SAGE

THE SYSTEM ADMINISTRATORS GUILD

ISBN 1-931971-16-1

11 Mark C. Langston Documentation Writing for System Administrators SAGE

11 | *Short Topics in*
Systems Administration

Rik Farrow, Series Editor

Documentation Writing for System Administrators

Mark C. Langston

SAGE

THE SYSTEM ADMINISTRATORS GUILD

ISBN 1-931971-16-1

To purchase additional copies and for membership information, contact:

The USENIX Association
2560 Ninth Street, Suite 215
Berkeley, CA USA 94710
orders@sage.org
http://www.sage.org/

First Printing 2003

Printed in the United States of America on 50% recycled paper,
10–15% post-consumer waste.

Contents

Foreword

Documentation is a critical skill for system administrators. I really do not think that it generally comes into consideration when senior sysadmins get hired or when they interview prospective junior sysadmins. But after having read Mark Langston's excellent booklet, I find myself convinced that the ability to document well and the habit of thorough documentation should be considered essential qualifications for any sysadmin position.

I once asked someone in my group to resign, because he refused to document his work. This young man was brilliant, a genius, as well as a hard worker. Yet he believed that the solutions he came up with were his own and not to be shared, even though he was working as part of a team in a support group. The young man did resign. Although I regretted losing such a talented person, I felt relieved that I no longer had someone on our team who hoarded everything he learned.

Having worked as a solitary system administrator, I can confess to having done a poor job of documentation. I could argue that the place where I worked was not large, and that what I was doing appeared so obvious that it did not require documentation. But the truth was that it was obvious only to me. Tasks such as getting an early version of AIX or Apollo's DomainIX participating in NFS file sharing really deserved proper documentation. No one should have to go through those experiences again. Once was surely enough.

Mark Langston provides clear, friendly, and helpful guidance, not only in the techniques of writing documentation, but also in building the habit of good documentation. Even though I now work alone, I have begun documenting the changes I make in the systems I manage, because life really is easier when I can refer to good notes that remind me of the clever, or necessary, things I have done.

Good documentation leaves you, the practicing sysadmin, with more time to devote to really interesting tasks or projects. I heartily recommend that you read this booklet and learn to practice the art of documentation.

Rik Farrow

Introduction (a.k.a. "Documentation: Why?")

"An hour spent documenting at 2 p.m. may prevent an urgent call at 2 a.m."

Documentation is ubiquitous in our field. It seems to expand until it fills every box, binder, shelf, and flat surface. We're awash in READMEs, HOWTOs, and FAQs. Thanks to the Internet, we've got access to bug-tracking databases, discussion groups, and the Web sites of most vendors and standards bodies. It would seem we have a handle on the whole documentation thing.

Or do we?

Having so much information so readily available lulls us into a false sense of security. We know that within a few moments we can locate a document—either digital or dead-tree—that will tell us all we need to know about a given topic or procedure. We're confident in our ability to apply this information to the task at hand. We're comfortable in the knowledge that, with enough research and some quick thinking, we can solve just about any problem.

There are a few flaws in this train of thought.

- You need to document your specific environment and needs. Vendor-supplied documentation tends to address generic situations, rather than specific implementations. Even when vendor-supplied information is specific, it rarely describes your problem perfectly.
- Discussion groups, bug databases, and similar resources offer incomplete, contradictory, and often incorrect information. In many cases, not enough context is supplied to determine the applicability of the information to your situation.
- Documentation most often explains how to do things right, rather than what to do when things go wrong.
- The most critical information remains undocumented and may reside with one or a small number of individuals, who become single points of failure.
- The documentation you find may not mesh well with your own processes and procedures. Identical tasks may be completed in different manners by different individuals. You need documentation that directly fits your particular needs, rather than having to translate from another person's way of doing things to your own each time the document is used.
- External references, particularly those that aren't printed, may be unavailable when you need them.

- The information may be available, but in a form that's ineffective, poorly organized, incomplete, or inappropriate for your needs.

My goal is to provide you with solutions for these and similar problems. By the time you reach the end of this booklet, you should understand the need to incorporate the practice of documentation as part of your daily routine, and you should be armed with the knowledge necessary to produce effective documentation that will become an integral part of your organization's approach to IT.

But Nobody Reads Documentation!

You see documentation everywhere. You've probably got (literally) tons of it yourself. Chances are good that you're the type who installs first and reads the fine manual (RTFM) later. Our professional culture is rife with jokes about the misuse and disuse of documentation. Why is that?

Perhaps because much of the documentation we must deal with on a daily basis is nonexistent, poorly written, poorly organized, and/or poorly presented. In other words, it's not useful, accessible, accurate, and/or available (our friends, the Four Keys to Good Documentation).

I've heard sysadmins—particularly junior sysadmins—complain that manpages are neither useful nor accessible. I've watched them spend many minutes puzzling over a manpage looking for the answer to a question, only to come away from the experience empty-handed and somewhat bitter. Does this mean manpages are poor documentation? Hardly! But it means that they were the wrong tool for the job, from two perspectives. First, the person may not have known where to look for a better solution or may not have had other resources available. Second, the manpages were written for a particular audience, which did not include the junior admin.

I've seen users shun ticketing and problem-tracking tools in favor of a quick email message to a sysadmin, a phone call, or the dreaded "line of users at your door" approach. Is it because ticketing systems are poorly designed, inaccurate, or otherwise useless? Not a bit. Many times, it's because the interface is confusing and/or time-consuming. People tend toward the most direct solution to a problem. When your documentation seems more of a hindrance than a help, people will go out of their way to avoid using it.

You can escape these and other problems by following five simple rules when setting out to write your documentation:

1. Know your audience.
2. Know your content.
3. Know your requirements.
4. Make it a habit.
5. Advertise.

1. Know your audience. I can't stress this point too strongly. Know who will be reading your documentation. Know what they expect from it and how they intend to use it. Know what they need to get out of it.

2. Know your content. Know what type of information you're going to present and what level of detail is appropriate for your audience. Proficiency in the subject matter is extremely useful. If you find you know little more about the subject than your intended audience does, it may be in everyone's best interest for you to delegate the work to someone with a firmer grasp of the material: the better the writer understands the subject, the better the chances of avoiding jargon and overly complex explanations.

