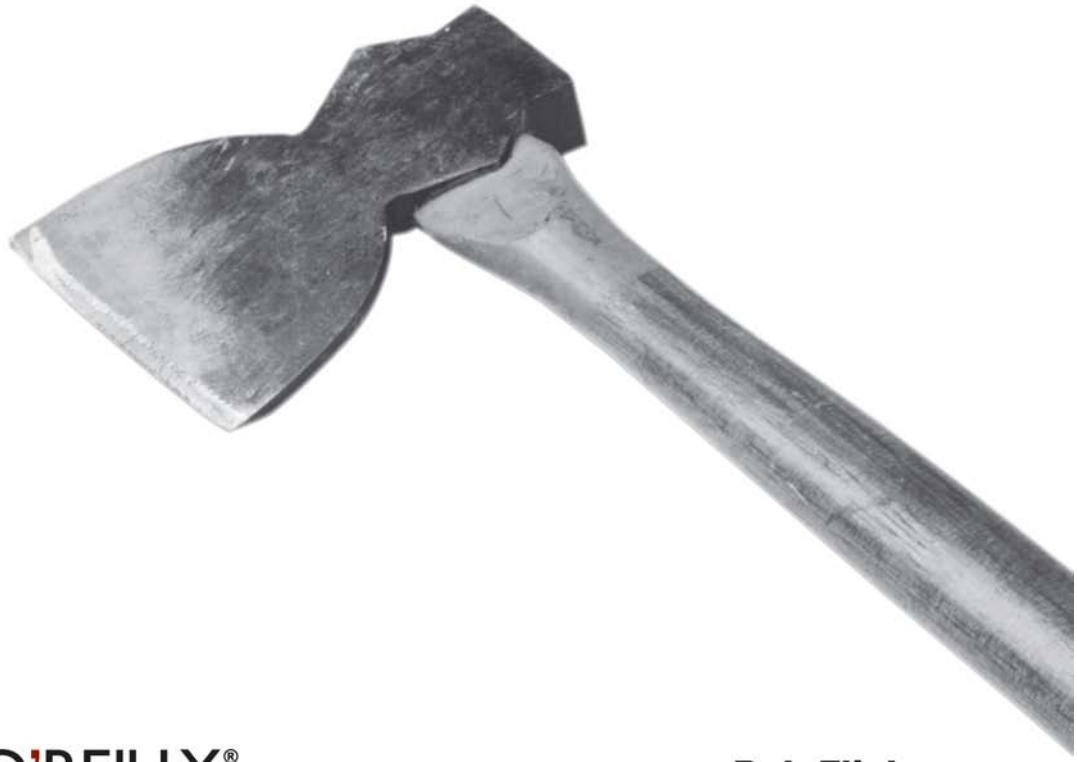
LINUX SERVER HACKS™

100 Industrial-Strength Tips & Tools



O'REILLY®

Rob Flickenger

LINUX SERVER HACKS 100 Industrial-Strength Tips & Tools

A competent system administrator knows that a Linux server is a high performance system for routing large amounts of information through a network connection. Setting up and maintaining a Linux server requires understanding the ins and outs of the Linux operating system and its supporting cast of utilities as well as many layers of applications software. You'll find basic documentation online but there's a lot beyond the basics that you have to know, and this only comes from people with hands-on, real-world experience. This kind of "know how" is what we capture in *Linux Server Hacks*.

Linux Server Hacks is a collection of 100 industrial-strength hacks, providing tips and tools that solve practical problems for Linux system administrators. Every hack can be read in just a few minutes but will save hours of searching for the right answer.

Some of the hacks are subtle, many of them are non-obvious, and all of them demonstrate the power and flexibility of a Linux system. You'll find hacks devoted to tuning the Linux kernel to make your system run more efficiently, using CVS or RCS to track revisions to system files, and using monitoring tools to track system performance. *Linux Server Hacks* also helps you manage large-scale Web installations running Apache, MySQL, and other open source tools that are typically part of a Linux system.

