Automating Linux and Unix System Administration

Second Edition

Nate Campi and Kirk Bauer

Apress®

Automating Linux and Unix System Administration, Second Edition

ISBN-13 (pbk): 978-1-4302-1059-7

ISBN-13 (electronic): 978-1-4302-1060-3

Printed and bound in the United States of America 9 8 7 6 5 4 3 2 1

Lead Editor: Frank Pohlmann
Technical Reviewer: Mark Burgess
Editorial Board: Clay Andres, Steve Anglin, Mark Beckner, Ewan Buckingham, Tony Campbell, Gary Cornell, Jonathan Gennick, Michelle Lowman, Matthew Moodie, Jeffrey Pepper, Frank Pohlmann, Ben Renow-Clarke, Dominic Shakeshaft, Matt Wade, Tom Welsh
Project Manager: Kylie Johnston
Copy Editors: Nina Goldschlager, Heather Lang
Associate Production Director: Kari Brooks-Copony
Production Editor: Ellie Fountain
Compositor: Linda Weidemann, Wolf Creek Press
Proofreader: Nancy Sixsmith
Indexer: Becky Hornyak
Cover Designer: Kurt Krames
Manufacturing Director: Tom Debolski

Distributed to the book trade worldwide by Springer-Verlag New York, Inc., 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax 201-348-4505, e-mail orders-ny@springer-sbm.com, or visit http://www.springeronline.com.

For information on translations, please contact Apress directly at 2855 Telegraph Avenue, Suite 600, Berkeley, CA 94705. Phone 510-549-5930, fax 510-549-5939, e-mail info@apress.com, or visit http://www.apress.com.

Apress and friends of ED books may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Special Bulk Sales–eBook Licensing web page at http://www.apress.com/info/bulksales.

The source code for this book is available to readers at http://www.apress.com.

I dedicate this book to my dear grandmother Mary Lou.
Her influence makes everyone around her a better person,
and her presence lights up a room.
She is beautiful inside and out,
and she meets adversity with faith,
quiet dignity, and grace.

—Nate Campi

Contents at a Glance

Contents

About the Authors

NATE CAMPI is a UNIX and Linux system administrator by trade, currently working as a UNIX operations manager in San Francisco. His system administration experience is almost entirely with companies with large-scale web operations based on open source software. In his copious free time, he enjoys jogging, watching spaghetti westerns, experimenting with Linux systems, and spending time with his family.

KIRK BAUER has been involved in computer programming since 1985. He has been using and administering UNIX systems since 1994. Although his personal favorite UNIX variant is Linux, he has administered and developed on everything from FreeBSD to Solaris, AIX, and HP-UX. He is the author of various open source solutions such as Logwatch.

Kirk has been involved with software development and system/network administration since his first year at the Georgia Institute of Technology. He has done work for the Georgia Tech Research Institute, Fermi National Accelerator Laboratory, and DHL. In 2000, Kirk was one of the founders and the chief technology officer of TogetherWeb, which was purchased in 2003 by Proficient Systems. Kirk is now a systems engineer with F5 Networks.

Kirk graduated from Georgia Tech in 2001 with a bachelor's degree in computer engineering and is currently pursuing his MBA at Arizona State University. He lives in Peoria, Arizona, with his two dogs, and is looking forward to getting married to his lovely fiancée, Rachel.

About the Technical Reviewer

MARK BURGESS holds a first class honors degree in physics and a Ph.D. in theoretical physics from the University of Newcastle upon Tyne. After working as a physicist, he began to apply the methods of physics to the study of computers and eventually changed research fields to study the formalization of system administration. His current research interests include the behavior of computers as dynamic systems and applying ideas from physics to describe computer behavior. Mark is the author of the popular configuration management software package cfengine. He has received a number of awards including the SAGE 2003 Professional Contribution Award "for groundbreaking work in systems administration theory and individual contributions to the field." He currently holds the Professorship in Network and System Administration at Oslo University College.

Acknowledgments

Only two names are on the book cover, but many talented and dedicated people worked to make this book the best it could be.