Of course, the opposite can hold true as well—subject-matter experts are sometimes prone to forget their audience, going into complex, jargon-heavy explanations that quickly become far too esoteric for the audience, which goes to show why knowing your audience is such a crucial factor in writing good documentation. If you know your audience and identify a subject-matter expert who's good at describing things to the people in that target audience, your documentation project is likely to prosper.

3. Know your requirements. Before you sit down to start writing, or design your delivery format, or (worst-case scenario) start telling everyone in your organization about your fabulous new documentation resource, you should define your requirements clearly. What do you need to put into the effort? What does your audience need to get out of it? How do they need to get it? What are your constraints on delivering it? Once you've answered these questions, you'll have a clearer picture of what it is you need to document, how it should be documented, and how it should be delivered.

4. Make it a habit. All documentation should be viewed as "living" documentation. Once it's written, it shouldn't be tossed aside and forgotten. I can promise you that sometime, somewhere, someone will find it and try to use your documentation. If you haven't been maintaining it, it will be worse than useless—it may be dangerous (ever rewire a three-phase circuit and forget to tell anyone?). Any failure to maintain your documentation will be quickly discovered by your audience, who will henceforth ignore your efforts.

For example, let's say you've got an online collection of frequently asked questions (FAQs) for your group. You decide to make this collection available to the rest of the organization, as a way of minimizing your support calls. That's certainly an admirable goal! But let's say 5–10% of the questions and answers are out of date, incorrect, and/or poorly worded. Your users won't ignore those bits of information; they'll interpret that sort of inaccuracy as reflecting badly on the entire documentation system. A system like the one I just described would quickly fall by the wayside, rarely to be used by its target audience. Your audience expects and deserves accuracy and completeness, and they'll quickly turn on you if they don't get it. Documentation is perceived as authoritative. To insure that it remains so, you must work to make the maintenance of your documentation a regular habit. If you've documented a certain system, or component, or area of knowledge in your organization, you should always remember to update the documentation as soon as a change is made to whatever's been documented. In fact, you shouldn't consider your work complete until you've not only finished the technical part of the job, but finished documenting the work as well.

5. Advertise. So you've been hard at work, analyzing your needs and those of your target audience, and you've constructed a miracle of modern documentation, the very model of successful sysadmin documentation effort. Why are people still bugging you with questions about things that are clearly explained in your documentation? It could be as simple as this: they don't know the material exists. Once you've finished your documentation and made it available, you must tell people about it, particularly if it's documentation that's not taking up a significant portion of their desk (the satisfying thud when it arrives is usually a dead giveaway for that type of documentation). If people don't know information is available, or how to access and use it, they can't take advantage of it. So tell them! Let them know that this new resource exists. When you're asked a question that is covered in the documentation, refer them to it rather than answering the question yourself. Help your audience help themselves. Excellent documentation nobody knows about is just as useless as poorly written, poorly presented, or nonexistent documentation.

Preview

The chapters that follow will introduce you to several types of information you may wish to document (documentation content), describe various tools and techniques for documentation (documentation format), provide you with a procedural framework to get started with documentation in your organization (documentation methodology), and offer a series of examples meant to illustrate the utility of certain types of documentation.

Throughout the booklet, the terms *system administrator* and *sysadmin* are used interchangeably. The abbreviation IT, standing for "Information Technology," is meant to encompass all areas of a sysadmin's responsibilities.

Presuppositions

I assume you're an active system administrator with a need for or interest in documentation as part of your job, and/or are responsible for documentation of the system administrator role in your organization. I also assume that you'll use this booklet to guide your documentation efforts.

1. Documentation Content

Once you've made the decision to document, or to revise your existing documentation, the next step is to define the scope of the task: What, exactly, should you document? The answer to this question will vary from situation to situation, but the two basic types of documentation are (a) procedural and (b) informational. Procedural documentation captures the essence of an action or series of actions. It offers a set of instructions, telling the reader how to achieve some goal (e.g., how to build a stable, functioning HTTP daemon from source code). Informational documentation records the state of an object or collectivity, for example, system configurations, network diagrams, software inventories, license collections, service-level agreements, or project requirements.

This chapter discusses these two major categories of information, the importance of knowing your audience, and the four keys to good documentation. We reserve issues concerning documentation format for the next chapter.

Procedural Documentation

As system administrators, we use procedural documentation almost daily. We'd be lost without it, because so much of our work requires us to use unfamiliar tools to perform unfamiliar work. If procedural documentation is so abundant, why would we want more of it?

The following example of procedural documentation is taken from the INSTALL file of OpenSSL-0.9.6.h:

1a. Configure OpenSSL for your operating system automatically:

```
$ ./config [options]
```

This guesses at your operating system (and compiler, if necessary) and configures OpenSSL based on this guess. Run ./config -t to see if it guessed correctly. If you want to use a different compiler, you are cross-compiling for another platform, or the ./config guess was wrong for other reasons, go to step 1b. Otherwise go to step 2. On some systems, you can include debugging information as follows:

```
$ ./config -d [options]
```

1b. Configure OpenSSL for your operating system manually. OpenSSL knows about a range of different operating systems, hardware, and compiler combinations. To see the ones it knows about, run:

```
$ ./Configure
```

Pick a suitable name from the list that matches your system. For most operating systems there is a choice between using "cc" and "gcc". When you have identified your system (and, if necessary, compiler) use this name as the argument to ./Configure. For example, a "linux-elf" user would run:

```
$ ./Configure linux-elf [options]
```

If your system is not available, you will have to edit the Configure program and add the correct configuration for your system. The generic configurations "cc" and "gcc" should usually work on 32 bit systems. Configure creates the file Makefile.ssl from Makefile.org and defines various macros in crypto/opensslconf.h (generated from crypto/opensslconf.h.in).

2. Build OpenSSL by running:

```
$ make
```

As you can see, this documentation focuses on the stepwise completion of a task. In this case, the task is the compilation of OpenSSL. Note also that the documentation tries to address various specific situations (e.g., which C compiler is available, the compilation platform, the variant of config in use), but at the end of the day, it's a generic document. It focuses on the task, leaving it to you to know when and how to modify the process.

