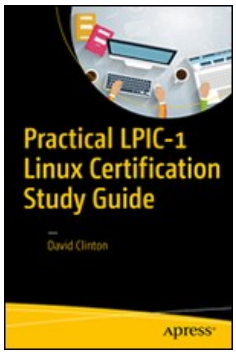

Practical LPIC-1 Linux Certification Study Guide

by David Clinton

Apress. (c) 2016. Copying Prohibited.

Reprinted for Rene Hernandez Terrones renet@mx1.ibm.com, IBM

renet@mx1.ibm.com

Reprinted with permission as a subscription benefit of **Skillport**,
http://skillport.books24x7.com/

All rights reserved. Reproduction and/or distribution in whole or in part in electronic,paper or other forms without written permission is prohibited.

Chapter 1: Topic 101: System Architecture

© David Clinton 2016
D. Clinton, *Practical LPIC-1 Linux Certification Study Guide*,
DOI 10.1007/978-1-4842-2358-1_1

Device Management: The Linux Boot Process

Unless you end up working exclusively with virtual machines or on a cloud platform like Amazon Web Service, you'll need to know how to do techie things like putting together real machines and swapping out failed drives. However, since those skills aren't part of the Linux Professional Institute Certification (LPIC) exam curriculum, I won't focus on them in this book. Instead, I'll begin with booting a working computer.

Whether you're reading this book because you want to learn more about Linux or because you want to pass the LPIC-1 exam, you will need to know what happens when a machine is powered on and how the operating system wakes itself up and readies itself for a day of work. Depending on your particular hardware and the way it's configured, the firmware that gets things going will be either some flavor of BIOS (Basic Input/Output System) or UEFI (Intel's Unified Extended Firmware Interface).

As illustrated in Figure 1-1, the firmware will take an inventory of the hardware environment in which it finds itself and search for a drive that has a Master Boot Record (MBR) living within the first 512 (or, in some cases, 4096) bytes. The MBR should contain partition and filesystem information, telling BIOS that this is a boot drive and where it can find a mountable filesystem.

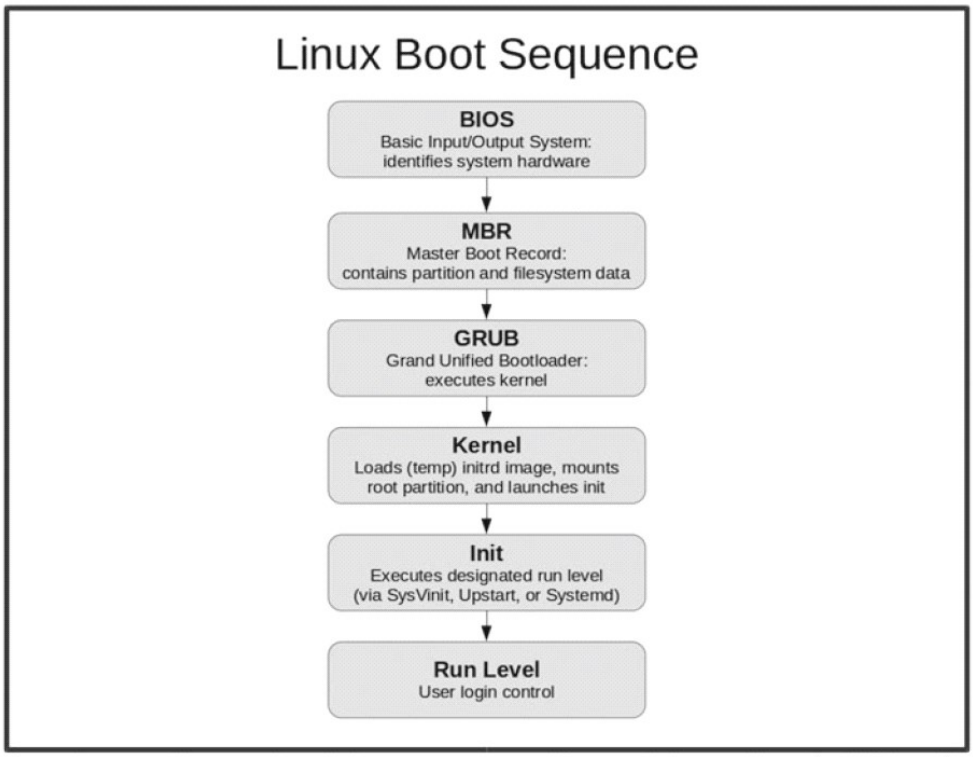

Figure 1-1: The six key steps involved in booting a Linux operating system.