Written for users who already understand the basics, *Linux Server Hacks* is built upon the expertise of people who really know what they're doing.

Hacking is "an appropriate application of ingenuity… Whether the result is a quick-and-dirty patchwork job or a carefully crafted work of art, you have to admire the cleverness that went into it."

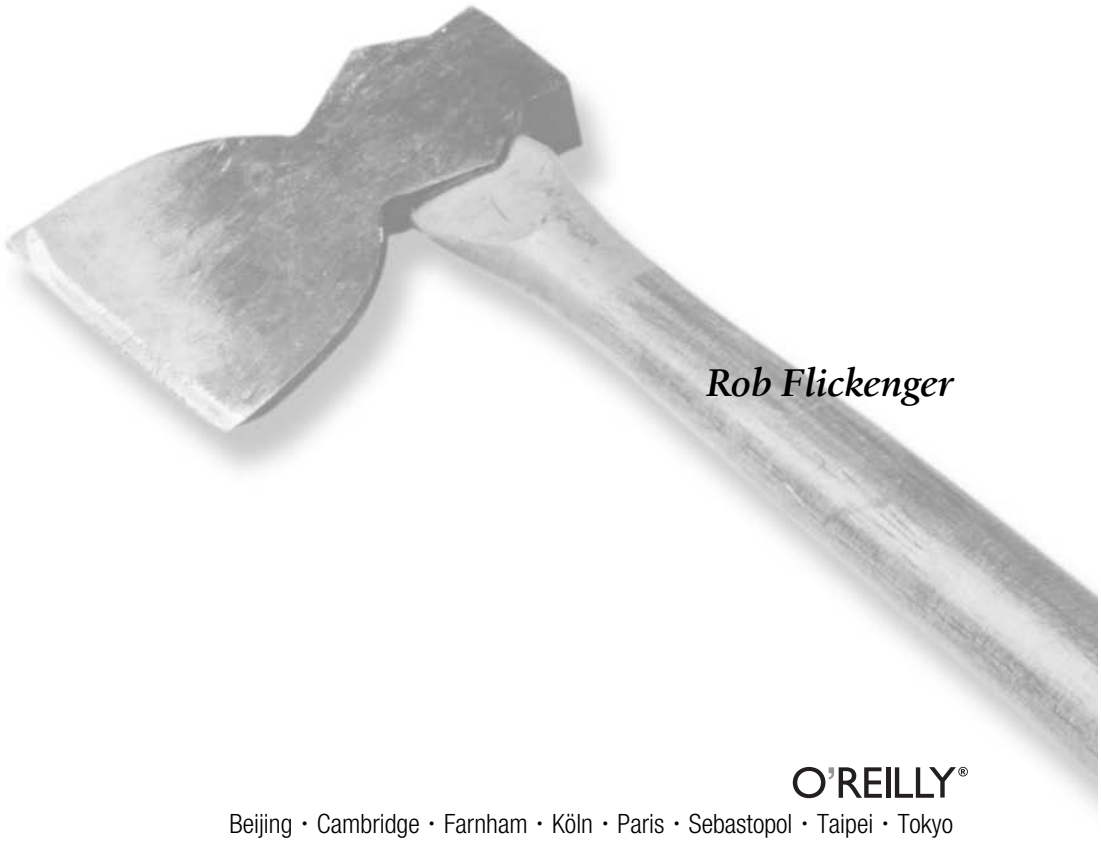
— Eric S. Raymond
New Hacker's Dictionary

Got a Hack? Go to: *hacks.oreilly.com*

US $24.95 **CAN $38.95**
ISBN: 978-0-596-00461-3

O'REILLY®
www.oreilly.com

LINUX SERVER HACKS



Rob Flickenger

O'REILLY®

Beijing · Cambridge · Farnham · Köln · Paris · Sebastopol · Taipei · Tokyo

Linux Server Hacks
by Rob Flickenger

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

Editor:	Dale Dougherty	**Production Editor:**	Sarah Sherman
Series Editor:	Rael Dornfest	**Cover Designer:**	Edie Freedman
Executive Editor:	Dale Dougherty	**Interior Designer:**	David Futato

Printing History:

January 2003: First Edition.