The documentation we encounter daily is often written by people outside our environment, to address general cases. It may describe how to get from point A to point B, whereas we may need to arrive at point Q instead. This gap between the general instructions and our configuration makes the process an excellent candidate for in-house procedural documentation.

Procedural documentation can be extremely useful in problem-solving and troubleshooting. Without such documentation, each error must be corrected either from memory or from information gleaned from outside sources such as books, search engines, and Web sites. Writing documentation that covers the errors likely to occur in your own environment greatly narrows this search space and increases the likelihood that the necessary information will be available, accurate, and applicable.

Let's say you or your team will be building OpenSSL over and over again in your environment. Rather than remember when and how to deviate from the standard install process, you develop a site-specific OpenSSL installation procedure. The previous example may be modified to look a bit more like this:

1. On our Solaris systems, make sure that /usr/ccs/bin and /usr/local/bin are in your path:

```
$ echo $PATH
```

If they are not, include them:

```
$ export PATH=/usr/local/bin:/usr/ccs/bin:$PATH
```

2. Configure OpenSSL automatically:

```
$ ./config
```

The configuration process should complete with no errors. If errors were encountered, save the full text of the error(s) and email the text to a team lead, asking for assistance.

3. Build OpenSSL by running:

```
$ make
```

The build process should complete with no errors. If errors were encountered, save the full text of the error(s) and email the text to a team lead, asking for assistance.

This revised text has the same goal as the original, but it takes into account your specific systems, organizational structure, and policies. It's designed to insure that the process is problem-free and adapted to your environment.

The method you use for your documentation will be covered later. What is important here is to realize that you should be documenting, or updating your documentation, constantly. You should approach each day with the assumption that anything not written down will be forgotten. If you handle a situation you may encounter again and you don't document your actions, remember that you'll be forced to reinvent the wheel the next time that situation crops up. I worked with a professor as an undergrad who used to make notes religiously. He called his notebooks his "external memory." It was no coincidence that he was a cognitive psychologist with an interest in how human memory works.

In a fitting tribute to human nature, his habits failed to rub off on me soon enough. Sometime later, I was working as a junior admin in an environment that required quite a bit of Sun Online DiskSuite (ODS) work, and I'd had only limited experience with ODS. The result? I spent 96 hours straight over a long holiday weekend rebuilding a system from recovered data pulled raw directly from disk sectors. The system's configuration hadn't been documented, and because it was a test system, company policy stated that it would not be backed up. Nonetheless, when the disk crashed, I had several DBAs and two managers calling me frantically every hour, demanding to know when their "critical" system would be back online.

If I'd had adequate documentation of the system configuration (software installed, license keys, partition tables, RAID configurations, network configuration, and so forth), the work would have taken an afternoon instead of four 24-hour days. I only needed to live this nightmare once; from that point forward, I was a strong proponent of the need for documentation in system administration.

Producing documentation as you work has several other benefits. You're creating a record of your work that will come in handy down the road, when you need to troubleshoot; you're producing a tangible product of your labor, which can be quite useful when you need to discuss your own or your group's work with management; and you're giving yourself small, logical breaks in your work to stop and review what you've just accomplished, which can be an invaluable aid in catching small mistakes before they become big problems.

If you're being shown a procedure by someone else, documenting the information given can eliminate the need to have the same person repeat the information to you or to others who may need it later. It also helps both of you to clarify your understanding before you try to perform the task yourself.

Unsurprisingly, many organizations lack any formal policy for documentation. Don't let such a void dissuade you. The absence of a policy is no excuse to avoid documenting. If your organization has no explicit guidelines regarding what and how to document, take the opportunity to shine. Start documenting immediately, and make it a part of your daily routine. People may look at you a bit oddly for a while, but the first time your documentation averts a catastrophe, they'll appreciate your efforts. At that point, work with your management to establish guidelines for documentation.

What kinds of documents should you create? There are many types of procedural documentation in use today. The type and amount of procedural documentation you need must be determined by your specific situation. For example, working as the sole admin for a small company, you may only want to script the installation and common maintenance tasks you perform, perhaps with a bit of *post hoc* editing for clarity and any necessary annotations. As the manager of a team of sysadmins, you may want common software installation tasks, hardware build tasks, and operations tasks documented so that each member of the team will adhere to company policies and contractual obligations. As a contractor delivering a service to a company, you may want to document your work, as well as any actions your customer must take, in considerable detail.

Whatever your specific needs, procedural documentation allows you to address multiple goals:

- You standardize departmental behavior.
- You provide a means to automate and optimize tasks.
- You facilitate troubleshooting.
- You assist others who must accomplish the same or similar tasks in the future.

You Standardize Departmental Behavior

Procedural documentation provides a reference point from which we may work, eliminating the need to remember or re-create a procedure. This makes the work of an individual sysadmin more consistent. When used by a group of sysadmins, procedural documentation becomes a standard from which to work, insuring that the work of the group is consistent. This standardization frees you from worrying about the daily routine, allowing you to spend mental cycles on more interesting problems. Admittedly, it may prevent you or your co-workers from creating some of those "more interesting problems," but that's a sacrifice I think we're all willing to make.

You Provide a Means to Automate and Optimize Tasks

When you work from procedural documentation, you're behaving much like a computer: acting on a set of instructions. Procedural documentation can be a wonderful indicator of the opportunity to automate certain tasks. For example, the document may contain several consecutive steps that are nothing but a series of commands and conditionals. These can be automated, thus increasing efficiency by decreasing the time necessary to complete the procedure. Documenting the steps necessary to solve a problem can also reveal unnecessary or inefficient actions. Analyzing the steps taken to solve a problem may lead you to better solutions. Analyzing the context in which these procedures must be applied may lead you to more effective procedures for your particular circumstance.

