TEACH YOURSELF

LINUX ADMINISTRATION

and prepare for the LPIC-1 certification exams.

Complete. Quick.

David Clinton

Bootstrap IT

www.bootstrap-it.com

Copyright © 2016 David Clinton and Bootstrap IT.
All rights reserved.
No part of this publication may be reproduced or distributed in any form
or by any means, or stored in a database or retrieval system,
without the prior written permission of the publisher.

ISBN: 978-1-329-80370-1

info@bootstrap-it.com
www.bootstrap-it.com

Table of Contents

INTRODUCTION

Introduction to the book

About Linux

About the LPIC-1 exam

Linux survival skills

About the book you hold in your hands

First of all, welcome.

Whether you're reading this book because you've decided to earn the Linux Professional Institute's Server Professional certification or because you simply want to learn more about Linux administration, you've made a great choice. Right now, for a thousand reasons, Linux administration skills are opening doors to some of the hottest job markets on earth. And with the ongoing explosive growth of the cloud computing world - the vast majority of which being built with Linux - the opportunities will only get richer.

Now, about this book. I chose to have the chapters closely follow the LPIC exam topics. Not only will this make it much easier for you to study for each of the two exams required for LPIC-1 certification, but I believe that the exam objectives are actually nicely aligned with the tools you'll need in the real world. Whether or not you end up taking the exam, if you manage to learn this material, you'll have done yourself a real favor.

By far the most important element of your success, however, will have very little to do with this or any other book. No matter how much time you spend studying a book, very little of the information you read will magically translate into knowledge and skills, unless you put it to work.

If you want to really "get" this stuff, you'll have to roll up your sleeves, open up a terminal, and *do it*. As soon as you finish a chapter or a section, try out what you've learned on a real living, breathing Linux system. Even better, take on your own projects. Be ambitious. Be adventurous. Take (managed) risks.

To this end, I include suggestions for practical exercises at the end of each chapter (right before the "test yourself" quizzes). Be prepared to spend longer than you expected on some of those tasks...sometimes longer than it took you to read the chapters they're based on. Also, accept that you will probably make some mistakes that will require even more time to fix. This is all as it should be. Remember: you learn more from experience than anything else.

You will notice that I used the words "complete" and "quick" to describe this book. Let me explain what I meant. The book is complete in the sense that every concept, principle, process, and resource that might make an appearance on the exam is fully represented (even a few that are now quite obsolete and/or useless: I'm looking at you, X font server).

However, your journey through this book may also be relatively quick, since I've tried to be as selective as possible about what I include. As you will see soon enough, I didn't even try to include every single option for every single utility, which would have been highly impractical. But it would also have been largely useless, because I don't believe any normal human being could possibly absorb page after page after page of that kind of dry, abstract information.

If you want to see the full, formal documentation for a particular Linux utility, simply consult the man pages that came pre-installed with your Linux distribution. As an example, from the command line, you can type:

```
man cp
```

Besides including only the more common command options, I also tried to avoid talking about more general IT issues that don't relate directly to the LPIC exam. It's not that they're not important, but I figured that they may only interest a relatively small number of my readers and, importantly, they're all easily accessible on the Internet.

I'd like to introduce you to one of my best friends: the Internet search engine.

So if you're curious about something that isn't discussed in these pages or if a project you're working on needs greater detail, then by all means, dive in deep. But because I know that the Internet has answers to just about any question you're likely to have, I'm able to focus this book more narrowly on the curriculum that interests everyone.

Having said that, please visit our website, bootstrap-it.com. We'll try to make your visit worthwhile and, more importantly, provide you all with the opportunity to talk to us...and to each other. Let us know how you're doing and what you think.

About Linux

There's so much we could say about Linux:

- It's the operating system used by more than 95% of the world's supercomputers.
- Google, Netflix, and Facebook? Linux, Linux, and Linux.
- The vast majority of virtual machines fired up on the leading cloud computing platforms (like Amazon's AWS) are running Linux - and that includes Microsoft's Azure!
- There's a very good chance that the software powering your car, TV, smart phone, air traffic control system, and even neighborhood traffic lights, is one flavor or another of Linux.

If there's innovation in the worlds of science, finance, communications, entertainment, and connectivity, it's almost certainly being driven by Linux. And if there are dozens of attractive, virus-free, secure, and reliable desktop and mobile operating systems freely available to fill all kinds of roles, those too are driven by Linux.