ISBN13: 978-0-596-00461-3

This book uses RepKover™, a durable and flexible lay-flat binding.

[C] [4/08]

Contents

Credits

About the Author

Rob Flickenger authored the majority of hacks in this book. Rob has worked with Linux since Slackware 3.5. He was previously the system administrator of the O'Reilly Network (an all-Linux shop, naturally) and is the author of *Building Wireless Community Networks*, also by O'Reilly.

Contributors

The following people contributed their hacks, writing, and inspiration to this book:

- Rael Dornfest ("Apache Toolbox" **[Hack #75]**) is a maven at the O'Reilly & Associates focusing on technologies just beyond the pale. He assesses, experiments, programs, and writes for the O'Reilly network and O'Reilly publications.
- Schuyler Erle (contributed code for httptop, mysql-table-restore, balance-push, find-whois, and vtundgen) is, by day, a mild-mannered Internet systems developer for O'Reilly & Associates. By night, he crusades for justice and freedom as a free software hacker and community networking activist.
- Kevin Hemenway ("Quick Configuration Changes with IfDefine" **[Hack #75]**, "Simplistic Ad Referral Tracking" **[Hack #75]**, "Mimicking FTP Servers with Apache" **[Hack #75]**), better known as Morbus Iff, is the creator of disobey.com, which bills itself as "content for the discontented." Publisher and developer of more home cooking than you could ever imagine, he'd love to give you a Fry Pan of Intellect upside the head. Politely, of course. And with love.

- Seann Herdejurgen ("iptables Tips & Tricks" **[Hack #44]**, "Disk Age Analysis" **[Hack #53]**) has been working with Unix since 1987 and now architects high availability solutions as a senior systems engineer with D-Tech corporation in Dallas, Texas. He holds a master's degree in computer science from Texas A&M University. He may be reached at: *http://seann.herdejurgen.com/*.
- Dru Lavigne ("Archiving with Pax" **[Hack #36]**) is an instructor at a private technical college in Kingston, ON where she teaches the fundamentals of TCP/IP networking, routing, and security. Her current hobbies include matching every TCP and UDP port number to its associated application(s) and reading her way through all of the RFCs.
- Cricket Liu ("Views in BIND 9" **[Hack #75]**) matriculated at the University of California's Berkeley campus, that great bastion of free speech, unencumbered Unix, and cheap pizza. He worked for a year as Director of DNS Product Management for VeriSign Global Registry Services, and is a co-author of *DNS and BIND*, also published by O'Reilly & Associates.
- Mike Rubel ("Automated Snapshot-Style Incremental Backups with rsync" **[Hack #36]**, *http://www.mikerubel.org*) studied mechanical engineering at Rutgers University (B.S. 1998) and aeronautics at Caltech (M.S. 1999), where he is now a graduate student. He has enjoyed using Linux and GNU software for several years in the course of his numerical methods research.
- Jennifer Vesperman (all of the CVS pieces except "CVS: Anonymous Repositories" **[Hack #22]** were adapted from her online CVS pieces for O'ReillyNet) contributes to open source as a user, writer, and occasional programmer. Her coding experience ranges from the hardware interface to an HDLC card to the human interface of Java GUIs. Jenn is the current coordinator and co-sysadmin for Linuxchix.org.

Acknowledgments

I would like to thank my family and friends for their support and encouragement. Thanks especially to my dad for showing me "proper troubleshooting technique" at such an early age and inevitably setting me on the path of the Hacker before I had even seen a computer.

Of course, this book would be nothing without the excellent contributions of all the phenomenally talented hackers contained herein. But of course, our hacks are built by hacking the shoulders of giants (to horribly mix a metaphor), and it is my sincere hope that you will in turn take what you learn here and go one better, and most importantly, tell everyone just how you did it.

How to Become a Hacker

The Jargon File contains a bunch of definitions of the term "hacker," most having to do with technical adeptness and a delight in solving problems and overcoming limits. If you want to know how to *become* a hacker, though, only two are really relevant.