You Facilitate Troubleshooting

If you've ever written poorly documented code and returned to examine the code 6–12 months later, you've almost certainly wondered, "Who wrote this code? What were they thinking here? Why did they do $FOO in this particular manner?" This experience is not unique to programming; sysadmins face similar problems regularly. On occasion, I log into unfamiliar systems. Without documentation, these systems are an enigma. I have no information about current configuration, problem history, acceptable procedures for certain tasks, or other guidelines. With documentation, it's relatively easy to find and fix a problem in a manner that won't cause further problems down the road.

You Assist Others

Procedural documentation is a timesaver when applied to situations that are identical, but it can also be extremely useful in situations that differ somewhat from the original. When you are faced with an odd error or an unfamiliar task, referring to documentation that addresses a similar goal, tool, or system component can be very useful. It provides a starting point for problem-solving, and it may provide clues to the solution (which should then be documented!).

These goals are all desirable to a certain degree, even in single-sysadmin environments. Without procedural documentation, you're stuck reinventing the wheel at every turn, and you may find that the external documentation you used last time has changed, disappeared, or aged to the point of irrelevancy in the interim. Documenting your own procedures gives you control over the information you need to do your job, as well as the ability to step back to examine the job you do.

Informational Documentation

"Informational documentation" means nonprocedural documentation, such as system configurations, network diagrams, software inventories, license collections, service-level agreements, or project requirements. This type of information is sometimes called "tacit knowledge" or "static content." However, as any sysadmin can tell you, it's anything but static; it changes regularly, and some seem to be in a state of constant flux. Calling such documentation static gives the wrong impression. These are living documents, to be used and revised and maintained, rather than locked in a filing cabinet and forgotten.

The following is an example of informational documentation:

```
gateway.example.com    10.1.1.1
mx1.example.com        10.1.1.2
mx2.example.com        10.1.1.3
netapp.example.com     10.1.1.10
www.example.com        10.10.1.100
winxp350.example.com   10.2.1.50
```

Sure, it's just a list of hostnames and IPs. With a bit of massaging, it could easily act as an /etc/hosts file or a zone file for DNS. Still, as it stands it's nonprocedural information. Chances are you've got something similar lying around, either printed on the fly for reference while doing some work away from the keyboard, or stored as a result of your organization's documentation policy.

When writing informational documentation, it's important to remain focused on a clearer goal than "Document anything and everything." If you're unsure whether some information should be documented, refer back to your reasons for producing the documentation. For example, if you are documenting system configurations, you may have any of several goals: capturing the information necessary to rebuild the systems in case of catastrophic failure; creating a hardware and/or software inventory; or creating a build book to guide your junior admins when configuring new systems. Each of these goals requires certain types of information, and that information could be captured, organized, and presented in various ways, some more suited to certain goals than others. Simply documenting everything and hoping to address multiple goals simultaneously is a recipe for disaster.

It's also important to keep the term "informational" in mind as a rule of thumb for deciding what and how to document: if the data changes rapidly, the documentation won't remain informational for long. To at least some extent, the content should dictate the medium for the documentation. Using the previous example of hostname/IP mappings, you wouldn't want to keep paper documentation of this information if it changes frequently. You'd be forced to create new paper copies every time a change is made, or else you'd have to resign yourself to the fact that the paper documentation would almost always be wrong. Incorrect documentation isn't very useful. Information that changes frequently is more easily maintained in an electronic form that allows printing of hard copy on demand. With this approach, changes to the documentation could be automated, minimizing the manhours necessary to maintain it and increasing its reliability.

2. Writing Good Documentation

Know Your Audience

Before you sit down to document everything you can get your hands on, you need to ask yourself two important questions: Why are you documenting, and for whom? The answer may be, "I've got to clarify this installation procedure so that our junior admins start turning out consistent system builds." It may be, "Everyone in the company has to get it through their heads that a few simple actions would greatly enhance security around here." It may even be, "I need to make sure I add extensive comments to this code, or I won't recognize it six months from now."

Whatever your answers are, they will serve as guidelines and reminders as you write. If your audience is the entire company, you know you should keep the technobabble to a minimum. If you're writing for the junior admins on your team, you can safely use technical terms and refer to procedures and equipment only you and your team know about.

I like to define my audience in writing when beginning a new document, and I keep the definition at the head of the document until I've finished. Sometimes it stays as part of the final document—a convenient reminder to anyone picking up the document that it was intended for certain people with certain knowledge, which they may or may not possess. This has saved me several times when more junior or senior people have read through a document not intended for them, then come to me with questions or directives regarding its contents.

If you don't keep your audience in mind at all times, it's easy to drift away from your primary goals, writing to a different audience. This tends to be particularly true for technical people unaccustomed to writing nontechnical documents, or to writing technical documents for nontechnical readers. Many is the time I've picked up an internal document meant to explain a technical process to a nontechnical audience (e.g., remote login and authentication for personnel traveling on business), only to discover a technical document that's comprehensible only to a person with knowledge on a par with that of the author. It's not the author's fault, necessarily; he didn't intentionally obfuscate the process. He's just not used to communicating such information in writing; he's used to talking and fielding questions. Even the best tech support person can falter when writing documentation, because she's not interacting with her audience, clarifying assumptions and adjusting her level of detail.

Assumptions made while writing are dangerous, leading to any number of documentation problems. What you include and, more important, what you do not include in your documentation are directly related to what you assume your audience knows. If you explain that the first step in a process is to "log into the nearest POP," you've made a bucketload of assumptions about what your reader knows. You assume that they know what you mean by "log in," as well as what tools are necessary to do so. You assume that they know what "POP" means, and that they are capable of understanding what "nearest" means in context, as well as the method for determining which "POP" is, in fact, "nearest." If your audience knows all this, fine. If they don't, you're going to be fielding quite a few phone calls and email messages shortly after the document is distributed.

The Keys to Good Documentation

I've covered two major categories of information that you'll find in documentation, procedural and informational. I've provided you with the idea of an audience, and I've explained why keeping your audience in mind is paramount for successful documentation. Now I need to introduce a few more concepts: the components of good documentation, and an answer to the question, "How much is enough?"

The Components of Good Documentation

Good documentation is:

- Useful
- Accessible
- Accurate
- Available

A truly worthwhile document has all four of these characteristics.

