Sep 2005

In this tutorial, David Mertz and Brad Huntting continue preparing you to take the Linux Professional Institute® Intermediate Level Administration (LPIC-2) Exam 201. In this fourth of eight tutorials, you learn how to add and configure hardware to a Linux™ system, including RAID arrays, PCMCIA cards, other storage devices, displays, video controllers, and other components.

Section 1. Before you start

Learn what these tutorials can teach you and how you can get the most from them.

About this series

The Linux Professional Institute (LPI) certifies Linux system administrators at junior and intermediate levels. To attain each level of certification, you must pass two LPI exams.

Each exam covers several topics, and each topic has a weight. The weights indicate the relative importance of each topic. Very roughly, expect more questions on the exam for topics with higher weight. The topics and their weights for LPI exam 201 are:

Topic 201
Linux kernel (weight 5).

Topic 202
System startup (weight 5).

Topic 203
Filesystem (weight 10).

Topic 204
Hardware (weight 8). The focus of this tutorial.

Topic 209
File and service sharing (weight 8).

Topic 211
System maintenance (weight 4).

Topic 213
System customization and automation (weight 3).

Topic 214
Troubleshooting (weight 6).

The Linux Professional Institute does not endorse any third-party exam preparation material or techniques in particular. For details, please contact info@lpi.org.

About this tutorial

Welcome to "Hardware," the fourth of eight tutorials designed to prepare you for LPI exam 201. In this tutorial, you learn how to add and configure hardware to a Linux system, including RAID arrays, PCMCIA cards, other storage devices, displays, video controllers, and other components.

The tutorial is organized according to the LPI objectives for this topic, as follows:

2.204.1 Configuring RAID (weight 2)
You will be able to configure and implement software RAID. This objective includes using mkraid tools and configuring RAID 0, 1, and 5.

2.204.2 Adding new hardware (weight 3)
You will be able to configure internal and external devices for a system including new hard disks, dumb terminal devices, serial UPS devices, multi-port serial cards, and LCD panels.

2.204.3 Software and kernel configuration (weight 2)
You will be able to configure kernel options to support various hardware devices including UDMA66 drives and IDE CD burners. This objective includes using LVM (Logical Volume Manager) to manage hard disk drives and partitions as well as software tools to interact with hard disk settings.

2.204.4 Configuring PCMCIA devices (weight 1)

You will be able to configure a Linux installation to include PCMCIA support. This objective includes configuring PCMCIA devices, such as Ethernet adapters, to autodetect when inserted.

While you will often use userland tools to work with hardware devices, for the most part, basic support for those devices is provided by a Linux base kernel, kernel modules, or both. One notable exception to the close connection between the Linux kernel and hardware devices is in graphics cards and computer displays. A simple console text display is handled well enough by the Linux kernel (and even some graphics with framebuffer support), but generally advanced capabilities of graphics cards are controlled by XFree86 or more recently X.Org, X11 drivers. Almost all distributions include X11 and associated window managers and desktop environments; but for non-desktop servers, using X11 may be superfluous.

Prerequisites

To get the most from this tutorial, you should already have a basic knowledge of Linux and a working Linux system on which you can practice the commands covered in this tutorial.

In addition, some information on adding hardware is covered in two other tutorials: "LPI exam 201 prep (topic 201): Linux kernel" and "LPI exam 201 prep (topic 203): Filesystems." The LPI exam on hardware expects familiarity with kernel and filesystem tuning, so please refer to those other tutorials during exam preparation.

Section 2. Configuring RAID

What is RAID?

RAID (Redundant Array of Inexpensive Disks) provides mechanisms to combine multiple partitions on different hard drives into larger or more resilient virtual hard drives. Numerous RAID levels were initially defined, but only three remain in common use: RAID-0 (disk striping), RAID-1 (mirroring), and RAID-5 (striping with parity information). RAID-4 is also occasionally used; it is similar to RAID-5 but puts parity information on exactly one drive rather than distributing it.