There is a community, a shared culture, of expert programmers and networking wizards that traces its history back through decades to the first time-sharing minicomputers and the earliest ARPAnet experiments. The members of this culture originated the term "hacker." Hackers built the Internet. Hackers made the Unix operating system what it is today. Hackers run Usenet. Hackers make the Web work. If you are part of this culture, if you have contributed to it and other people in it know who you are and call you a hacker, you're a hacker.

The hacker mind-set is not confined to this software-hacker culture. There are people who apply the hacker attitude to other things, like electronics or music—actually, you can find it at the highest levels of any science or art. Software hackers recognize these kindred spirits elsewhere and may call them "hackers" too—and some claim that the hacker nature is really independent of the particular medium the hacker works in. But in the rest of this document, we will focus on the skills and attitudes of software hackers, and the traditions of the shared culture that originated the term "hacker."

There is another group of people who loudly call themselves hackers, but aren't. These are people (mainly adolescent males) who get a kick out of breaking into computers and breaking the phone system. Real hackers call these people "crackers" and want nothing to do with them. Real hackers mostly think crackers are lazy, irresponsible, and not very bright—being able to break security doesn't make you a hacker any more than being able to hotwire cars makes you an automotive engineer. Unfortunately, many journalists and writers have been fooled into using the word "hacker" to describe crackers; this irritates real hackers no end.

The basic difference is this: hackers build things, crackers break them.

If you want to be a hacker, keep reading. If you want to be a cracker, go read the alt.2600 newsgroup and get ready to do five to ten in the slammer after finding out you aren't as smart as you think you are. And that's all I'm going to say about crackers.

The Hacker Attitude

Hackers solve problems and build things, and they believe in freedom and voluntary mutual help. To be accepted as a hacker, you have to behave as though you have this kind of attitude yourself. And to behave as though you have the attitude, you have to really believe the attitude.

But if you think of cultivating hacker attitudes as just a way to gain acceptance in the culture, you'll miss the point. Becoming the kind of person who believes these things is important for *you*—for helping you learn and keeping you motivated. As with all creative arts, the most effective way to become a master is to imitate the mind-set of masters—not just intellectually but emotionally as well.

Or, as the following modern Zen poem has it:

> To follow the path:
> look to the master,
> follow the master,
> walk with the master,
> see through the master,
> become the master.

So if you want to be a hacker, repeat the following things until you believe them.

1. The world is full of fascinating problems waiting to be solved.

Being a hacker is a lot of fun, but it's a kind of fun that takes a lot of effort. The effort takes motivation. Successful athletes get their motivation from a kind of physical delight in making their bodies perform, pushing themselves past their own physical limits. Similarly, to be a hacker you have to get a basic thrill from solving problems, sharpening your skills, and exercising your intelligence.

If you aren't the kind of person that feels this way naturally, you'll need to become one in order to make it as a hacker. Otherwise you'll find your hacking energy is zapped by distractions like sex, money, and social approval.

(You also have to develop a kind of faith in your own learning capacity—a belief that even though you may not know all of what you need to solve a problem, if you tackle just a piece of it and learn from that, you'll learn enough to solve the next piece—and so on, until you're done.)

2. No problem should ever have to be solved twice.

Creative brains are a valuable, limited resource. They shouldn't be wasted on re-inventing the wheel when there are so many fascinating new problems waiting out there.

To behave like a hacker, you have to believe that the thinking time of other hackers is precious—so much so that it's almost a moral duty for you to share information, solve problems, and then give the solutions away just so other hackers can solve *new* problems instead of having to perpetually re-address old ones.

(You don't have to believe that you're obligated to give *all* your creative product away, though the hackers that do are the ones that get most respect from other hackers. It's consistent with hacker values to sell enough of it to keep you in food and rent and computers. It's fine to use your hacking skills to support a family or even get rich, as long as you don't forget your loyalty to your art and your fellow hackers while doing it.)