By they way, you may be interested to know that this book was produced in its entirety on Linux, using only open source software. The whole thing: research, testing, image processing, and typography.

The Linux Foundation recently (September, 2015) estimated that, over just the past few years, collaborative projects under their umbrella have produced an estimated five billion dollars in economic value. This was, again according to the Foundation, "work that would take 1,356 developers more than 30 years to replicate."

But where did all this innovation, productivity, and value come from? Who actually makes it all happen? It seems that the little operating system built a couple of decades ago by Linus Torvalds and then

donated to the world, is maintained by thousands of developers working for free. According to the Linux Foundation, through 2015, 7.71 changes were accepted into the Linux kernel each HOUR and those contributions were the work of, besides Torvalds himself, more than 4,000 developers scattered around the world - many of whom, it must be noted, sponsored by the companies they work for.

That's the *power* of open source. "Open source?" I hear you ask. "But who will support us when things go wrong?"

That's the *beauty* of open source. Because when I can't figure out how to do something or when I discover a bug in some open source software, I can usually quickly find the answer through an Internet search or, if not, there are knowledgeable and helpful folk online just waiting to help me. Try it out. You might, as I have from time to time, quickly find yourself in direct contact with the project developers themselves.

Some years ago, I wrote a white paper arguing the business case for transitioning small and medium sized businesses from proprietary office productivity software suites (Microsoft Office) to open source alternatives (LibreOffice). When I compared the response/resolution times delivered by Microsoft with the average times seen on volunteer-staffed online OpenOffice and LibreOffice help forums, the latter would consistently produce a quicker turnaround.

Now it's your turn. All that innovation is going to need administrators to apply it to the real world. After all, we system administrators know just how little developers would get done without us. As the IT world grows and changes, you will be on the cutting edge.

Or will you? Let me tell you a story about an old friend of mine who, 25 years ago, had a great job as a Unix admin. As he tells it, the problem was that Unix (which, for the purpose of this discussion, is effectively synonymous with Linux) was getting so good at automating processes and system audits that all kinds of mid-level admins simply became unnecessary. My friend lost his job.

Could this happen to you? Absolutely. Unless, that is, you make an effort to keep up with technology as it evolves. There will be new areas to keep your eyes on (embedded tech, container virtualization, and others not yet imagined). It's the 21st Century: you're never finished learning.

Nevertheless, I predict that 95% of the the basic Linux skills you will learn here will probably still be in use ten and even twenty years from now. This is solid, foundational material.

About the LPIC-1 exam

The two exams you'll need to pass to earn your Server Professional certification (LPIC-1 101 and 102) are also known as CompTIA Linux+ LX0-103 and LX0-104. Until a few years ago, CompTIA offered a Linux certification that was so similar to the LPIC that the two eventually merged. All you have to know is that, whatever they're called, they work the same way and will get you to the same place.

That is not true of LPI's Linux Essentials (LPI-010) exam, which is a single, introductory exam that's meant for individuals with far less experience and knowledge than a candidate for the Server Professional would have. Besides those, the LPIC offers two other sets of exams designed to demonstrate added skills and experience beyond those of the LPIC-1: the LPIC-2 (Linux Network Professional Certification), and LPIC-3 (Mixed Environments).

This book is based on the April, 2015 edition of the exams (Version 4.0). The people who maintain the certification and exams are, by design, very conservative in the way they adopt major changes, so you can be confident that the key exam topics won't be changing dramatically any time soon.

The Linux Professional Institute is vendor neutral, meaning that no one mainstream Linux distribution or software stack is favored over any other. You will therefore need to become familiar with a range of technologies. So, for example, expect to see both the Systemd and Upstart process managers, or both the apt and yum package managers. And that's a really good thing, because all of those systems are widely used (for now, at least) and all have unique valuable features. You can only gain from understanding how they all work. Success with the LPIC-1 will also automatically earn you the SUSE CLA certification.

Each exam is made up of 60 multiple choice and fill in the blank questions which must be completed within 90 minutes. To pass an exam, you will need to score 500 marks out of a total of 800. Since the questions are weighted by topic, there is no guarantee that one question will be worth the same number of marks as another. You can book an exam through the website of either the Pearson VUE or Prometric test administration companies.

As with most technical certification exams, you will need to present the exam provider with two forms of identification, one of them a government issued photo ID. You will also be expected to surrender any electronic devices or notebooks. (If you're very nice to the proctors, they might give them back to you once you're done.)