This tutorial discusses the *"new-style" RAID* under Linux (the default for 2.4 and 2.6 kernels with backports to earlier kernels available). The "old-style" RAID initially present in 2.0 and 2.2 kernels was buggy and should not be used. Specifically, "new-style" means the 0.90 RAID layer made by Ingo Molnar and others.

Using a RAID array

There are two basic parts to using a RAID array. The simple part is mounting the RAID device. Once a RAID (virtual) device is configured, it looks to the `mount` command the same as a regular block device partition. A RAID device is named as /dev/mdN once it is created, so you might mount it as:

```
% mount /dev/md0 /home
```

You can also include a line in /etc/fstab to mount the RAID virtual partition (this is usually the preferred method). The device driver reads superblocks of raw partitions to assemble a RAID partition once it has been created.

The more complicated part (more detailed, anyway) involves creating the RAID device out of relevant raw partitions. You can create a RAID partition with the tool `mkraid` combined with an /etc/raidtab configuration file.

You can also use the newer tool `mdadm`, which can usually operate without the need for a separate configuration file. In most distributions, `mdadm` is supplanting `raidtools` (which includes `mkraid`), but this tutorial discusses `mkraid` to follow the LPI exam objectives. The concepts are similar either way, but you should read the `mdadm` manpage to learn about its switches.

The layout of /etc/raidtab

The following definitions are used in the /etc/raidtab file to describe the components of a RAID. This list is not exhaustive.

- `raiddev`: The virtual disk partition provided by RAID (/dev/md?). This is the device that `mkfs` and fsck work with, and that is mounted in the same way as an actual hard disk partition.
- `raid-disk`: The underlying partitions used to build a RAID. They should be marked (with fdisk or similar tools) as partition type 0xFD.
- `spare-disk`: These disks (typically there's only one) normally lie unused. When one of the raid disks fails the spare disk is brought online as a replacement.

Configuring RAID-0

RAID-0 or "disk striping" provides more disk I/O bandwidth at the cost of less reliability (a failure in any one of the raid-disks and you loose the entire RAID device). For example the following /etc/raidtab entry sets up a RAID-0 device:

```
raiddev /dev/md0
    raid-level      0
    nr-raid-disks   2
    nr-spare-disks  0
```

```
    chunk-size      32
    persistent-superblock   1
    device          /dev/sda2
    raid-disk       0
    device          /dev/sdb2
    raid-disk       1
```

This defines a RAID-0 virtual device called /dev/md0. The first 32 KB chunk of /dev/md0 is allocated to /dev/sda2, the next 32 KB go on /dev/sdb2, the third chunk is on /dev/sda2, etc.

To actually create the device, simply run:

```
% sudo mkraid /dev/md0
```

If you use `mdadm`, switches will configure these options rather than the /etc/raidtab file.

Configuring RAID-1

RAID-1 or "disk mirroring" simply keeps duplicate copies of the data on both block devices. RAID-1 gracefully handles a drive failure with no noticeable drop in performance. RAID-1 is generally considered expensive since half of your disk space is redundant. For example:

```
raiddev /dev/md0
    raid-level      1
    nr-raid-disks   2
    nr-spare-disks  1
    persistent-superblock 1
    device          /dev/sdb6
    raid-disk       0
    device          /dev/sdc5
    raid-disk       1
    device          /dev/sdd5
    spare-disk      0
```

Data written to /dev/md0 will be saved on both /dev/sdb6 and /dev/sdc5. /dev/sdd5 is configured as a *hot spare*. In the event /dev/sdb6 or /dev/sdc5 malfunctions, /dev/sdd5 will be populated with data and brought online as a replacement.