We are very grateful to Paul W. Frields from Red Hat for Red Hat Enterprise Linux licenses. This book wouldn't have been possible without them. Mark Burgess lent his unique insight into both cfengine and the book writing process. Our editor Frank Pohlmann is incredibly skilled at finding the weak points in a description and forcing us to explain everything thoroughly. Thanks to our project manager Kylie Johnston; she is a consummate professional. Thanks to our copy editors Nina Goldschlager and Heather Lang, who are very talented and easy to work with. And thanks to our production editor Ellie Fountain.

We really need to thank our families for putting up with our mental absence while writing this book.

Finally, we'd like to thank the energy drink industry for enabling us to stay up late at night even when totally exhausted, go to work the next day feeling like we had been hit by a train, and do it all over again the very next night.

Introduction

The system administrator is one of the users of a system, and something more. The administrator wears many hats, as knowledgeable user of UNIX commands, as an operator of system hardware, and as a problem solver. The administrator is also called upon to be an arbitrator in human affairs. A multiuser computer is like a vast imaginary space where many people work and utilize the resources found there. The administrator must be the village elder in this space and settle the disputes that may arise with, hopefully, the wisdom of Solomon.

—Rebecca Thomas and Rik Farrow
(*UNIX Administration Guide for System V,*
Pearson PTR, 1989)

We find it interesting how little UNIX system administration has changed in the last twenty years. If you substitute "computer network" for "multiuser computer," this description still fits perfectly.

The main difference in UNIX system administration between 1989 and 2008 (besides ubiquitous networking) is the sheer number of systems that the average system administrator deals with. Automation is the primary tool to deal with the chaos that can result from so many systems. With it, you can deploy systems identically every time, restore systems to a known good state, and implement changes reliably across all systems (or only an appropriate subset).

We do not claim that the approaches, procedures, and tools used in this book are the only way to set up and maintain a UNIX-based environment. Instead, we walk you through the creation of an example environment, and during the process, help you gain a solid understanding of the basic principles of system automation. This way, you can decide for yourself how you want to set up your own UNIX-based environment.

This book *isn't* like most UNIX/Linux administration books, because it illustrates techniques and principles by building a real UNIX/Linux environment from scratch. We demonstrate that you can configure each host at your site, from installation through production service to system retirement, without logging in and making manual changes to the host. Instead, we'll configure the hosts via imaging systems designed for unattended installation, followed by management with an automation framework.

We wrote this book, because we felt that it is important to demonstrate that an entire site can be managed using automation. Our goal is to be able to quickly, easily, and reliably restore hosts to service after complete system failure. The host might have failed

due to hardware issues; an entire geographic region might be unreachable due to natural disaster, or you might simply have purchased updated hardware on which to run that particular host and need to upgrade. The point of our approach is to configure a host only once and, from that point on, allow an automation system to do that work for you.

Whether you choose to use our exact setup or something completely different, you'll have gained knowledge and experience by going though the process with us in our example environment. Our promise to you is that if you need to configure a new UNIX-based infrastructure from scratch (and you're able or allowed to use the operating systems and software we demonstrate), you can use this book to create a fully functional and scalable new infrastructure. Every service and piece of architecture that our new environment needs is set up using automation.

This book moves fast and will be best utilized if you follow along with the examples and implement the described steps on systems of your own. In addition, download the code and configuration files from the Source Code page of the Apress web site (`http://www.apress.com`).

Who This Book Is For

This book is written for the experienced system administrator. We have made every attempt to refer you to appropriate external sources when we weren't able to delve into great detail on a service or protocol that we were automating. In addition, little explanation is given to the usage of basic UNIX/Linux commands and shell scripts. You don't, however, have to be an advanced system administrator. We feel that a system administrator with only one or two years of full-time on-the-job experience is more than ready to utilize the concepts and tools in this book.

How This Book Is Structured

The book begins with four introductory chapters that you should be very familiar with before you move on to later, more detailed chapters. The later chapters, starting with Chapter 5, build a new UNIX environment: we set up an automation system; automate installation systems; and enhance the site with real applications, monitoring, reporting, and security.

Chapter 1, "Introducing the Basics of Automation," covers the reasons for and benefits of automation, as well as the methodology behind it. Also, the `sudo` utility is introduced and explained.