More than most certifications, the LPIC has done a great job communicating exactly what you will need to know. You should spend some time carefully reading through the two exam objectives pages from their web site (lpi.org/study-resources/lpic-1-101-exam-objectives and lpi.org/study-resources/lpic-1-102-exam-objectives) and then go through them again at the end of the process to make sure you haven't missed anything. For your convenience, I've included the objectives in an appendix at the end of this book.

You will notice that each topic is given a weight between one and five. Those indicate the relative importance of a topic in terms of how large a role it will play in the exam. Here's a simple chart that adds up the weights by topic to illustrate the importance of each:

Topic	**Weight**	
101	8	System Architecture
102	11	Linux Installation and Package Management
103	26	GNU and Unix Commands
104	15	Devices, Filesystems, Filesystem Hierarchy Standard
Total:	**60**	
105	10	Shells, Scripting and Data Management
106	4	User Interfaces and Desktops
107	12	Administrative Tasks
108	11	Essential System Services
109	14	Networking Fundamentals
110	9	Security
Total:	**60**	

Exam tips

Try to arrive at the exam center as relaxed and well rested as possible. Carefully and slowly read each question and each possible response. Look for important details and for details that are only there to distract you. If you're not absolutely sure which answer is correct, try to narrow down the field a bit by eliminating answers that are obviously incorrect. You can always skip hard questions and return to them later when you've completed the rest.

Finally, remember that more people fail this exam on their first try than pass: it's designed to inspire your best effort. So don't give up.

Linux Survival Skills

Why the single section...isn't this whole book about Linux survival skills? Well yes, but how are you going to survive between now and the time you finish reading it? Just to get you started, it might be useful to pick up a few super-critical, can't-live-without-me tools.

First of all, nearly everything in Linux administration will happen through the terminal. But I know that at least some of you are sitting in front of a shiny new Linux GUI interface right now and wondering where the #$%@! the terminal is (if you'll excuse my language). The answer is: that depends. Ubuntu, for instance, changes their menu design with just about every distribution, so exactly where Terminal will appear on your desktop is hard to predict. In some ways, things just got more complicated with some more recent desktop manager versions, which got rid of menus altogether.

If you're not interested in poking around looking for it, you can try hitting the ALT+f2 combination and then typing "terminal" (or "gnome-terminal") into the dialog box. Or, on some systems, CTRL+ALT+t will get you there directly.

Once you're in the terminal, try running a command. Type:

```
pwd
```

...which stands for "present work directory". This is the folder (something that's almost always called a "directory" in Linuxland, by the way) you're currently in. You can list the files and subdirectories in your current directory with ls:

```
ls -l
```

Adding the -l argument gives you a longer, more detailed list displaying file attributes. If it's already installed (and it usually will be), you can use the nano text editor to, in this case, create and edit a new text file:

```
nano myfile.txt
```

Go ahead and type a few words and then hit CTRL+x to save and exit. You can now quickly view your literary creation using cat:

```
cat myfile.txt
```

Try that again, but this time, type only "cat my" without the rest of the filename. Instead, hit the tab key and Linux Command Completion should figure out what you're after and finish the command for you. Just hit enter to accept the suggestion. Trust me: this one can save you a great many keystrokes and a whole lot of time over the coming years.

Let's create a new directory:

```
mkdir newplace
```

...and "change directory" into newplace and then run pwd once again:

```
cd newplace
pwd
```

Perhaps we'd like to copy the file we just created into this directory. To do this, we'll need to keep in mind where our personal "home" directory exists in the larger Linux filesystem. We'll assume that our account is called bootstrap-it, which is therefore the name of our home directory.

```
cp /home/bootstrap-it/myfile.txt .
```

"/home/bootstrap-it/myfile.txt" identifies the file we want to copy, and the dot (.) tells the cp command to copy it to the current directory. Run ls to confirm that a copy has arrived:

```
ls
```

You can change a file's name or move it using mv:

```
mv myfile.txt mynewfile.txt
ls
```

And you can permanently delete the file using rm:

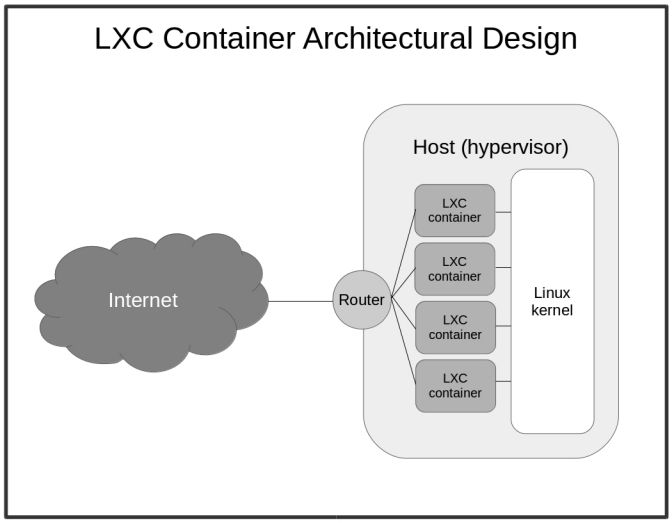
```
rm mynewfile.txt
```

To help you experiment with Linux skills without having to worry about making a mess of your important stuff, you might try working with disposable systems. One way to do that is by loading a Linux image on to a USB stick, and then booting your computer to a live Linux session. Unless you mount and play around with your existing hard drive, nothing you do will have any permanent impact on your "real" data or system settings, and nothing you do to the live filesystem will survive a reboot. This has the added potential advantage of exposing you to a

wide range of Linux distributions beyond the one that you've chosen for your main work.

LXC containers

You can also create virtual machines within a working installation using LXC. An LXC container (as its called) is a fully functioning, persistent virtual "machine" that likes to imagine that it lives all by itself on your hardware. You can play around in this sandbox-like environment to your heart's content and, when you break something (as you probably will), you can just destroy it and start again with a new one. I highly recommend using LXCs for exploration and experimentation. I use them myself all the time and they've saved me untold hours of heartache.



Here are the simple steps you'll need to get started with LXC (none of this is included among the LPIC-1 exam expectations). This assumes that you're using an Ubuntu machine - some commands may be a bit different for other distributions. First of all, make sure that openssh is installed on your host machine (we'll talk a lot more about what that is later in the book):

```
sudo apt-get update
sudo apt-get install openssh-server
```

Now install lxc:

```
sudo apt-get install lxc
```

We will create a new container called newcon using the ubuntu template:

```
sudo lxc-create -t ubuntu -n newcon
```

Once that's done (and it should only take a minute or two), we'll boot the new container:

```
sudo lxc-start -d -n newcon
```

-d tells lxc to detach from the container, to allow it to survive our exit from the shell. Now we'll list all the existing containers (it might take a short while before newcon is listed as fully up):

```
sudo lxc-ls --fancy
```

Assuming that the IP address for newcon (listed by our previous command) is 10.0.3.120, we'll ssh into the container:

```
ssh ubuntu@10.0.3.120
```

And voila! A brand new computer playground, waiting for us to come and play! Now you've got no excuses: get to work.

TOPIC 101: SYSTEM ARCHITECTURE

The Linux Boot Process

Run Levels

Pseudo Filesystems

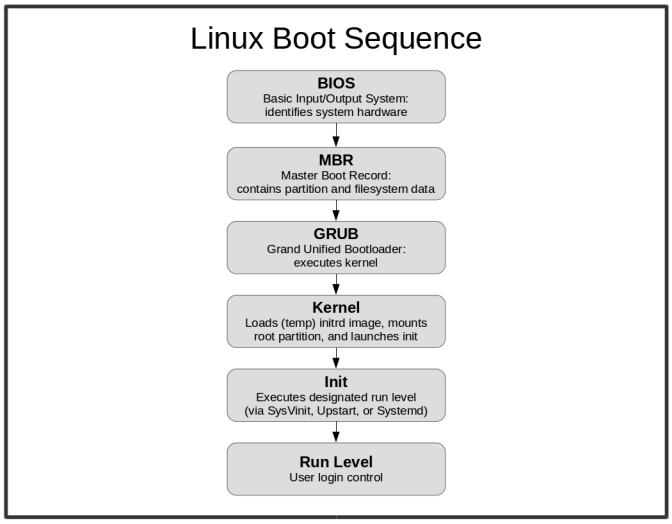
Device Management

The Linux Boot Process

Unless you end up working exclusively with virtual machines or on a cloud platform like Amazon's AWS, 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 LPIC exam curriculum, we won't focus on them in this book. Instead, we'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 certification 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).

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.



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 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.