3. Boredom and drudgery are evil.

Hackers (and creative people in general) should never be bored or have to drudge at stupid repetitive work, because when this happens it means they aren't doing what only they can do—solve new problems. This wastefulness hurts everybody. Therefore boredom and drudgery are not just unpleasant but actually evil.

To behave like a hacker, you have to believe this enough to want to automate away the boring bits as much as possible, not just for yourself but for everybody else (especially other hackers).

(There is one apparent exception to this. Hackers will sometimes do things that may seem repetitive or boring to an observer as a mind-clearing exercise, or in order to acquire a skill or have some particular kind of experience you can't have otherwise. But this is by choice—nobody who can think should ever be forced into a situation that bores them.)

4. Freedom is good.

Hackers are naturally anti-authoritarian. Anyone who can give you orders can stop you from solving whatever problem you're being fascinated

by—and, given the way authoritarian minds work, will generally find some appallingly stupid reason to do so. So the authoritarian attitude has to be fought wherever you find it, lest it smother you and other hackers.

(This isn't the same as fighting all authority. Children need to be guided and criminals restrained. A hacker may agree to accept some kinds of authority in order to get something he wants more than the time he spends following orders. But that's a limited, conscious bargain; the kind of personal surrender authoritarians want is not on offer.)

Authoritarians thrive on censorship and secrecy. And they distrust voluntary cooperation and information-sharing—they only like "cooperation" that they control. So to behave like a hacker, you have to develop an instinctive hostility to censorship, secrecy, and the use of force or deception to compel responsible adults. And you have to be willing to act on that belief.

5. Attitude is no substitute for competence.

To be a hacker, you have to develop some of these attitudes. But copping an attitude alone won't make you a hacker, any more than it will make you a champion athlete or a rock star. Becoming a hacker will take intelligence, practice, dedication, and hard work.

Therefore, you have to learn to distrust attitude and respect competence of every kind. Hackers won't let posers waste their time, but they worship competence—especially competence at hacking, but competence at anything is good. Competence at demanding skills that few can master is especially good, and competence at demanding skills that involve mental acuteness, craft, and concentration is best.

If you revere competence, you'll enjoy developing it in yourself—the hard work and dedication will become a kind of intense play rather than drudgery. That attitude is vital to becoming a hacker.

The complete essay can be found online at *http://www.catb.org/~esr/faqs/hacker-howto.html* and in an appendix to the "The Cathedral and the Bazaar" book (O'Reilly.)

—Eric S. Raymond

Eric S. Raymond is the author of the New Hacker's Dictionary, based on the Jargon File, and the famous "Cathedral and the Bazaar" essay that served as a catalyst for the Open Source movement. The text in this Foreword is an excerpt from his 1996 essay, "What is a hacker?" Raymond argues that hackers are ingenious at solving interesting problems, an idea that is the cornerstone of O'Reilly's Hacks series.

Preface

A hacker does for love what others would not do for money.
—/usr/games/fortune

The word *hack* has many connotations. A "good hack" makes the best of the situation of the moment, using whatever resources are at hand. An "ugly hack" approaches the situation in the most obscure and least understandable way, although many "good hacks" may also appear unintelligible to the uninitiated.

The effectiveness of a hack is generally measured by its ability to solve a particular technical problem, inversely proportional to the amount of human effort involved in getting the hack running. Some hacks are scalable and some are even sustainable. The longest running and most generally accepted hacks become standards and cause many more hacks to be invented. A good hack lasts until a better hack comes along.

A hack reveals the interface between the abstract and wonderfully complex mind of the designer, and the indisputable and vulgar experience of human needs. Sometimes, hacks may be ugly and only exist because someone had an itch that needed scratching. To the engineer, a hack is the ultimate expression of the Do-It-Yourself sentiment: no one understands how a hack came to be better than the person who felt compelled to solve the problem in the first place. If a person with a bent for problem solving thinks a given hack is ugly, then they are almost always irresistibly motivated to go one better—and hack the hack, something that we encourage the readers of this book to do.