Configuring RAID-5

RAID-5 requires at least three drives and uses error correction to get most of the benefits of disk striping but with the ability to survive a single drive failure. On the positive side, it requires only one extra redundant drive. On the negative side, RAID-5 is more complex; when a drive does fail, it drops into *degraded mode* which can dramatically impact I/O performance until a spare-disk can be brought online and repopulated.

```
raiddev /dev/md0
    raid-level        5
    nr-raid-disks     7
    nr-spare-disks    0
    persistent-superblock 1
    parity-algorithm  left-symmetric
    chunk-size        32
    device            /dev/sda3
```

```
      raid-disk         0
      device            /dev/sdb1
      raid-disk         1
      device            /dev/sdc1
      raid-disk         2
      device            /dev/sdd1
      raid-disk         3
      device            /dev/sde1
      raid-disk         4
      device            /dev/sdf1
      raid-disk         5
      device            /dev/sdg1
      raid-disk         6
```

Using mke2fs or mke3fs

If you format a RAID-5 virtual device using e2fs or e3fs, you should pay attention to the *stride* option. The `-R stride=nn` option will allow mke2fs to better place different ext2-specific data structures in an intelligent way on the RAID device.

If the chunk size is 32 KB, it means that 32 KB of consecutive data will reside on one disk. If an ext2 filesystem has 4 KB block size then there will be eight filesystem blocks in one array chunk. We can indicate this information to the filesystem by running:

```
 % mke2fs -b 4096 -R stride=8 /dev/md0
```

RAID-5 performance is greatly enhanced by providing the filesystem with correct stride information.

Kernel support and failures

Enabling the *persistent-superblock* feature allows the kernel to start the RAID automatically at boot time. New-style RAID uses the persistent superblock and is supported in 2.4 and 2.6 kernels. Patches are available to retrofit 2.0 and 2.2 kernels.

Here's what happens when a drive fails:

- **RAID-0**: All data is lost>
- **RAID-1/RAID-5**: The failed drive is taken offline and the spare disk (if it exists) is brought online and populated with data.

The document "The Software-RAID HOWTO" in the Linux HOWTO project discusses swapping in drives for failed or updated drives, including when such drives are hot-swappable and when a reboot will be required. Generally, SCSI (or Firewire) are hot-swappable while IDE drives are not.

Section 3. Adding new hardware

About hardware

Linux, especially in recent versions, has an amazingly robust and broad capability to utilize a variety of hardware devices. In general, there are two levels of support that you might need to worry about in supporting hardware. At a first level, there is a question of supporting a device at a basic system level; doing this is almost always by means of loading appropriate kernel modules for your hardware device.

At a second level, some devices need more-or-less separate support from the X11R6 subsystem: generally either XFree86 or X.Org (very rarely, a commercial X11 subsystem is used, but this tutorial does not discuss that).

Support for the main hot-swappable device categories -- such as those using PCMCIA or USB interfaces -- are covered in their own topic sections to follow.

About X11

A quick note: X.Org is essentially the successor project to XFree86 (technically a fork). While XFree86 has not officially folded, almost all distribution support has shifted to X.Org because of licensing issues. Fortunately, except for some minor renaming of configuration files, the initial code base for both forks is largely the same; some newer special features are more likely to be supported in X.Org only.

X11R6 is a system for (networked) presentation of graphical applications on a user workstation. Perhaps counter-intuitively, an "X server" is the physical machine that a user concretely interacts with using its keyboard, pointing device(s), video card, display monitor, etc. An "X client" is the physical machine that devotes CPU time, disk space, and other non-presentation resources to running an application. In many or most Linux systems, the X server and X client are the self-same physical machine and a very efficient local communication channel is used for an application to communicate with user I/O devices.

An X server -- such as X.Org -- needs to provide device support for the I/O devices with which a user will interact with an application. Overwhelmingly, where there is any difficulty, it has to do with configuring video cards and display monitors. Fortunately, this difficulty has decreased in recent X.Org/XFree86 versions with much more automatic detection performed successfully. Technically, an X server also needs to support input devices -- keyboards and mice -- but that is usually fairly painless since these are well standardized interfaces. Everything else -- disk access, memory, network interfaces, printers, special devices like scanners, and so on -- are all handled by the X client application, generally by the underlying Linux kernel.

Kernel device support

Almost everything you need to know about device support in the Linux kernel boils down to finding, compiling, and loading the right kernel modules. That topic is

covered extensively in the Topic 201 tutorial -- readers should consult that tutorial for most issues.