On most modern Linux systems, the MBR is actually made up of nothing but a 512 byte file called boot.img. This file, known as GRUB Stage 1 (GRUB stands for GRand Unified Bootloader), really does nothing more than read and load into RAM (random access memory) the first sector of a larger image called core.img. Core.img, also known as GRUB Stage 1.5, will start executing the kernel and filesystem, which is normally found in the /boot/grub directory.

The images that launch from /boot/grub are known as GRUB Stage 2. In older versions, the system would use the initrd (init ramdisk) image to build a temporary filesystem on a block device created especially for it. More recently, a temporary filesystem (tmpfs) is mounted directly into memory—without the need of a block device—and an image called initramfs is extracted into it. Both methods are commonly known as initrd.

Once Stage 2 is up and running, you will have the operating system core loaded into RAM, waiting for you to take control.

> **Note** This is how things work right now. The LPI exam will also expect you to be familiar with an older legacy version of GRUB, now known as GRUB version 1. That's GRUB **version** 1, mind you, which is not to be confused with GRUB **Stage** 1, 1.5, or 2! The GRUB we're all using today is known as GRUB version 2. You think that's confusing? Just be grateful that they don't still expect you to know about the LILO bootloader!

Reprinted for IBM/renet@mx1.ibm.com, IBM
Apress, David Clinton (c) 2016, Copying Prohibited

Besides orchestrating the boot process, GRUB will also present you with a startup menu from which you can control the software your system will load.

Note In case the menu doesn't appear for you during the start sequence, you can force it to display by pressing the right Shift key as the computer boots. This might sometimes be a bit tricky: I've seen PCs configured to boot to solid state drives that load so quickly, there almost isn't time to hit Shift before reaching the login screen. Sadly, I face no such problems on my office workstation.

As you can see from Figure 1-2, the GRUB menu allows you to choose between booting directly into the most recent Ubuntu image currently installed on the system, running a memory test, or working through some advanced options.