Chapter 2, "Applying Practical Automation," covers the steps behind automating a common procedure—adding a new user account. During the process, the core tenets of automation are covered.

Chapter 3, "Using SSH to Automate System Administration Securely," covers the basics of using secure shell (SSH), discusses SSH security concerns, describes how to set up public key authentication in SSH, and delves into various other related topics such as SSH log analysis.

Chapter 4, "Configuring Systems with cfengine," explains the concepts behind cfengine, as well as the various cfengine daemons and utilities. A full discussion takes place of the common configuration settings in the main cfengine configuration file. The requirements for a minimal cfengine architecture with two hosts are fully explored.

Chapter 5, "Bootstrapping a New Infrastructure," covers the cfengine configuration for a new, automated UNIX/Linux environment. A "master" cfengine host is set up, with all the required configuration files to manage new Red Hat Linux, Debian Linux, and Solaris hosts. This is the first step in building a UNIX/Linux environment from scratch using automation.

Chapter 6, "Setting Up Automated Installation," demonstrates the automated installation of Red Hat Linux using Kickstart, Debian Linux using Fully Automatic Installation (FAI), and Sun Solaris using Jumpstart. The hosts deployed in this chapter continue to be used in the later development of our example UNIX/Linux infrastructure.

Chapter 7, "Automating a New System Infrastructure," covers the automation of these services and procedures in our new infrastructure: the Network Time Protocol (NTP), Domain Name System (DNS), standardized local account files and new user accounts, mail routing, and home directories mounted with the Network File System (NFS).

Chapter 8, "Deploying Your First Application," covers the deployment and configuration of the Apache web server, demonstrating various ways to automate the distribution of both the web server daemon binaries and the web content. Along the way, you learn about sharing data with NFS, rsync, `scp`, cfengine data copies, and Subversion.

Chapter 9, "Generating Reports and Analyzing Logs," covers automated syslog and cfengine log analysis and reporting in our new infrastructure.

Chapter 10, "Monitoring," uses cfengine to automate the deployment and configuration of Ganglia and Nagios in our example environment.

Chapter 11, "Infrastructure Enhancement," uses cfengine to manage version control with Subversion, including branching the cfengine configuration tree to create testing and development environments. Also, backups are handled, in a very simple way.

Chapter 12, "Improving System Security," covers the implementation of security enhancements with cfengine. Measures undertaken include removing the SUID bit from root-owned binaries, protecting system accounts, applying UNIX/Linux patches and vendor updates, shutting down unneeded daemons, adding host-based firewalls, and more.

Appendix A, "Introducing the Basic Tools," provides a basic introduction to the tools used throughout this book and provides a good starting point for understanding and utilizing the examples presented in this text. This appendix covers the following tools: bash, Perl, `grep`, `sed`, and AWK.

Appendix B, "Writing cfengine Modules," covers extending cfengine through modules. This is a quick but thorough introduction using examples.

Downloading the Code

The source code for this book is available to readers at `http://www.apress.com` in the Source Code section of this book's home page. Please feel free to visit the Apress web site and download all the code there. You can also check for errata and find related titles from Apress.

Contacting the Authors

We have gone through several stages of proofreading and error checking during the production of this book in an effort to reduce the number of errors. We have also tried to make the examples and the explanations as clear as possible.

There may, however, still be errors and unclear areas in this book. If you have questions or find any of these errors, please feel free to contact us at `nate@campin.net`. You can also visit the Apress web site at `http://www.apress.com` to download code from the book and see any available errata.

CHAPTER 1

Introducing the Basics of Automation

When one of this book's authors was in high school, he got his first part-time job keeping some of the school's computers running. He loved it. He did everything by hand. And because the school had only two or three computers, doing everything by hand wasn't a big issue. But even then, as the number of systems grew to five, six, and finally more than ten, he realized just how much time you can spend doing the same things over and over again. This is how his love of automation was born.

This book's other author found automation through necessity as well, although later in his career. During the so-called "tech downturn" around the year 2003 in Silicon Valley, he suddenly found himself the sole member of what had been a three-person system-administration team. The number of systems and responsibilities were increasing, while staffing levels had dramatically decreased. This is when he found the cfengine automation framework. Cfengine drastically reduced the amount of time required to implement system changes, allowing him to focus on improving the infrastructure instead.