To review kernel modules, the main tools a system administrator needs to think about are lsmod, insmod, and modprobe. Also rmmod to a lesser extent. The tools lsmod, insmod and rmmod are low-level tools to, respectively, list, insert, and remove kernel modules for a running Linux kernel. modprobe performs these same functions at a higher level by examining dependencies, then making appropriate calls to insmod and rmmod behind the scenes.

Examining hardware devices

Several utilities are useful for scoping out what hardware you actually have available. The tool lspci provides detailed information on findable PCI devices (including those over PCMCIA or USB buses, in many cases). Correspondingly setpci can configure devices on PCI buses. lspnp lists plug-and-play BIOS device nodes and resources. lsusb similarly examines USB devices (and has a setpnp to modify configurations).

Setting up an X11 server (part one)

Basically, X.Org (or XFree86) come with a whole lot of video drivers and other peripheral drivers; you need to choose the right ones to use. Ultimately, the configuration for an X server lives in the rather detailed, and somewhat cryptic, file /etc/X11/xorg.conf (or xfree86.conf). A couple standard utilities can be used for somewhat friendlier modification of this file, but ultimately a text editor works. Some frontends included with X.Org itself include xorgcfg for graphical configuration (assuming you have it working well enough to use that) and xorgconfig for a text-based setup tool. Many Linux distributions package friendlier frontends.

The tool SuperProbe is often useful in detecting the model of your video card. You may also consult the database /usr/X11R6/lib/X11/Cards for detailed information on supported video cards.

Setting up an X11 server (part two)

Within the configuration file /etc/X11/xorg.conf, you should create a series of "Section" ... "EndSection" blocks, each of which defines a number of details and options about a particular device. These section names consist of:

```
   * Files:          File pathnames
   * ServerFlags:    Server flags
   * Module:         Dynamic module loading
   * InputDevice:    Input device description
   * Device:         Graphics device description
   * VideoAdaptor:   Xv video adaptor description
   * Monitor:        Monitor description
   * Modes:          Video modes descriptions
   * Screen:         Screen configuration
   * ServerLayout:   Overall layout
   * DRI:            DRI-specific configuration
```

```
   * Vendor:         Vendor-specific configuration
```

Setting up an X11 server (part three)

Among the sections, `Screen` acts as a master configuration for the display system. For example, you might define several `Monitor` sections, but select the one actually used with:

```
       Section "Screen"
           Identifier    "Default Screen"
           Device        "My Video Card"
           Monitor       "Current Monitor"
           DefaultDepth  24
           SubSection "Display:
                 Depth       24
                 Modes       "1280x1024" "1024x768" "800x600"
           EndSubSection
           # more subsections and directives
       Endsection
```

The section named `ServerLayout` is the real "master" configuration -- it refers to both the `Screen` to use and to various `InputDevice` sections. But if you have a problem, it is almost always with selecting the right `Device` or `Monitor`. Fortunately, DPMS monitors nowadays usually obviate the need to set painful `Modeline` options (in the bad old days, you needed to locate very specific timings on your monitors scan rates; usually DPMS handles this for you now).

Section 4. Configuring PCMCIA devices

About PCMCIA

PCMCIA devices are also sometimes called PC-Card devices. These (thick) credit-card sized peripherals are generally designed to be easily hot-swappable and transportable and are used most widely in notebook computers. However, some desktop or server configurations also have PCMCIA readers, sometimes in an external chassis connected via one of several possible buses (a special PCI or ISA card, a USB translator, etc). A variety of peripherals have been created in PCMCIA form factor, including wireless and Ethernet adaptors, microdrives, flash drives, modems, SCSI adapters, and other special-purpose devices.

Technically, a PCMCIA interface is a layer used to connect to an underlying ISA or PCI bus. For the most part, the translation is transparent -- the very same kernel modules or other tools that communicate with an ISA or PCI device will be used to manage the same protocol provided via PCMCIA. The only real special issue with PCMCIA devices is recognition of the insertion event and of the card type whose drivers should load.