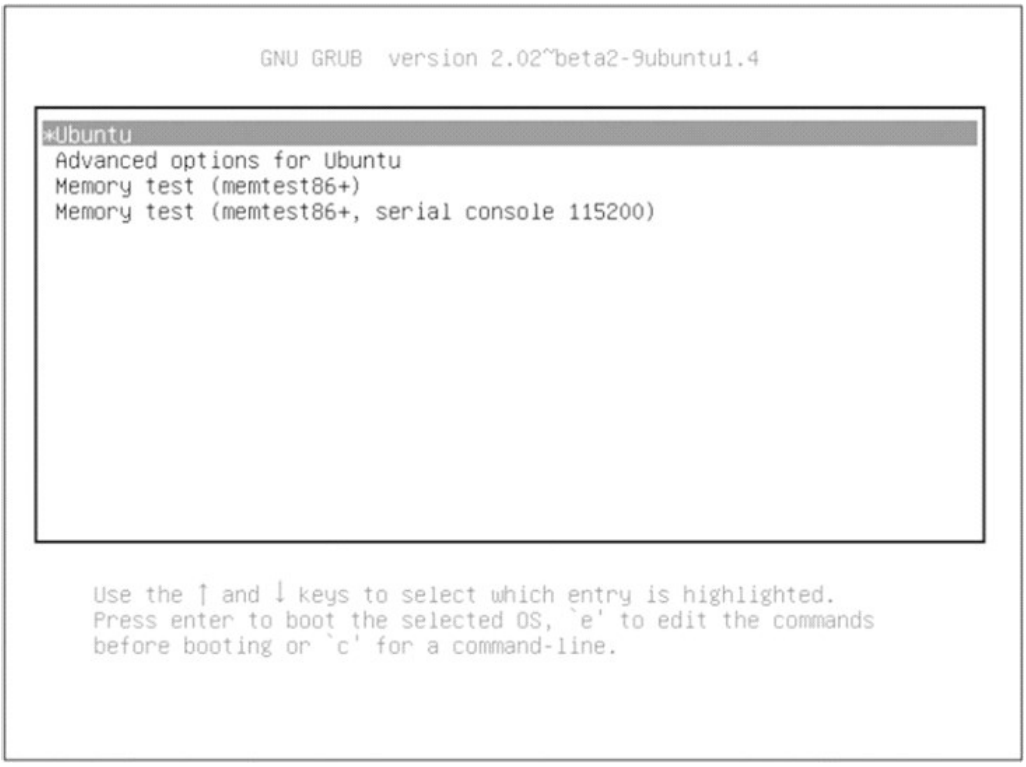

Figure 1-2: A typical GRUB version 2 boot menu

The Advanced menu (see Figure 1-3) allows you to run in recovery mode or, if there happens to be any available, to select from older kernel images. This can be really useful if you've recently run an operating system upgrade that broke something important.

Reprinted for IBM/renet@mx1.ibm.com, IBM

Apress, David Clinton (c) 2016, Copying Prohibited

Figure 1-3: A GRUB advanced menu (accessed by selecting "Advanced options" in the main menu window)

Pressing "e" with a particular image highlighted will let you edit its boot parameters (see Figure 1-4). I will warn you that spelling—and syntax—really, really count here. No, really. Making even a tiny mistake with these parameters can leave your PC unbootable, or even worse, bootable, but profoundly insecure. Of course, these things can always be fixed by coming back to the GRUB menu and trying again—and I won't deny the significant educational opportunities this will provide. But I'll bet that, given a choice, you'd probably prefer a quiet, peaceful existence.

Figure 1-4: A GRUB boot parameters page (accessed by hitting "e" while an item is highlighted in the main menu window)

Pressing "c" or Ctrl+c will open a limited command-line session.

Note You may be interested—or perhaps horrified—to know that adding rw init=/bin/ bash to your boot parameters will open a full root

Reprinted for IBM/renet@mx1.ibm.com, IBM Apress, David Clinton (c) 2016, Copying Prohibited

session for anyone who happens to push the power button on your PC. If you think you might need this kind of access, I would advise you to create a secure BIOS or GRUB password to protect yourself.

Troubleshooting

Linux administrators are seldom needed when everything is chugging along happily. We normally earn our glory by standing tall when everything around us is falling apart. So you should definitely expect frantic calls complaining about black screens or strange flashing dashes instead of the cute kitten videos your user had been expecting.

If a Linux computer fails to boot, your first job is to properly diagnose the problem. And the first place you should probably look to for help is your system logs. A text record of just about everything that happens on a Linux system is saved to at least one plain text log file. The three files you should search for boot-related trouble are dmesg, kern.log, and boot.log, all of which usually live in the /var/log/ directory (some of these logs may not exist on distributions running the newer Systemd process manager).

Since, however, these logs can easily contain thousands of entries each, you may need some help zeroing in on the information you're looking for. One useful approach is to quickly scroll through say, kern.log, watching the time stamps at the beginning of each line. A longer pause between entries or a full stop might be an indication of something going wrong.

You might also want to call on some command-line tools for help. Cat will print an entire file to the screen, but often far too fast for you to read. By piping the output to grep, you can focus on only the lines that interest you. Here's an example:

```
cat /var/log/dmesg | grep memory
```

By the way, the pipe symbol (|) is typed by pressing the Shift+\ key combination. You're definitely going to need that later. (I'll discuss this kind of text manipulation a lot more in the coming chapters.)

I'm going to bet that there's something about this whole discussion that's been bothering you: if there's something preventing Linux from booting properly, how on earth are you ever going to access the log files in the first place?

Good question and I'm glad you asked. And here's my answer. As long as the hard drive is still spinning properly, you can almost always boot your computer into a live Linux session from a Linux iso file that's been written to a USB or CDRom drive, and then find and mount the drive that's giving you trouble. From there, you can navigate to the relevant log files. Here's how that might work.

You can search for all attached block devices using the command lsblk (List BLocK devices):

```
lsblk
```

Once you find your drive, create a new directory to use as a mount point:

```
sudo mkdir /tempdrive
```

Next, mount the drive to the directory you created (assuming that lsblk told you that your drive is called sdb1):

```
sudo mount /dev/sdb1 /tempdrive
```

Finally, navigate to the log directory on your drive:

```
cd /tempdrive/var/log
```

Don't worry, I'm going to talk a lot more about using each of those tools later. For now, though, I should very briefly introduce you to the way Linux manages system access.