In this chapter you will learn the basics of automating system administration so that you can begin to make your life easier—as well as the lives of everybody who uses or depends on your systems. The topics covered in this book apply to a wide variety of situations. Whether you have thousands of isolated systems (sold to your customers, for example), a large number of diverse machines (at a large company or university campus), or just a few servers in your home or small business, the techniques we'll cover will save you time and make you a better administrator.

Throughout this book, we will assume the reader has a basic set of UNIX skills and some prior experience as a system administrator (SA). We will use numerous tools throughout the book to provide example automation solutions. These tools include the following:

- The Bash shell
- Perl
- Cfengine

- Regular expressions
- The `grep` command
- The `sed` stream editor
- AWK

If you are not familiar with one or more of these tools, read their introductions in the Appendix before you proceed. See Chapter 4 for an introduction to cfengine.

Do You Need Automation?

If you have one Linux system sitting on your desk at home, you don't *need* automation. You can take care of everything manually—and many people do. But you might *want* automation anyway because it will ensure your system has the following characteristics:

- *Routine tasks such as performing backups and applying security updates take place as scheduled*: This saves the user time and ensures that important tasks aren't forgotten.
- *The system is consistently set up*: You might have one system, but how often is it replaced due to faulty hardware or upgrades? When the system hardware is upgraded or replaced, an automation system will configure the software again in the same manner as before.
- *The system can be expertly configured, even if you're not an expert*: If you use automation built by someone more experienced with system configuration and automation, you benefit from his or her expertise. For example, you benefit from the Red Hat Network (RHN) when using a licensed installation of Red Hat Enterprise Linux. RHN regularly supplies automated software updates that are reliable and timely, resulting in a more secure and stable system. Most users don't have the required system configuration and programming skills to implement such a system, so Red Hat developed a solution that any of their software licensees can use freely.
- *The system is in compliance with guidelines and standards*: You might be responsible for only one system, but if the system belongs to your employer, it might be subject to regulatory or other legislative requirements around security and configuration. If this is the case, an automation system that enforces those requirements supplies the documentation needed to prove compliance. Even if no laws or credit

card–company guidelines apply, your employer might require that all systems on its network meet certain minimal security standards. Usually a one-time manual configuration isn't enough to satisfy these standards; an automated solution is required.

- *The system is reliable*: If solutions to occasional problems are automated, the system is more reliable. When a disk fills up with temporary files, for example, the user who employs an automation system can schedule a daily cleanup procedure to prevent failed writes to disk and system crashes from full disks.

Likewise, you might think you don't need automation if you have only one server in your company. However, you might want it because backups and timely security updates are easy tasks for a busy system administrator to neglect, even in this most basic setup. In addition, if your company's server is a file server or mail server, its drives will tend to fill up and cause problems. In fact, any security or stability problem with this type of computer will likely result in expenses for the company, and any loss of data could be disastrous. This is exactly the reason OS vendors rotate the log files for the daemons they install on the system, because they know the end result of unmaintained log files. An automation system can also help out your successor or the person covering for you during your vacation.

When it comes down to it, the number of machines isn't an important factor in the decision to use automation. Think of automation as insurance that the machine is being monitored. A Red Hat Package Manager (RPM) install or security update can undo a manual change to a configuration file, for example. If an automation system enforces the policy that the configuration file contains a particular entry or value, it will reapply the change if necessary.

In addition to log-file rotation, your OS distributor already automates many tasks on a stand-alone system. It makes security checks, updates databases with information on file locations (e.g., `slocate`), and collects system accounting and performance information. All this and more happens quietly and automatically from within a standard UNIX or Linux system.

Automation is already a core part of UNIX philosophy, and cron jobs have historically been the de facto method for automating UNIX tasks. In this book we favor cfengine for task automation, but for now you can think of cfengine as a next-generation cron daemon.

For the sake of the single system, it's fine to go the simple route. You can add more log-rotation settings to already automated systems such as the "logrotate" utility (standard on all Linux distributions that we can think of). You don't need something complex, but you do need automation if you want to ensure important tasks happen regularly and reliably.