Good documentation is *useful:* A good document serves a purpose and addresses a need. It is adequate for its purpose, containing neither too little nor too much information, and its readers are able to apply its contents to real-world situations.

Good documentation is *accessible:* A good document does not obfuscate its subject matter. It communicates clearly with its intended audience. Its purpose and audience are obvious to the reader.

Good documentation is *accurate:* A good document is factually correct and complete. It is kept current as the subject matter changes, or deprecated as the subject matter ages.

Good documentation is *available:* A good document is there when you need it. The best document in the world is useless if you can't get it when you need it most.

A document that lacks one or more of these characteristics is not only a poor document in its own right, it's potentially dangerous. Imagine, if you will, that the wiring schematic for your server room doubles as your network diagram, server inventory, and list of admin responsibilities. This format may be quite useful for the IT department head, as it provides a quick reference to often-used information. It probably highlights the most important power feeds, outlets, cabling runs, wallplates, systems, and so forth.

If that document is handed to your electricians as a wiring schematic, though, they'll wonder what you expect them to do with it. Much of the information in the document is irrelevant to their tasks and may well confuse them. The electricians don't need a quick reference to all aspects of your server room. They need a detailed, accurate, and complete wiring diagram, without any extraneous information.

If the document doesn't look like a wiring schematic to people who should know what one looks like, you're asking for trouble. Nonstandard symbols, unclear diagrams, extraneous information, and "in-house" terminology are all signs that the document will prove inaccessible to the people who are expected to use it. As the user, do you really want to guess whether that 240V, 3-phase outlet has a green or a black ground wire?

If the document was drawn up 18 months ago, prior to dozens of undocumented wiring revisions, do you really want an electrician working from it? Of course not; it's inaccurate.

If the document resided only on your intranet Web site—the intranet Web site hosted on the server located in the server room containing the wiring that needs emergency repair because the entire building's circuits just blew and you've got an electrical fire under your firewall—it's not very available now, is it?

Writing the documentation is only the first step. Documentation must be reviewed regularly to insure that it continues to meet each of these criteria. The first such review should occur immediately after the document has been written, before its publication and distribution. Subsequent reviews should occur on a schedule that matches the rate at which these criteria may change in your organization, given the document's subject matter. Reviews should also occur whenever changes are made that directly impact the document's content.

You may want to consider establishing some means by which document users can submit bug reports for your documents. Perhaps you could include a feedback form on each page of your Web-based documentation. You might go so far as to set up a bug-tracking or ticketing system people can use to submit feedback on documents as they're used. With such tools in place, the document review process will be more efficient, allowing the reviewers to more accurately incorporate changes derived from immediate, real-world feedback during document use.

How Much Documentation Is Enough?

How much documentation is enough? The answer is, of course, "As much as you need." I've seen organizations buried in documentation, and I've seen organizations with no documentation whatsoever. Both situations are difficult to work in. The amount of documentation has to match the needs of those in the organization. I've seen groups who get on swimmingly with roomsful of documentation, and some with just a few pages available, which contain all they need.

So how do you know when to stop? For procedural documentation, the answer's similar to that for the question, "When is it the appropriate time to automate a task?" If you find yourself or others performing some process repeatedly, or you believe with some certainty that the task will need to be performed repeatedly in the future by yourself or others, it should probably be documented. “Okay,” you say. “What if I automate this procedure I’m documenting? Should I still document the procedure?” Certainly! Not only will the document come in handy when the program bombs, or it’s accidentally deleted and the backups are corrupt, or you’ve forgotten the filename and/or location, but it’s also a good explanation of the program used to automate the task. Self-documenting code is an admirable goal, but failing that, solid code with solid documentation explaining it is your best bet.

“What if I’m the only person who’ll ever perform the procedure? Should I still document it?” Again the answer is a resounding “Yes!” Ever look at code you wrote 6 months ago? 12 months ago? It’s as though a stranger wrote it. The same holds true for procedure and process: if you don’t document it, you’re going to wind up forgetting it. Say you’re asked to perform a fairly complex task, such as “Build an Apache Web server with a Postgres back end, Jakarta, and SSL on OpenBSD.” (Note to future readers: At the time of writing, this was no trivial task.) Even if you only perform the task once, the documentation of the process will be invaluable down the road when you need to troubleshoot the installation or explain to someone else what work was done.

Informational documentation is a more complex issue. Here, the answer to the question “How much?” should be directly related to the answer to the question “For what purpose?” Whereas procedural documentation often pays for itself the first time you need it, informational documentation may sit on a shelf and rot. Worse, if you institute a good documentation policy incorporating document review, you’ll be creating quite a lot of future work for yourself or others, so it’s important to be goal-driven when deciding what informational documentation to create.

For sysadmins, a network diagram is often a must-have, and a very good example of necessary informational documentation. It’s not created just to impress people (though you’ll occasionally see network diagrams in public relations and marketing material); it’s a useful troubleshooting and design tool. Similarly, a list of all equipment broken down in various ways is an example of good informational documentation (e.g., by wallplate, patch, and switch port, OS, vendor, function, location). Lists of license keys, installed software, assigned IPs and IP ranges are all good examples. Better yet, they all lend themselves to automated generation and maintenance. Some other kinds of good informational documentation aren’t directly tied to your systems and networks. For instance, bills of lading and sale for your equipment are good to keep around, as are service contracts and organizational policy documents.

I’ve been pulling examples from experience, but a much more effective approach to answering the questions “What and how much?” is to examine your group’s functions and purpose. Assignments such as disaster prevention and recovery, data retention, security, and reliable packet transit are all commonly included in the lists of services a sysadmin or group of sysadmins is expected to provide. Working from this list of services, it’s easier to determine what documentation you’ll need. For disaster recovery, you’ll want your backup policy, your disaster recovery policy, the relevant configuration files for any system that may need to be rebuilt (in case the backed-up data is unavailable), and so on. In fact, you'll probably want this information stored in multiple forms, both on- and off-site, with at least one paper copy (more on this in the next chapter).

In short, you'll want documentation that addresses the goals you're expected to achieve. This is true not only for informational documentation, but for procedural documentation as well. Using this approach with procedural documentation, you may well uncover tasks that should be documented but aren't, because you haven't encountered the situation yet. In areas like disaster recovery, this kind of discovery can be a blessing in disguise.