Normal users are, by default, only allowed to edit files that they have created. System files, like those in the /var or /etc directory hierarchies, are normally accessible exclusively to the root user, or to users who have been given administrative authority. In many Linux distributions (like Ubuntu), users who need admin powers are added to the sudo group, which allows them to preface any command with the word sudo (as in sudo mkdir /tempdrive).

Invoking sudo and then entering a password temporarily gives the user full admin authority. From a security perspective, taking powers only when needed is far preferred to actually logging in as the root user.

Run Levels

There's more than one way to run a Linux computer. And, coming from the rough and tumble open source world as Linux does, there's more than one way to *control* the multiple ways you can run a Linux computer. I'll get back to that in just a minute or two.

But let's start at the beginning. One of Linux's greatest strengths is the ability for multiple users to log in and work simultaneously on a single server. This permits all kinds of savings in cost and labor and, to a large degree, is what lies behind the incredible flexibility of container virtualization.

However, there may be times when you just want to be alone. Perhaps something's gone badly wrong and you have to track it down and fix it before it gets worse. You don't need a bunch of your friends splashing around in the same pool while you work. Or maybe you suspect that your system has been compromised and there are unauthorized users lurking about. Whatever the case, you might sometimes want to temporarily change the way Linux behaves.

Linux run levels allow you to define whether your OS will be available for everyone or just a single admin user, or whether it will provide network services or graphic desktop support. Technically speaking, shutting down and rebooting your computer are also done through their own run

Reprinted for IBM/renet@mx1.ibm.com, IBM Apress, David Clinton (c) 2016, Copying Prohibited

levels.

While you will find minor differences among Linux distributions, here are the standard run levels and their designated numbers:

Boot parameter:

0: Halt

1: Single user mode

2: Multi-user, without NFS

3: Full multi-user mode

4: Unused

5: X11

6: Reboot

Run levels can be invoked from the command line using either init or telinit Running

```
init 6
```

for instance, would cause your computer to reboot. On some distributions, you can also use commands like "shutdown" to—well—shut down. Thus:

```
sudo shutdown -h now
```

would halt ("h") a system right away and

```
sudo shutdown -h 5
```

would shut down the system, but only after 5 minutes, and

```
sudo shutdown -r now
```

would reboot.

Incidentally, since there might be other users logged into the system at the time you decide to change the run level, the shutdown command will automatically send a message to the terminals of all other logged in users, warning them of the coming change.

You can also send messages between terminals using the wall command (these messages will, of course, not reach graphical user interface [GUI] desktop users). So suppose you'd like all your colleagues to read your important memo about a new policy governing billing pizza deliveries to the company credit card. You could create a text file and cat it to the wall command:

```
cat pizza.txt | wall
```

With this, who needs Facebook?

So you've learned about the various run levels and about how they can be invoked from the command line. But how are they defined? As you've just seen, you control the way your computer will operate by setting its run level. But, as I hinted earlier, there's more than one way to do that.

Years ago, run levels were controlled by a daemon (that is, a background process) called init (also known as SysVinit). A computer's default run level was stored in a text file called inittab that lived in the /etc directory. The critical line in inittab might have looked like this:

```
id:3: initdefault
```

However, these days, if you go looking for the inittab file on your computer, the odds are that you won't find it. That's because, as computers with far greater resources became available, and as the demands of multitasking environments increased, more efficient ways of managing complex combinations of processes were needed. Back in 2006, the Upstart process manager was introduced for Ubuntu Linux and was later adopted by a number of other distributions, including Google's Chrome OS.

Under Upstart, the behavior of the computer under specific run levels is defined by files kept in directories under /etc with names like rc0.d, rc1.d, and rc2.d. The default run level in Upstart is set in the /etc/init/rc- sysinit.conf file. Its critical entry would use this syntax:

```
env DEFAULT_RUNLEVEL=3
```

Configuration files representing individual programs that are meant to load automatically under specified conditions are similarly kept in the /etc/init/ directory. Here's part of the ssh.conf file defining the startup and shutdown behavior of the Secure Shell network connectivity tool:

```
start on runlevel [2345]
stop on runlevel [!2345]
respawn
respawn limit 10 5
umask 022
```

Now that you've worked so hard to understand how both the init and Upstart systems worked, you can forget all about them. The Linux world has pretty much moved on to the systemd process manager. As of version 15.04, even Ubuntu no longer uses Upstart.