You should do everything you can to prevent problems before they happen. If you can't do that, follow the advice of one of our old managers: make sure the same problem

never happens again. If a disk fills, set up a log-rotation script run from cron that deletes unneeded temporary files—whatever addresses the root cause. If a process dies, set up a process monitor to restart it when it exits. In later chapters, we will show you how to accomplish these tasks using cfengine. The automation systems at most sites grow over time in response to new issues that arise.

SAs who respond to all problems with permanent (read: automated) solutions go a long way toward increasing overall availability of their sites' applications and services. Automated solutions also allow them to get some sleep while on call. (The sleep factor alone is reason enough for most SAs to spend a lot of time on automation.)

So, back to the question—do you *need* automation? We'll introduce a variety of situations that require automation and discuss them further throughout the book.

Large Companies with Many Diverse Systems

The most traditional situation requiring automation involves a large company or organization with hundreds or even thousands of systems. These systems range from web servers to file servers to desktop workstations. In such a situation, you tend to have numerous administrators and thousands of users.

You might treat the systems as several groups of specialized servers (i.e., all workstations in one group, all web servers in another) or you might administer all of them together. Either way, with a large number of different systems, automation is the only option. Cfengine is especially suited to this type of environment. It uses a high-level configuration file and allows each system to pull its configuration from the configuration server. One of cfengine's key strengths: Not only can it configure hundreds or even thousands of systems in exactly the same manner, but it can also configure a single system in a unique way. We'll discuss cfengine thoroughly in later chapters.

Medium-Sized Companies Planning for Growth

Any medium-sized or small company is in just about the same situation as the large companies. You might have only 50 servers now and some basic solutions might work for you, but you probably hope to expand. Automation systems built on cfengine scale from a few systems to many thousands of systems. The example cfengine infrastructure demonstrated in Chapter 5 assists scalability by segmenting the configuration into many files. Sites with more than 25,000 hosts use cfengine.

You might have only one type of a particular system, but if it fails, cfengine can reproduce the original system quickly and reliably. Normally at that point some user or application data needs to be restored, but that's much easier than reproducing a system from a base install.

Internet Service Providers

If you work at an Internet Service Provider (ISP), you probably have more computers than employees. You also (hopefully) have a large number of customers who pay you money for the service you provide. Your systems might offer a wide variety of services, and you need to keep them all running. Other types of companies have some critical servers, but most of their systems are not critical for the companies' success (e.g., individual workstations, testing systems, and so on). At an ISP, almost all of your systems are critical, so you need to create an automation system that promotes system stability and availability.

Application Service Providers

If you're an application service provider (ASP), you might have hundreds of systems that all work together or numerous groups of independent systems. Your system-administration tasks probably include deploying and configuring complex, custom software. You must synchronize such changes among the various systems and make them happen only on demand. Stability is very important, and by minimizing changes you can minimize downtime. You might have a central administration system or a separate administration for each group of systems (or both). When you create your automation system, be sure to keep an eye on scalability—how many systems do you have now, and how many will you have in the future?

Fortunately with cfengine you already have an automation system; what you need to keep in mind is that in such an environment you often need additional capacity in a hurry. Being able to boot new hardware off the network and have cfengine configure it appropriately means that the most time-consuming aspect of adding new systems is the time required to order, rack, and cable up the new systems. This is the ideal situation for an ASP, and the SA staff in such shops should aspire to it.

Web Server Farms

Automation within web clusters is common today. If you have only a couple of load balancers and a farm of web servers behind them, all your systems will be virtually identical. This makes things easier because you can focus your efforts on scalability and reliability without needing to support differing types of systems. In a more advanced situation, you also have database systems, back-end servers, and other systems. In this case, you need a more flexible automation system, such as cfengine. Regardless of the underlying infrastructure, web servers will be plentiful. You need a quick and efficient way to install and configure new systems (for expansion and recovery from failures). Sound familiar? The core needs and considerations are common across different business types. We'll return to these recurring themes at the end of the chapter.