In the end, even the most capable server, with the most RAM and running the fastest (and most free) operating system on the planet, is still just a fancy back-scratcher fixing the itch of the moment, until a better, faster and cheaper back-scratcher is required.

Where does all of this pseudo-philosophical rambling get you? Hopefully, this background will give you some idea of the mindset that prompted the compiling of this collection of solutions that we call Linux Server Hacks. Some are short and simple, while some are quite complex. All of these hacks are designed to solve a particular technical problem that the designer simply couldn't let go without "scratching." I hope that some of them will be directly applicable to an "itch" or two that you may have felt yourself as a new or experienced administrator of Linux servers.

How This Book Is Organized

A competent sysadmin must be a jack-of-all-trades. To be truly effective, you'll need to be able to handle every problem the system throws at you, from power on to halt. To assist you in the time in between, I present this collection of time-saving and novel approaches to daily administrative tasks.

- *Server Basics* begins by looking at some of the most common sorts of tasks that admins encounter: manipulating the boot process, effectively working with the command line, automating common tasks, watching (and regulating) how system resources are used, and tuning various pieces of the Linux kernel to make everything run more efficiently. This isn't an introduction to system administration but a look at some very effective and non-obvious techniques that even seasoned sysadmins may have overlooked.
- *Revision Control* gives a crash-course in using two fundamental revision control systems, RCS and CVS. Being able to recall arbitrary previous revisions of configuration files, source code, and documentation is a critical ability that can save your job. Too many professional admins are lacking in revision control basics (preferring instead to make the inevitable, but unsupportable *.old* or *.orig* backup). This section will get you up and running quickly, giving you commands and instructions that are succinct and to the point.
- The next section, *Backups*, looks at quick and easy methods for keeping spare copies of your data. I pay particular attention to network backups, *rsync*, and working with ISOs. I'll demonstrate some of the enormous flexibility of standard system backup tools and even present one way of implementing regular "snapshot" revisions of a filesystem (without requiring huge amounts of storage).
- *Networking* is my favorite section of this entire book. The focus isn't on basic functionality and routing, but instead looks at some obscure but insanely useful techniques for making networks behave in unexpected ways. I'll set up various kinds of IP tunnels (both encrypted and

otherwise), work with NAT, and show some advanced features that allow for interesting behavior based on all kinds of parameters. Did you ever want to decide what to do with a packet based on its data contents? Take a look at this section.

- *Monitoring* is an eclectic mix of tips and tools for finding out exactly what your server is up to. It looks at some standard (and some absolutely required "optional") packages that will tell you volumes about who is using what, when, and how on your network. It also looks at a couple of ways to mitigate inevitable service failures and even help detect when naughty people attempt to do not-so-nice things to your network.
- Truly a font of hackery unto itself, the *SSH* section describes all sorts of nifty uses for *ssh*, the cryptographically strong (and wonderfully flexible) networking tool. There are a couple of versions of *ssh* available for Linux, and while many of the examples will work in all versions, they are all tested and known to work with OpenSSH v3.4p1.
- *Scripting* provides a short digression by looking at a couple of odds and ends that simply couldn't fit on a single command line. These hacks will save you time and will hopefully serve as examples of how to do some nifty things in shell and Perl.
- *Information Services* presents three major applications for Linux: *BIND 9*, *MySQL*, and *Apache*. This section assumes that you're well beyond basic installation of these packages, and are looking for ways to make them deliver their services faster and more efficiently, without having to do a lot of work yourself. You will see methods for getting your server running quickly, helping it scale to very large installations and behave in all sorts of slick ways that save a lot of configuration and maintenance time.

How to Use This Book

You may find it useful to read this book from cover to cover, as the hacks do build on each other a bit from beginning to end. However, each hack is designed to stand on its own as a particular example of one way to accomplish a particular task. To that end, I have grouped together hacks that fit a particular theme into sections, but I do cross-reference quite a bit between hacks from different sections (and also to more definitive resources on the subject). Don't consider a given section as a cut-and-dried chapter with rigidly defined subject boundaries but more as a convenient way of collecting similar (and yet independent) hacks. You may want to read this book much like the way most people browse web pages online: follow whatever interests you, and if you get lost, follow the links within the piece to find more information.