In Chapter 5 we'll examine the question of what and how much in more detail. Right now, I'd like to discuss different types of document formats you can use.

3. Documentation Format

We've explored the various types of documentation, developed some understanding of what makes good documentation, and arrived at a few methods for deciding how much documentation you'll need. Now it's time to investigate various approaches to documentation and see how you can make documentation part of your daily routine.

The Habit of Documenting

One of the biggest problems with documentation is that it seems like extra work. You spend hours getting a particularly tricky problem solved. Instead of celebrating when you're done, you are then expected to sit down and record every step you took. And that's only if you remember to do it right that minute. Often, you'll forget and end up trying to recall the details of the procedure days or weeks later, when you've been reminded that someone else needs the documentation you were supposed to write. What can you do?

Documentation, much like any other necessary daily task, needs to be routinized—you need to make a habit of it. Made a change to a system or application configuration? Document it right then and there. Moved or renamed an important file? Make a note of it. About to perform a complex task for the first time? That's the perfect opportunity to document, because you're often researching the procedure as you go. As you research, capture what you discover in the documentation. It may appear at first that you'll be adding a lot of extra time to any task, but in reality that time would need to be spent on documentation eventually; why not make it part and parcel of the work itself? That way, when you're done with the work, you're done with the documentation, too.

Do you have to jot it all down by hand as you go? Certainly not, though hardcopy documentation has its place. The rest of this section is devoted to exploring the various ways you can capture the information you need for your documentation. Each has its strengths, and each has its weaknesses. It's up to you to determine which method or methods are right for your situation.

In this chapter we'll cover three ways to capture and organize the information you wish to document: paper, Web-based, and flat file. Approaches such as database-derived documentation will be covered in the section on Web-based documentation, and I'll touch on code annotation in the section on flat files. Then I'll briefly describe a few tools you may find useful.

I've also got a bit of a mantra which you're required to repeat to yourself while reading this section:

Format Influences Content. Content Influences Format.

This mantra encapsulates the two most important points to take away from this section. The method you use to capture, organize, and present your content will affect the content itself, by placing constraints either on the presentation of the information or on its use. Similarly, the information you need to present should determine the format you'll choose. Ideally, you'll want to choose a format that maximizes each of the four keys to good documentation—that it's useful, accessible, accurate, and available—discussed in the previous section. Since we don't live in an ideal world, you should determine which subset of the keys is most important for the task at hand, then select the format that maximizes those keys.

The discussion of each of the three major formats is preceded by a list of pros and cons and is followed by a rating from 1 to 3 on each of the four keys. A summary table is provided at the end of this section for quick reference.

Paper Documentation

Pros and Cons

Pros:

- tangible
- portable
- doesn't require power
- long-lasting
- easy to annotate

Cons:

- easily damaged or destroyed
- changes propagate slowly
- costly maintenance
- storage becomes an issue
- low information density

Paper, or hardcopy, documentation is often what people think of when they hear the term "documentation": row after row, shelf after shelf of binders and manuals, drawer after drawer of folders in file cabinets. Paper documentation is relatively cheap to produce, portable, works under most conditions, doesn't require a power source, is easy to annotate, and, with proper care, degrades very slowly. Even I, an avowed gadget freak, carry a pocket-sized notebook with me everywhere, because you never know when your PDA's batteries will run down, the OS will crash, or you'll accidentally hit the wrong button and delete the wrong document.

However, paper documentation is difficult and expensive to distribute, particularly across long distances. To use paper documentation, people must have physical access to it. If you make changes to the documentation, you'll need to distribute the changed pages and make sure users insert them in the appropriate locations, or distribute complete new documents. The old pages or documents must be removed from circulation, and often must be destroyed or recycled. This cycle can become expensive and labor-intensive very quickly. It's also a relatively slow process—changes made to a document may take days or weeks to filter down to all copies, and there's the distinct possibility that some copies of the old documentation may remain outdated.

Paper documentation is fragile: an ill-placed cup of coffee or tea can damage or destroy paper-based documentation. It's also combustible (the paper, not the coffee, unless you happen to drink particularly strong coffee). Its bulk makes storage and transport of large quantities both difficult and expensive.

Finally, its information density is rather low in comparison to digital alternatives: the amount of physical material necessary to store x bytes of information on paper is much higher than that needed to store an identical amount of information digitally.

How Does Paper Rate?

Let's evaluate paper documentation against the four keys of good documentation, as discussed in the previous chapter. Paper documentation can be useful, but its usefulness is largely a function of its content. To the extent that the medium shapes the message, paper documentation tends toward the extremes of usefulness: far too much or far too little information. It's rare to find hardcopy that covers just what you need, no less and no more. This may be related to the relative difficulties associated with creating and maintaining paper documentation. Since it's not usually feasible to create several thousand new booklets every time a single word, table, or graphic needs to be changed, the creators opt for over-specificity, either in subject matter (thus providing too little information to the average user) or in scope (thus providing too much).

Paper documentation, though sometimes cumbersome, is the medium we humans are most accustomed to using in this day and age. Many of us were taught how to produce both fact and fiction on paper as part of the educational process. Society has seen fit to provide us with a mental grab-bag of tools for producing paper documentation and to insure the documentation we produce is accessible. Some people are better at it than others, but more often than not a random stranger could explain to you how to structure information into paragraphs, how to outline a long document, and how to present a thesis. Much of our daily lives includes applying these tools to varying degrees, particularly now that email has become a fundamental means of communication in both our professional and our social lives.

Accuracy is problematic for paper documentation. Sure, you can go to great lengths to insure a document is accurate when it's produced, but it's labor-intensive to keep hardcopy accurate after its original publication. A single change may require the destruction of all previous copies and the production and distribution of replacements.

Paper documentation excels in *availability*. From pocket-sized notebooks and Post-Its to binders and books, hardcopy is something you can touch, feel, point to, and carry around. It doesn't require electricity to use; there are no passwords to remember before you can open it. Even in candlelight while trying to restore power to your server room, paper documentation continues to function.