Beowulf Clusters

Beowulf clusters are large groups of Linux systems that can perform certain tasks on par with a traditional supercomputer. Regardless of whether you use a Beowulf cluster or another type of computational cluster, each cluster usually has one control system and hundreds of computational units. To set up and maintain the cluster efficiently, you need the ability to install new systems with little or no interaction. You have a set of identical systems, which makes configuration easy. You also usually have maintenance periods during which you can do what you want on the systems, which is always nice. But when the systems are in use, making changes to them might be disastrous. For this reason, you will usually want to control the times when the systems will accept modifications.

Hosts in such clusters will typically boot off the network and load up a minimal operating system entirely into memory. Any local storage on the system is probably for application data and temporary storage. Many of the network boot schemes like this completely ignore the containment of system drift during the time between boot and shutdown.

In a worst-case scenario, an attacker might access the system and modify running processes, access other parts of your network from there, or launch attacks against other sites. A less extreme problem would be one where certain applications need to be kept running or be restarted if they consume more than a defined amount of memory. An automation system that ignores the need to control a running system is at best only half an automation system. Using a system reboot to restore a known good state is sufficient if the site administrators don't wish to do any investigation or improvement. A reboot is only a temporary solution to a system problem. An attacker will simply come back using the same mechanism as before, or processes will still die or grow too large after a reboot. You need a permanent solution.

A cluster designed to network-boot can just as easily run cfengine and use it to contain system drift. You'll find helpful cfengine features that can checksum security-critical files against a known good copy and alert administrators to modifications. Other cfengine features can kill processes that shouldn't be running or restart daemons that are functioning incorrectly. Systems that are booted from identical boot media don't always have the same runtime behavior, and cfengine allows you to control the runtime characteristics of your systems.

For some of the best documentation on system drift and ways to control it, check out the book *Principles of Network and System Administration, Second Edition* by Mark Burgess (Wiley, 2004). The author approaches the subject from an academic standpoint, but don't let that scare you away. He uses real-world examples to illustrate his points, which can pay off at your site by helping you understand the reasons behind system drift. The book will help you minimize these negative effects in your system and application design.

Network Appliances

Finally, many companies produce what we call "network appliances," which are systems that run some UNIX variant (often Linux or FreeBSD) and are sold to customers as a "drop-in" solution. Some current examples of these products include load balancers and search engines. The end user administers the systems but might know very little about UNIX. End users also usually do not have `root` access to the system. For this reason, the system must be able to take care of itself, performing maintenance and fixing problems automatically. It will also need to have a good user interface (usually web-based) that allows the customer to configure its behavior. Such vendors can leverage cfengine so that they can focus on their core competency without worrying about writing new code to keep processes running or file permissions correct.

What Will You Gain?

The day-to-day work of system administration becomes easier with automation. We can promise the following benefits, based on our own experience.

Saving Time

You can measure the time saved by automation in two ways. The first is in the elapsed wall-clock time between the start and end of a task. This is important, but not as important as the amount of actual SA time required. If the only SA time required is in setting up the task to be automated in the first place and occasionally updating the automation from time to time, the benefits are much greater than faster initial completion. This frees the SA to work on automating more tasks, testing out new software, giving security or reliability lectures to the in-house programmers, or simply keeping current with recent technology news.

Reducing Errors

Unfortunately, you'll see a rather large difference between systems built according to documentation and systems configured entirely through automated means. If you were to audit two systems for differences at a site where all systems were configured by cfengine, the differences should—in theory—arise only from errors outside the automation system, such as a full disk. We know from firsthand experience that systems configured according to a written configuration guide invariably differ from one another. After all, humans are fallible. We make mistakes.

You can reduce errors at your site by carefully testing automated changes in a non-production environment first. When the testing environment is configured properly, only then do you implement the change in your production environment.

For the sake of this book, the term "production" means the systems upon which the business relies, in any manner. If the company is staffed primarily with nontechnical people, perhaps only the SA staff understands the differentiation when the term is used. Trust us, though: the business people understand when particular hosts are important to the business and will speak out about perceived problems with those systems.

Documenting System Configuration Policies

Whether the automated configuration at a site is done by shell scripts, Perl scripts, or a tool such as cfengine, the automation serves as documentation. It is in fact some of the most usable documentation for a fellow SA, simply because it is authoritative.