Conventions Used in This Book

The following is a list of the typographical conventions used in this book:

Italic
: Used to indicate new terms, URLs, filenames, file extensions, directories, commands and options, and program names.

`Constant Width`
: Used to show code examples, the contents of files, or the output from commands.

`Constant Width Bold`
: Used in examples and tables to show commands or other text that should be typed literally.

`Constant Width Italic`
: Used in examples and tables to show text that should be replaced with user-supplied values.

The thermometer icons, found next to each hack, indicate the relative complexity of the hack:

 beginner  moderate  expert

How to Contact Us

We have tested and verified the information in this book to the best of our ability, but you may find that features have changed (or even that we have made mistakes!). Please let us know about any errors, inaccuracies, bugs, misleading or confusing statements, and typos that you find in this book.

You can write to us at:

O'Reilly & Associates, Inc.
1005 Gravenstein Hwy N.
Sebastopol, CA 95472
(800) 998-9938 (in the U.S. or Canada)
(707) 829-0515 (international/local)
(707) 829-0104 (fax)

To ask technical questions or to comment on the book, send email to:

bookquestions@oreilly.com

Visit the web page for *Linux Server Hacks* to find additional support information, including examples and errata. You can find this page at:

http://www.oreilly.com/catalog/linuxsvrhack

For more information about this book and others, see the O'Reilly web site:

http://www.oreilly.com

Gotta Hack?

Got a good hack you'd like to share with others? Go to the O'Reilly Hacks web site at:

http://hacks.oreilly.com

You'll find book-related resources, sample hacks and new hacks contributed by users. You'll find information about additional books in the Hacks series.

CHAPTER ONE

Server Basics

Hacks #1–22

A running Linux system is a complex interaction of hardware and software where invisible daemons do the user's bidding, carrying out arcane tasks to the beat of the drum of the uncompromising task master called the Linux kernel.

A Linux system can be configured to perform many different kinds of tasks. When running as a desktop machine, the visible portion of Linux spends much of its time controlling a graphical display, painting windows on the screen, and responding to the user's every gesture and command. It must generally be a very flexible (and entertaining) system, where good responsiveness and interactivity are the critical goals.

On the other hand, a Linux server generally is designed to perform a couple of tasks, nearly always involving the squeezing of information down a network connection as quickly as possible. While pretty screen savers and GUI features may be critical to a successful desktop system, the successful Linux server is a high performance appliance that provides access to information as quickly and efficiently as possible. It pulls that information from some sort of storage (like the filesystem, a database, or somewhere else on the network) and delivers that information over the network to whomever requested it, be it a human being connected to a web server, a user sitting in a shell, or over a port to another server entirely.

It is under these circumstances that a system administrator finds their responsibilities lying somewhere between deity and janitor. Ultimately, the sysadmin's job is to provide access to system resources as quickly (and equitably) as possible. This job involves both the ability to design new systems (that may or may not be rooted in solutions that already exist) and the talent (and the stomach) for cleaning up after people who use that system without any concept of what "resource management" really means.

The most successful sysadmins remove themselves from the path of access to system resources and let the machines do all of the work. As a user, you know that your sysadmin is effective when you have the tools that you need to get the job done and you never need to ask your sysadmin for anything. To pull off (that is, to hack) this impossible sounding task requires that the sysadmin anticipate what the users' needs will be and make efficient use of the resources that are available.

To begin with, I'll present ways to optimize Linux to perform only the work that is required to get the job done and not waste cycles doing work that you're not interested in doing. You'll see some examples of how to get the system to do more of the work of maintaining itself and how to make use of some of the more obscure features of the system to make your job easier. Parts of this section (particularly Command Line and Resource Management) include techniques that you may find yourself using every day to help build a picture of how people are using your system and ways that you might improve it.