Paper documentation rated according to the four keys of good documentation (out of ***):

Useful: *
Accessible: **
Accurate: *
Available: ***

Web-based Documentation

Pros and Cons

Pros:

- searchable
- intangible
- high information density
- selective printing
- low maintenance cost
- instantaneous change propagation
- easy to back up
- revision history
- dynamic and interactive
- inherently nonlinear

Cons:

- searchable
- intangible
- inherently nonlinear
- many prerequisites for use

More and more, people are turning to Web-based documentation. It's relatively easy to create, trivial to distribute, and its maintenance can be automated. The support documents for the latest game? On the distributor's Web site. The manual for your shiny new frob? In HTML, on the CD-ROM enclosed in the box (and, frustratingly to some, nowhere else). Frequently, a Web page is also where you'll find your employer's Human Resources documents, financial forms, phonebooks, troubleshooting guides, operating instructions, and policies. It's become the most common place to store network diagrams, software and hardware inventories, and procedural documentation.

Web pages have become so popular for the storage and presentation of information because they offer a sophistication you just can't get from a piece of paper. For example, Web pages are easily searchable. Sure, a book may have a table of contents and an index, but Web pages can have these as well. In addition, Web pages may offer the user the ability to perform a freeform search for a string or strings within the document, or even among a set of documents, and return the results within a second or two, neatly organized and sorted based on various criteria. This is invaluable when you're looking for, say, the cause of or solution to a particular error message. Pull up the Web page, type in a search, and *voilà* ! Unless your book has indexed each error string, you'll spend a bit more time flipping through the chapters, hunting for a section relevant to the error you seek. By the time you find it, the person who used the Web-based search tool may already have fixed what was wrong.

Why carry around twenty pounds of books, when you can have electronic, HTML-ized copies at the press of a button? The information density per square inch of electronic documentation is enormous, compared to the same amount of printed information. These days, those twenty pounds of books can fit on a card the size of a postage stamp and still leave room for another metric ton or two.

When you do need hardcopy, it's only a button-push away. You can print selectively from Web-based documentation, creating paper versions of only the information you absolutely need in a tangible, low-tech form. Why distribute half a dead tree just to disseminate one policy change, when the same information can be put online, there for all to see? If people want a printed copy, they're welcome to generate it. You've just eliminated the need to create one paper copy per employee, along with the task of distributing it, storing it, and getting rid of it when it's thrown away or recycled.

Or have you? I'll touch on that and other looming questions in just a bit. Right now, I want to finish singing the praises of Web-based documentation.

Electronic documentation provides you with maintenance options paper can't offer. For example, it's often possible to automate changes to electronic documentation. The earlier hostname/IP example could easily be generated automatically from a hosts table or DNS zone transfer at regularly scheduled times, rather than maintained by hand. As another example, let's say you set up an online Frequently Asked Questions (FAQ) section for the IT Services department on your company intranet. You and your co-workers could take advantage of the forms capability of HTML to simplify the creation and maintenance of FAQ entries. You could even provide a means by which non-IT personnel could submit questions or requests for clarification, providing both an audit trail for document revision and a level of interactivity between the author and reader that's more difficult to achieve with paper documentation. Versioning tools such as RCS (*http://www.gnu.org/software/rcs/rcs.html*) and CVS (*http://www.cvshome.org/*) may also be used to automate the maintenance of document revision history. There's also a good introduction to using CVS for documentation here: *http://citeseer.nj.nec.com/326583.html.*

Another approach to automated or semi-automated electronic documentation organization, retrieval, and even creation is to use a database to store your information. Databases, including open source projects such as MySQL (*http://www.mysql.com/*) and PostgreSQL (*http://www.postgresql.org*), provide you with a powerful, standard interface for storing, retrieving, and manipulating information. You could decide to use the database on its own, running manual Structured Query Language (SQL) queries against it to store and retrieve the information you want. However, you may find that using a Web-based front end provides you with added convenience and flexibility. Some online documentation tools, such as various implementations of Wiki (which we'll discuss in just a bit) use a database back end. However, SQL isn't a difficult language, and with various database interface modules available for multiple programming and scripting languages, creating a Web-based database front end should not present much challenge to a toolsmith.

Another benefit of using databases for documentation is the ability to create what some refer to as "modular" or "reusable" documentation. This approach allows you to create individual, marked-up sections of information in a relatively context-free manner and then generate larger documents from aggregates of these modules on-the-fly. There's a good book on the subject entitled *Single Sourcing: Building Modular Documentation*, by Kurt Ament (Noyes Publications, 2002). Implementation of these approaches is beyond the scope of this document, but if you're at all interested in the subject, I heartily recommend the referenced sources. A quick Google search for the various terms used here should also provide you with some useful preliminary information.

Web-based documentation—for that matter, most electronic documentation—offers the user options they simply don't have with paper documentation. For example, in Web-based documentation it's trivial to highlight a section of text, copy it, and paste it into a new document for revision and annotation. It's also a simple matter to lift command or code examples directly from electronic documentation and implement them immediately. (Helpful hint: When including example commands or code, make sure to mention whether it's functional or should be used with caveats if executed as is!)

Electronic documentation, particularly Web-based documentation, is inherently nonlinear. Both Web-based and hardcopy documentation tends to be organized hierarchically, but Web-based documentation also tends to be navigable in a hierarchical manner, allowing you to quickly access just the information you need. While this is also possible with hardcopy (particularly with an index and/or table of contents), paper documentation also tends to be written with the assumption that you've read the previous material and you intend to read the subsequent material. Web-based documentation tends toward self-contained elements: each link connects to information that makes no assumptions about where you've been or where you're going. This means choosing a page at random from Web-based documentation is more likely to yield coherent results. Of course, these properties are not magically bestowed on your documentation just because you present them via HTML. You must work to adapt your content to the format.

You probably noticed the list of cons for Web-based documentation is very similar to the list of pros. That's no mistake: many strengths of Web-based documentation can easily become weaknesses under certain circumstances.