Nowadays, the PCMCIA peripheral packaging is being eclipsed by USB and/or

Firewire devices. While PCMCIA is a bit more convenient as a physical form-factor (usually hiding cards in the machine chassis), USB is closer to universal on a range of machines. As a result, many devices that have been packaged as PCMCIA in the past might now be packaged as USB "dongle" style devices; these are more readily available at retail outlets.

Recognizing a PCMCIA device (part one)

In modern kernels -- 2.4 and above -- PCMCIA support is available as a kernel module. Modern distribution will include this support, but if you compile a custom kernel, include the options `CONFIG_HOTPLUG`, `CONFIG_PCMCIA`, and `CONFIG_CARDBUS`. The same support was previously available in separately in the pcmcia-cs package.

The modules `pcmcia_core` and `pcmcia` support loading PCMCIA devices. `yenta_socket` is also generally loaded to support the CardBus interface (PCI-over-PCMCIA):

```
% lsmod | egrep '(yenta)|(pcmcia)'
pcmcia                 21380  3 atmel_cs
yenta_socket           19584  1
pcmcia_core            53568  3 atmel_cs,pcmcia,yenta_socket
```

Once a card is inserted into a PCMCIA slot, the daemon `cardmgr` looks up a card in the database /etc/pcmcia/config then loads appropriate supporting drivers as needed.

Recognizing a PCMCIA device (part two)

Let us take a look at the PCMCIA recognition in action. I inserted a card into a Linux laptop with a PCMCIA slot and with the previously listed kernel module support. I can use the tool `cardctl` to see what information this peripheral provided:

```
% cardctl ident
Socket 0:
  product info: "Belkin", "11Mbps-Wireless-Notebook-Network-Adapter"
  manfid: 0x01bf, 0x3302
  function: 6 (network)
```

This information is provided by the `pcmcia_core` kernel module by querying the physical card. Once the identification is available, `cardmgr` scans the database to figure out what driver(s) to load. Something like:

```
% grep -C 1 '0x01bf,0x3302' /etc/pcmcia/config
card "Belkin FSD6020 v2"
  manfid 0x01bf,0x3302
  bind "atmel_cs"
```

In this case, we want the kernel module `atmel_cs` to support the wireless interface this card provides. You can see what actually got loaded by looking at either /var/lib/pcmcia/stab or /var/run/stab, depending on your system:

```
 % cat /var/run/stab
 Socket 0: Belkin FSD6020 v2
```

```
0       network atmel_cs        0       eth2
```

Debugging a PCMCIA device

In the above example, everything worked seamlessly. The card was recognized, drivers loaded, and the capabilities ready to go. That is the best case. If things are not as smooth, you might not find a driver to load.

If you are confident that your PCMCIA device can use an existing driver (for example, it is compatible with another chipset), you can manually run `insmod` to load the appropriate kernel module. Or if you use this card repeatedly, you can edit /etc/pcmcia/config to support your card, indicating the needed driver. However, guessing a needed module is unlikely to succeed -- you need to make sure your card really is compatible with some other known PCMCIA card.

Loading PCMCIA devices can be customized with the setup scripts in /etc/pcmcia/, each named for a category of function. For example, when an 802.11b card like the previous example loads, the script /etc/pcmcia/wireless runs. You can customize these scripts if your device has special setup requirements.

Using "schemes" for different configurations

If you need to use a PCMCIA device in multiple configurations, you may use the `cardctl scheme` command to set (or query) a configuration. For example:

```
% sudo cardctl scheme foo
checking: eth2
/sbin/ifdown: interface eth2 not configured
Changing scheme to 'foo'...
Ignoring unknown interface eth2=eth2.
% cardctl scheme
Current scheme: 'foo'.
```

In this case, I have not actually defined the `foo` scheme, but in general, if you change a scheme, device reconfiguration is attempted. Schemes may be used in setup scripts by examining the $ADDRESS variable:

```
# in /etc/pcmcia/network.opts (called by /etc/pcmcia/network)
case "$ADDRESS" in
work,*,*,*)
    # definitions for network in work scheme ...
    ;;
default,*,*,*)
    # definitions for network in default scheme ...
    ;;
esac
```

You may of course, set schemes in initialization scripts or via other triggering events (in a `cron` job, via a GUI interface, etc.).