These hacks assume that you are already familiar with Linux. In particular, you should already have root on a running Linux system available with which to experiment and should be comfortable with working on the system from the command line. You should also have a good working knowledge of networks and standard network services. While I hope that you will find these hacks informative, they are certainly not a good introduction to Linux system administration. For in-depth discussion on good administrative techniques, I highly recommend the *Linux Network Administrator's Guide* and *Essential System Administration*, both by O'Reilly and Associates.

The hacks in this chapter are grouped together into the following five categories: Boot Time, Command Line, Automation, Resource Management, and Kernel Tuning.

Boot Time

1. Fine tune your server to provide only the services you really want to serve
2. Forgoing the Console Login
3. Common Boot Parameters
4. Creating a Persistent Daemon with init

Command Line

5. n>&m: Swap Standard Output and Standard Error
6. Building Complex Command Lines
7. Working with Tricky Files in xargs

8. Immutable Files in ext2/ext3
9. Speeding Up Compiles

Automation

10. At home in your shell environments
11. Finding and eliminating setuid/setgid binaries
12. Make sudo work harder for you
13. Using a Makefile to automate admin tasks
14. Brute forcing your new domain name

Resource Management

15. Playing Hunt the Disk Hog
16. Fun with /proc
17. Manipulating processes symbolically with procps
18. Managing system resources per process
19. Cleaning up after ex-users

Kernel Tuning

20. Eliminating unnecessary drivers from the kernel
21. Using large amounts of RAM
22. hdparm: fine tune IDE drive parameters

HACK #1 Removing Unnecessary Services

Fine tune your server to provide only the services you really want to serve

When you build a server, you are creating a system that should perform its intended function as quickly and efficiently as possible. Just as a paint mixer has no real business being included as an espresso machine attachment, extraneous services can take up resources and, in some cases, cause a real mess that is completely unrelated to what you wanted the server to do in the first place. This is not to say that Linux is incapable of serving as both a top-notch paint mixer and making a good cup of coffee simultaneously—just be sure that this is exactly what you intend before turning your server loose on the world (or rather, turning the world loose on your server).

When building a server, you should continually ask yourself: what do I really need this machine to do? Do I really need FTP services on my web server? Should NFS be running on my DNS server, even if no shares are exported? Do I need the automounter to run if I mount all of my volumes statically?

To get an idea of what your server is up to, simply run a *ps ax*. If nobody is logged in, this will generally tell you what your server is currently running. You should also see what programs for which your *inetd* is accepting connections, with either a `grep -v ^# /etc/inetd.conf` or (more to the point) `netstat -lp`. The first command will show all uncommented lines in your *inetd.conf*, while the second (when run as root) will show all of the sockets that are in the LISTEN state, and the programs that are listening on each port. Ideally, you should be able to reduce the output of a *ps ax* to a page of information or less (barring preforking servers like *httpd*, of course).

Here are some notorious (and typically unnecessary) services that are enabled by default in many distributions:

portmap, rpc.mountd, rpc.nfsd
: These are all part of the NFS subsystem. Are you running an NFS server? Do you need to mount remote NFS shares? Unless you answered *yes* to either of these questions, you don't need these daemons running. Reclaim the resources that they're taking up and eliminate the potential security risk.

smbd and nmbd
: These are the Samba daemons. Do you need to export SMB shares to Windows boxes (or other machines)? If not, then these processes can be safely killed.

automount
: The automounter can be handy to bring up network (or local) filesystems on demand, eliminating the need for root privileges when accessing them. This is especially handy on client desktop machines, where a user needs to use removable media (such as CDs or floppies) or to access network resources. But on a dedicated server, the automounter is probably unnecessary. Unless your machine is providing console access or remote network shares, you can *kill* the automounter (and set up all of your mounts statically, in */etc/fstab*).

named
: Are you running a name server? You don't need *named* running if you only need to resolve network names; that's what */etc/resolv.conf* and the bind libraries are for. Unless you're running name services for other machines, or are running a caching DNS server (see "Setting Up Caching DNS with Authority for Local Domains" **[Hack #78]**), then *named* isn't needed.