If new SAs at a site read some internal documentation about installing and configuring some software, they don't have any assurance that following the documentation will achieve the desired effect. The SA is much better off using a script that has been used all the previous times the software needed to be installed and configured.

Either the script will work and the proper results will emerge, or it'll break because of some change in the environment. The change should be much easier to find based on error output from the script. If the steps on a wiki page or a hard copy of the documentation don't work, on the other hand, the error could be due to typos in the doc, steps omitted, or updates to the procedure not making it back into the docs. Using automation instead helps insulate the SA against these scenarios.

Realizing Other Benefits

This book applies to a wide range of people and situations, so not all the material will be of interest to all readers. If you haven't yet created an automation system or implemented an open source framework (such as cfengine) from scratch, this book will show you how to get started and how to take the system from initial implementation through full site automation. You will also learn the principles that should guide you in your quest for automation. As your skills and experience grow, you will become more interested in some of the more advanced topics the book discusses and find that it points you in the right direction on related subjects.

If you already have an automation system of some sort, this book will provide you with ideas on how to expand it. There are so many ways to perform any given task that you are sure to encounter new possibilities. In many cases, your current system will be advanced enough to leave as is. In other cases, though, you will find new ways to automate old tasks and you'll find new tasks that you might never have considered automating.

Don't write off a complicated manual task as too difficult to automate before carefully evaluating the decisions made during the process. You'll usually find during manual inspection that the decision process is based on attributes of the system that cfengine or a script can collect. The act of documenting a change before making it usually forces the SA to approach the problem in a systematic way. The change process will end up producing better results when the process is planned this way.

Imagine that you often have to restart a web-server process on one of your servers, in a sequence of actions such as this:

- You check a log file for a commonly recurring error message.
- You check if CPU utilization is high.
- You test the web server using a command-line utility, looking for a successful HTTP status message.

You can collect each of these manual checks automatically, and a script or cfengine can make the decision to restart. If this makes you nervous, write the script's collection aspects first, and at the point where a system change would be made, instruct the script to print a message to the screen about the decision it has reached. Run the script, then manually go through your decision process independently of the script. Enhance the script each time its decision differs from yours. You'd be surprised at the complex procedures you can automate this way. You don't have to enable the automated restart itself until you're comfortable that it will do the right thing.

AUTOMATING A DIFFICULT PROBLEM/RESPONSE PROCEDURE

One of us works at a site where the SA staff used complex manual procedures to fix a distributed cluster when application errors would occur. The manual process would often take several hours to completely restore the cluster to a working state.

The staff slowly automated the process, beginning with simple commands in a shell script to avoid repeatedly typing the same commands. Over time the staff enhanced the script with tests to determine which errors were occurring and to describe the state of the cluster's various systems. Based on these tests, the script could determine and perform the correct fix.

Eventually, the SA staff used the automated process to repair the cluster in as little as a few minutes. In addition, the script incorporated so many of the decisions previously made by the SA staff members that it became the foremost authority on how to deal with the situation. Essentially, the script serves as documentation on how to deal with multiple issues and situations on that particular application cluster.

When it comes to computer systems, every environment is different—each has different requirements and many unique situations. Instead of attempting to provide the unattainable "one solution fits all," this book shows how to set up an example environment. As we configure our example environment, we will explain the decision process behind the solutions we have chosen. After you have learned these options, you will be able to make an informed choice about what you should automate in your environment and how you should do it.

What Do System Administrators Do?

Life as a system administrator usually falls into three categories:

- Tedious, repetitive tasks (a.k.a. boring tasks)
- New, innovative tasks (a.k.a. why you love the job)
- Answering users' questions, or otherwise dealing with monitoring alarms, issues or emergencies (a.k.a. pulling your hair out)

The goal of this book is to help you create new and innovative solutions to eliminate those tedious and repetitive tasks. And if you find a way to automate the task of answering users' questions, please let us know! But even if you can't, you can at least create a system that detects and even fixes many problems before they come to the attention of the users, or more important, your monitoring systems. Also, any task you have automated is a task the users could potentially perform on their own.