Reprinted for IBM/renet@mx1.ibm.com, IBM Apress, David Clinton (c) 2016, Copying Prohibited

Systemd focuses more on processes than run levels. Nevertheless, you can still set your default run level by linking the default.target file in the /etc/systemd/system/ directory to the appropriate file in /usr/lib/systemd/system/.

Here's the content of default.target from a typical Fedora installation:

```
# This file is part of systemd.
#
# systemd is free software; you can redistribute it and/or modify it
# under the terms of the GNU Lesser General Public License as published by
# the Free Software Foundation; either version 2.1 of the License, or
# (at your option) any later version.
[Unit]
Description=Graphical Interface
Documentation=man:systemd.special(7)
Requires=multi-user.target
After=multi-user.target
Conflicts=rescue.target
Wants=display-manager.service
AllowIsolate=yes
```

Notice the multi-user.target values, indicating that this machine will, by default, boot to a full multi-user session. Much like the /etc/init/ directory in Upstart, /usr/lib/systemd/ contains configuration files for installed packages on systemd systems.

In fact, systemd is much more than just a simple process manager: it also includes a nice bundle of useful tools. For instance, running

```
systemctl list-units
```

will display all the currently available units and their status. A unit, by the way, is a resource that can include services, devices, and mounts. If you want to prepare, say, the Apache web server service—called httpd in Fedora—you would use systemctl and enable:

```
systemctl enable httpd.service
```

To actually start the service, you use:

```
systemctl start httpd.service
```

Pseudo Filesystems

In Linux, a filesystem is a way to organize resources—mostly files of one sort or another—in a way that makes them accessible to users or system resources. In a later chapter, I'll discuss the structure of a number of particularly common Linux filesystems (like ext3, ext4, and reiserFS) and how they can enhance security and reliability. For now, though, let's look at a specific class: the pseudo filesystem.

Since the word pseudo means fake, it's reasonable to conclude that a pseudo filesystem is made up of files that don't actually exist. Instead, the objects within such a structure simply *represent* real resources and their attributes. Pseudo filesystems are generated dynamically when your computer boots.

The /dev directory contains files representing hardware devices—both real and virtual. That's why, as you saw earlier in this chapter, a /dev address (/dev/sdb1) is used to identify and mount a hard drive. As you've also seen, lsblk displays all recognized physical block drives. Running

```
lsblk -a
```

however, will also show you *all* the block devices currently represented in /dev (even virtual devices).

The contents of the /sys directory represent the sysfs system and contain links to devices. The /sys/class/block directory, therefore, would include links to block devices, while the /sys/class/printer directory would contain links to printers.

The files within the /proc directory contain runtime system information. That is to say, a call to files within this hierarchy will return information about a system resource or process. Applying cat to the cpuinfo file, for instance,

```
cat /proc/cpuinfo
```

will return a technical description of your computer's CPU. Note however that poking the cpuinfo file with the "file" command reveals something interesting:

```
file /proc/cpuinfo cpuinfo: empty
```

It's empty!

You should spend some time exploring these directories. You might be surprised what you uncover.

You can quickly access subsets of the information held by these filesystems through a number of terminal commands: lspci will output data on all the PCI and PCI Express devices attached to your system. Adding the -xvvv argument:

```
lspci -xvvv
```

will display more verbose information; lsusb will give you similar information for USB devices; and lshw (list hardware) will—especially when

Reprinted for IBM/renet@mx1.ibm.com, IBM Apress, David Clinton (c) 2016, Copying Prohibited

run as the root—display information on your entire hardware profile.

Even though it doesn't contain pseudo files, I should also mention the /run directory hierarchy, since its contents are volatile, meaning that they are deleted each time you shut down or reboot your PC. So /run is therefore a great place for processes to save files that don't need to hang around indefinitely.

Device Management

Up to this point, you've seen how Linux learns enough about its hardware neighborhood to successfully boot itself, how it knows what kind of working environment to provide, and how it identifies and organizes hardware devices. Now let's find out how to manage these resources.

First, I should explain the role played by kernel modules in all this. Part of the genius of Linux is that its kernel—the software core that drives the whole thing—permits real-time manipulation of some of its functionality through modules. If you plug in a USB drive or printer, for instance, the odds are that Linux will recognize it and make it instantly available to you. This might seem obvious, but getting it right in a complicated world with thousands of devices in use is no simple thing.