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,*

---

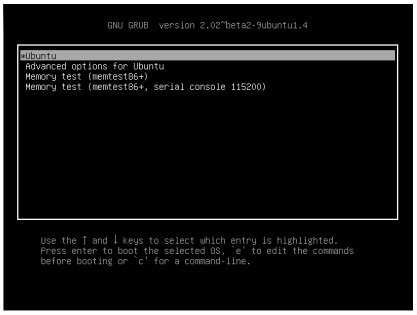
*which is not to be confused with GRUB **stage** 1, 1.5, or 2! The GRUB we're all using today is known as GRUB version 2.*

You think that's confusing? Just be grateful that they don't still expect you to know about the LILO bootloader!

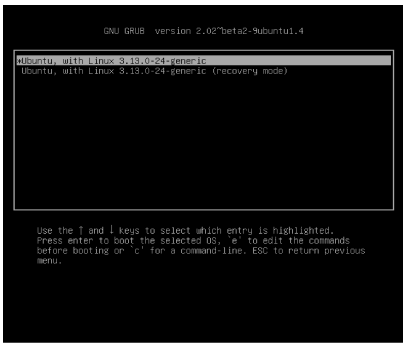
Besides orchestrating the boot process, GRUB will also present you with a startup menu from which you can control the software your system will load.

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.

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.



The Advanced menu allows you to run in recovery mode or, if there happened 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.



Pressing "e" with a particular image highlighted will let you edit its boot parameters. 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.

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

You may be interested - or perhaps horrified - to know that adding rw init=/bin/bash to your boot parameters will open a full root 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 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. (We're going to 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 we 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 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, we're 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. We'll get back to that in just a minute or two.

But let's start at the beginning. One of the 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 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.

```
sudo shutdown -h 5
```

would shut down the system, but only after five minutes.

```
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 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 we've learned about the various run levels and about how they can be invoked from the command line. But how are they defined? As we've just seen, you control the way your computer will operate by setting its run level. But, as we 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 start up and shut down 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.

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 that 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 start:

```
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, we'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, we're going to 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, we used a /dev address (/dev/sdb1) to identify and mount a hard drive. As we'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. lshw ("list hardware") will - especially when run as the root - display information on your entire hardware profile.

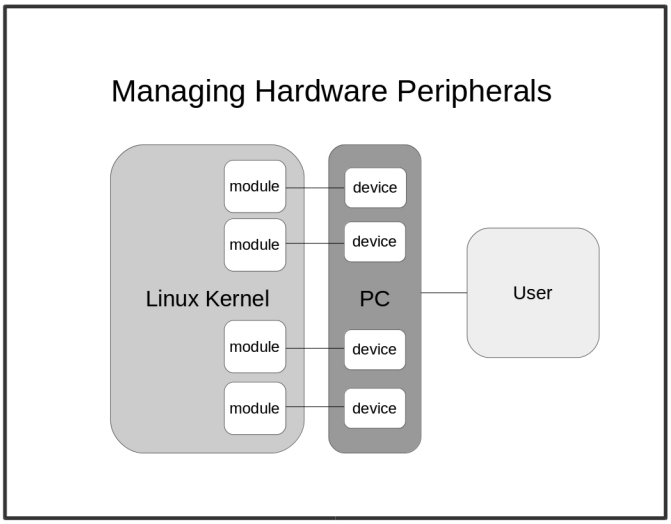
Even though it doesn't contain pseudo files, I should also mention the /run directory hierarchy, since its contents are volatile, meaning that they are deleted each time you shut down or reboot your PC. /run is therefore a great place for processes to save files that don't need to hang around indefinitely.

Device Management

Up to this point, we'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 we'll need to find out how we can manage these resources.

First of all, 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"). udev, using communication provided by the D-Bus system, should recognize the device and automatically load a kernel module to manage it.



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 we're 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 us.

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*.

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

```
sudo modprobe lp
```

...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 WiFi. The usual troubleshooting got me nowhere, so I used lshw:

```
sudo lshw -C network
```

...to see what the system had to say about my WiFi interface. The phrase "network UNCLAIMED" showed up next to the entry for my 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 my particular WiFi model, I realized that I would have to manually add the ath9k module. I did that using:

```
sudo modprobe ath9k
```

...and we've 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 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
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

Answer key:

1 b, 2 a , 3 d , 4 a , 5 c , 6 a , 7 b , 8 d , 9 c , 10 a