System administrators spend time on other tasks, of course, but we won't address them here because they aren't pertinent to this discussion. (These might include browsing the Slashdot web site, checking on reservations for the next science-fiction convention, or discussing a ham-radio setup with other geeks around the office.) Suffice it to say that following the guidelines in this book will allow you to spend more time on these other tasks and less time on the tedious tasks and emergencies.

You can classify the tedious tasks into the following categories:

- *Preinstallation*: Assigning an IP address, configuring existing servers and network services, and so on
- *Installation*: Installing a new operating system and preparing it for automation
- *Configuration*: Performing initial configuration and reconfiguration tasks
- *Managing data*: Duplicating or sharing data (users' home directories, common scripts, web content, etc.), backups, and restores

- *Maintenance and changes*: Rotating logs, adding accounts, and so on
- *Installing/upgrading software*: Using package management and/or custom distribution methods
- *System monitoring and security*: Performing log analysis and security scans; monitoring system load, disk space, drive failures, and so on

Methodology: Get It Right from the Start!

Automating tasks proves much more useful when you apply a consistent methodology. Not only will you have less direct work (by having code that is easier to maintain and reuse), but you will also save yourself and others time in the future. Whenever possible, we'll include techniques in this book that support these basic methodologies:

- Activities you have performed must be reproducible.
- Any system's state must be verifiable.
- Problems should be detected as they occur.
- Problems should be repaired automatically, if possible.
- The automation methods must be secure.
- The system should be documented and easy to understand.
- Changes should be testable in a safe environment.
- Every system change should be examined for side effects that also must be handled automatically.

Perhaps the most important aspect of any automated system is reproducibility. If you have two machines configured just the way you like them, you should be able to add an identically configured third machine to the group with minimal effort. If somebody makes an incorrect change or loses a file, restoring the system to full functionality should be relatively easy. These nice capabilities all require that you can quickly and perfectly reproduce what you have done in the past or to other systems. Even if you don't plan to add more systems, you can bet that at some point one of your systems will fail. It might be the CPU or disk(s), or you might have a fire in your server room. (You do have a disaster recovery plan, right?) The experienced SA protects his systems against their inevitable failure, and automation is a big part of the solution.

You also need to be able to verify a system's status. Does it have the latest security updates? Is it configured correctly? Are the drives being monitored? Is it using your newest automation scripts, or old ones? These are all important questions, and you should be able to easily determine the answers if your automation system is implemented properly.

In many cases, detecting problems is a great step forward in your automation process. But how about automatically fixing problems? This too can be a powerful technique. If systems fix their own problems, you will get more full nights of sleep. But if your auto-repair methods are overzealous, you might end up causing more problems than you solve. We will definitely explore self-repair whenever appropriate.

An administrator always has to consider security. With every solution you implement, you must be certain you are not introducing any new security issues. Ideally, you want to create solutions that minimize or even eliminate existing security concerns. For example, you might find it convenient to set up Secure Shell (SSH) so that it uses private keys without a passphrase, but doing so usually opens up serious security holes.

There will always be people who follow in your footsteps. If you ask them, the most important component of your work is good documentation. We already mentioned that in many cases automation techniques provide automatic documentation. You should take full advantage of this easy documentation whenever possible. Consider, as an example, a web server under your control. You can manually configure the web server and document the process for yourself and others in the future, or you can write a script to configure the web server for you. With a script, you can't neglect anything—if you forget to do something, the web server does not run properly.

As obvious as it might sound, it is important to test out your automation before you deploy it on production servers. One or more staging machines are a must. We will discuss techniques for propagating code across machines and explain how you can use these techniques for pushing code to your staging server(s).

Whenever you automate a task, you must consider dependencies. If you automated the installation of software updates and Apache is automatically upgraded on your systems, that's great. But if the configuration files are replaced in the process, will they be regenerated automatically? You need to ask yourself these kinds of questions when you automate a task.

What do you do about these dependencies? They should be your next project. If you can automatically upgrade but can't automatically configure Apache, you might want to address that task next. Even if you have already automated this task, you need to make sure the automation event is triggered after the software is updated. You might also need to update a binary checksum database or services on your systems. Whether or not these tasks are automated, you need to be sure they will not be forgotten.