Hotplug devices—like USB drives and cameras—can be safely added to a computer while it's actually running (or "hot"). Invoking udev, using communication provided by the D-Bus system, should recognize the device and automatically load a kernel module to manage it (see Figure 1-5).

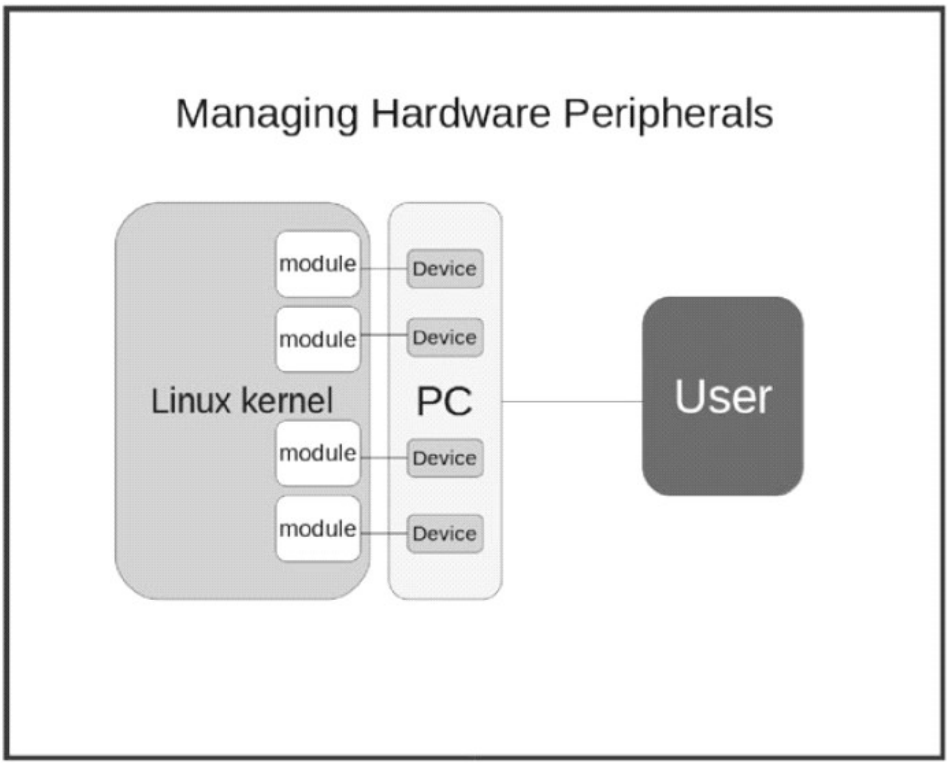

Figure 1-5: Linux kernel modules interpreting device activity for the Linux kernel

By and large, if you've got to open your computer's case to add a device, it's going to be of the coldplug variety: meaning, you shouldn't try to insert your device with the computer running. While I'm on that topic, it can't hurt to remind you that you should never touch exposed circuit boards without fully grounding yourself first. I've seen very expensive devices destroyed by static charges too small to be felt by humans.

Either way, once your device is happily plugged in, the appropriate kernel module should do its job connecting what needs connecting. But there will be times when you'll need to control modules yourself. To define device naming and behavior, you can edit its udev rules.d file. If there isn't already a .rules file specific to your device, you can create one in any one of these directories:

```
/etc/udev/rules.d/
/run/udev/rules.d/
/usr/lib/udev/rules.d/
```

If there are overlapping .rules files in more than one of those locations, udev will read and execute the first one it finds using the above order.

Even if a kernel module is not actually loaded into memory, it might well be installed. You can list all currently installed modules using this rather complex application of the find tool:

```
find /lib/modules/$(uname -r) -type f -iname "*.ko"
```

where *uname -r* will return the name of the kernel image that's currently running (to point "find" to the correct directory), the object type is *"file"* and the file extension is *.ko*.

Reprinted for IBM/renet@mx1.ibm.com, IBM
Apress, David Clinton (c) 2016, Copying Prohibited

Running lsmod will list only those modules that are actually loaded. To load an installed module, you can use modprobe:

```
sudo modprobe lp
```

which will load the printer driver, while:

```
sudo modprobe -r lp
```

will remove the module.

Don't think that manually managing kernel modules is something only veteran administrators and developers need to do. In just the past month, I've had to get my hands dirty with this task not once, but twice, and to solve problems on simple PCs, not rack-mounted servers!