Section 5. Configuring Universal Serial Bus devices

About USB

As the section on PCMCIA mentioned, USB is somewhat newer technology that has largely eclipsed PCMCIA in importance. USB allows chaining of up to 127 devices on the same bus using a flexible radial topology of hubs and devices. USB comes in several versions with increasing speeds, the latest being 2.0. The latest USB version theoretically supports up to 480 MBsec. USB 1.1 supported the lower speed of 12 MBsec. In practice, there are a lot of reasons that particular devices might, in fact, operate much slower than these theoretical numbers -- but it is still a reasonably fast bus interface.

Recognizing a USB device (part one)

At an administrative level, USB works very similarly to PCMCIA. The kernel module is `usbcore`. Support is better in 2.4+ kernels than in earlier 2.2 kernels. Above the `usbcore` level, one of several kernel modules support: uhci, uhci_hcp, ohci, ohci_hcp, ehci, ehci_hcp. Exactly which kernel modules you need depends on the chipset your machine uses; and in the case of ehci whether it supports USB 2.0 high speed. Generally, if your machine support ehci (or ehci_hcp), you will also want a backward-compatible ehci module loaded. Brad Hards' "The Linux USB sub-system" contains details on exactly which chipsets support which kernel modules. For multiuse kernels, you should just compile in all the USB modules.

Assuming you get a kernel with the correct support, the hotplug subsystem should handle loading any drivers needed for a specific inserted USB device. The file /proc/bus/usb/devices contains detailed information on the currently available USB devices (both hubs and peripherals).

Recognizing a USB device (part two)

Normally the USB bus is mounted as a dynamically generated filesystem similar to the /proc/ filesystem. The filesystem type is known as either usbdevfs or usbfs. Depending on your distribution, /proc/bus/usb/ might get mounted either in initialization scripts such as /etc/rcS.d/S02mountvirtfs or via an /etc/fstab configuration. In the latter case, you might have a line like:

```
# /etc/fstab
none /proc/bus/usb usbdevfs defaults 0 0
```

Otherwise, an initialization script might run something like:

```
mount -t usbdevfs none /proc/bus/usb
```

The recognition of devices and control over the USB subsystem is contained in the /etc/hotplug/, especially within /etc/hotplug/usb.rc and /etc/hotplug/usb.agent. Inserting a USB device will `modprobe` a driver. You may customize the initialization of a device further by creating a /etc/hotplug/usb/$DRIVER script for your particular peripheral.

Resources

Learn

- At the LPIC Program, find task lists, sample questions, and detailed objectives for the three levels of the Linux Professional Institute's Linux system administration certification.
- "Common threads: Advanced filesystem implementor's guide, Parts 1 - 13" (developerWorks, starting June 2001) is an excellent series on Linux filesystems.
- "Understanding Linux configuration files" (developerWorks, December 2001) explains configuration files on a Linux system that control user permissions, system applications, daemons, services, and other administrative tasks in a multi-user, multi-tasking environment.
- Find more resources for Linux developers in the developerWorks Linux zone.

Get products and technologies

- Order the SEK for Linux, a two-DVD set containing the latest IBM trial software for Linux from DB2®, Lotus®, Rational®, Tivoli®, and WebSphere®.
- Build your next development project on Linux with IBM trial software, available for download directly from developerWorks.

Discuss

- Participate in the discussion forum for this content.
- Get involved in the developerWorks community by participating in developerWorks blogs.

About the authors

David Mertz, Ph.D.
David Mertz is Turing complete, but probably would not pass the Turing Test. For more on his life, see his personal Web page. He's been writing the developerWorks columns *Charming Python* and *XML Matters* since 2000. Check out his book *Text Processing in Python* .

Brad Huntting
Brad has been doing UNIX® systems administration and network engineering for about 14 years at several companies. He is currently working on a Ph.D. in Applied Mathematics at the University of Colorado in Boulder, and pays the bills by doing UNIX support for the Computer Science department.