The first time occurred when I logged into a laptop and noticed that there was no Wi-Fi. The usual troubleshooting got me nowhere, so I used lshw:

```
sudo lshw -C network
```

to see what the system had to say about the Wi-Fi interface. The phrase "network UNCLAIMED" showed up next to the entry for the adapter. Because it wasn't "claimed," the adapter had never been assigned an interface name (like wlan0) and it was, of course, unusable. I now suspect that the module was somehow knocked out by a recent software update.

The solution was simple. With some help from a quick Google search built around the name of this particular Wi-Fi model, I realized that I would have to manually add the ath9k module. I did that using:

```
sudo modprobe ath9k
```

and it's been living happily every after.

The second surprise happened when I couldn't get a browser-based web conferencing tool to recognize my webcam. Again, all the usual tricks produced nothing, but Internet searches revealed that I wasn't the first user to experience this kind of problem. Something was causing the video camera module to crash, and I needed a quick way to get it back on its feet again without having to reboot my computer. I first needed to unload the existing module:

```
sudo rmmod uvcvideo
```

Then it was simply a matter of loading it again, and we were off to the races:

```
sudo modprobe uvcvideo
```

Now Try This

Let's imagine that you recently added a PCI Express network interface card (NIC) to your system. Because it's new, udev assigned it the name em1 rather than em0 (the name used by your existing integrated NIC). The problem is that you've hard coded em0 into various scripts and programs, so they all expect to find a working interface with that name. But as you want to connect your network cable to the new interface, em0 will no longer work. Since you're far too lazy to update all your scripts, how can you edit a file in the /etc/udev/rules.d/ directory to give your new NIC the name em0?

> **Note** I would strongly advise you to create a backup copy of any file you plan to edit, and then make sure you restore your original settings once you're done!

Test Yourself

1. Pressing Ctrl+c in the GRUB menu will: ?

a. Allow you to edit a particular image

b. Open a command line session

c. Initiate a memory test

d. Launch a session in recovery mode

2. Adding rw init=/bin/bash to your boot parameters in GRUB will: ?

a. Allow root access on booting

b. Launch a session in recovery mode

c. Display the most recent contents of the /var/log/dmesg file

d. Allow logged messages to be edited

3. sudo is: ?

a. Another name for the Linux root user

b. The command that mounts devices in the root directory

c. The most direct tool for changing system run levels

Reprinted for IBM/renet@mx1.ibm.com, IBM

Apress, David Clinton (c) 2016, Copying Prohibited

d. A system group whose members can access admin permissions

4. On most Linux systems, run level 1 invokes: ?

a. Single user mode

b. X11 (graphic mode)

c. Reboot

d. Full multi-user mode

5. On Linux systems running systemd, the default run level can be found in: ?

a. /etc/systemd/system/inittab

b. /lib/systemd/system/default.target

c. /etc/systemd/system/default.target

d. /etc/init/rc-sysinit.conf

6. You can find links to physical devices in: ?

a. /dev

b. /etc/dev

c. /sys/lib

d. /proc

7. Which is the quickest way to display details on your network device? ?

a. lsblk

b. lspci

c. cat /proc/cpuinfo

d. lshw

8. Which tools are used to watch for new plug-in devices? ?

a. udev and modprobe

b. rmmod and udev

c. modprobe, uname, and D-Bus

d. udev and D-Bus

9. The correct order udev will use to read rules files is: ?

a. /etc/udev/rules.d/ /usr/lib/udev/rules.d/ /run/udev/rules.d/

b. /usr/lib/udev/rules.d/ /run/udev/rules.d/ /etc/udev/rules.d/

c. /etc/udev/rules.d/ /run/udev/rules.d/ /usr/lib/udev/rules.d/

d. /etc/udev/rules.d/ /run/udev/rules.d/

10. You can load a kernel module called lp using: ?

a. sudo modprobe lp

b. sudo modprobe load lp

c. sudo modprobe -l lp

d. sudo rmmod lp

Answers

1. b

2. a

3. d

4. a

Reprinted for IBM/renet@mx1.ibm.com, IBM

Apress, David Clinton (c) 2016, Copying Prohibited

5. c
6. a
7. b
8. d
9. c
10. a

Reprinted for IBM/renet@mx1.ibm.com, IBM
Apress, David Clinton (c) 2016, Copying Prohibited