Take, for example, the fact that the documentation is Web-based. This places a slew of prerequisites on your shoulders:

- You must have access to a computer when you want to use the documentation.
- You must have a local copy of the documentation, or network access and the ability to reach the Web server containing the documentation.
- You must have a browser capable of displaying the information you need.
- You must have electricity to power the computer and any other equipment necessary to meet these other requirements.

We often take such things for granted, but you'll certainly notice should you find yourself in need of the documentation when you don't have them all! What if your server room has lost all power and the server containing the only copy of the documentation is in there? What if you do have access, but the access is in location A and you need to use the documentation in location B, where these requirements aren't met? Web-based documentation can be a powerful tool, but it occasionally requires planning to use it effectively in situations such as these.

Even the ability to search electronic documentation can become a hindrance to its use. I've often used search engines that supplied me with thousands of hits on a very specific query. Conversely, it's not uncommon for an engine to return no results in response to a query. Does this mean the document you need doesn't exist? Not necessarily. It may be in the system somewhere, waiting for you to submit a query that will allow the engine to retrieve it for you. To effectively use search tools, the engine must be appropriate to the type of information it will search, and you must be skilled in submitting queries that give you just the results you need. Otherwise, you'll find yourself spending hours trying to get a single hit, or wading through thousands of pages of irrelevant information to find the one document you need.

The nonlinearity of Web-based documentation can also present a problem. As I mentioned earlier, your information will not magically optimize itself for Web-based presentation; you'll need to do that yourself. If the information is organized poorly for the medium, it'll be difficult to navigate and frustrating to use. Documentation that's hard to use will, unsurprisingly, go unused. This result defeats the purpose of documenting the information. Therefore it's essential to insure that the information you make available is appropriate for the presentation you choose. I don't intend to litter this booklet with references, but the works of Jakob Nielsen and Don Norman are germane to this topic. They bring considerable intellectual talent to bear on the issue of usability and the presentation of information in many forms, including Web-based information. Dr. Nielsen has a Web page full of references to usability at *http://www.useit.com/*, and Dr. Norman has a similarly useful site at *http://www.jnd.org/*.

Web-based documentation presents other barriers to use. To access the documentation at all, you must know where it lives. This is true of hardcopy as well, but with hardcopy you've got a guessable, memorable physical location to which you can point and declare, “It’s over there.” We’ve spent millions of years learning to navigate in three dimensions; physical space makes sense to us at many levels. We’re just learning to navigate information spaces, however. Thus, you’re more likely to forget the URL for a particular piece of documentation, or even for the site that contains it, than you are the location of a folder or book. Most browsers have a bookmarking facility, but relying on it places yet another requirement on you: you must have the URL bookmarked, and the ability to access and use the file containing the bookmarks.

In the list of pros, I point out how easy it is to cut-and-paste content from electronic documentation. While this is indeed a benefit, it is related to a drawback this format has, compared to paper documentation: highlighting of the original document is temporary, and, unless you’ve provided special tools, annotation of the original document is impossible for all those who don’t happen to have access to the machine on which the document is hosted and write permissions for the file containing the text.

In today’s security-conscious world, you may also be faced with certain authentication prerequisites for accessing the information. If you’re using a system already configured for these mechanisms, you shouldn’t have any trouble getting to what you need. But what if the only system available to you is the one you’re working on at the moment, and it’s not yet configured to pass the required authentication? What if that system isn’t, in fact, working? You’ll be stuck without the documentation as surely as if you didn’t have access to a computer.

I promised earlier that I’d explain why Web-based documentation does not equate to “the paperless workspace.” Sure, you’ve eliminated most of the paper-based creation, distribution, maintenance, and deprecation work associated with dead-tree documentation. In its place, however, you’ve created a bit of a monster. Take a look around your office. Go ahead, I’ll wait. Chances are, somewhere in that space there’s a printout of a Web page or other form of electronic documentation. It was probably printed when someone had a specific need for it, and that printout may have been extremely useful at the time. But how useful is it now? Ignoring for a moment the entire issue of archiving electronic data on paper, ask yourself why you have that printout. If you don’t really need it but you still have it, you’ve (perhaps literally) stumbled upon one of the problems brought about by on-demand printing: clutter (or, if you’re a neatnik, organization and storage). Another problem you may not fully realize until you try to use the printout again is similar to one faced by those who use paper documentation: version tracking and control. How do you know that the printout you’re holding gives current information? The Web page may have changed since that page was printed. Given the ease with which changes can be made manually or automatically to online information, this possibility is quite likely. How do you reconcile the two? You could type in the URL from a header or footer of the document, but why go to that trouble when you can just use a bookmark to reach the page in question? In any case, if you have to revisit the page to verify the currency of your paper copy, what’s the point of keeping the paper copy?

Valid reasons for keeping paper copies of online documentation do of course exist. You may wish to keep archival copies of online documentation for presentation purposes or to meet disaster recovery requirements. You may want to keep critical information such as system configuration and service history on paper as well as online, in case you need the information but the system it's stored on isn't available. You may want to have a paper copy of specific tasks handy, close to the system on which they're to be performed. You may want to keep a paper copy of inventories, network diagrams, or group and organizational policies. Just be aware of the problems created by electing to retain, rather than recycle, that documentation once it's served its immediate purpose.

How Does Web-Based Documentation Rate?

How does Web-based documentation measure up in terms of the four keys to good documentation? Web-based documentation, being the most flexible, is the most difficult to assess with respect to the four keys. It can be extremely *useful*, but only if care is given to the development and presentation of content. Too much, and you'll overwhelm the user; too little, and the documentation won't meet the user's needs. Similarly, the Web-based approach can make documentation extremely *accessible*, or hideously inaccessible. If the information's not organized well, you can waste your time trying to find what you need, rather than putting what you need to use. When developing Web-based documentation, it's sometimes easy to forget that documentation is a means to an end, not an end in itself.

Web-based documentation excels in the key principle of *accuracy*. The ability to automate creation and maintenance of information give you the means to deploy documentation that can be updated almost instantaneously when the situation changes. It's also highly *available*, although its reliance on technology imposes several preconditions on its use and introduces points of failure that don't exist when working with hardcopy.

Web-Based documentation rated according to the four keys of good documentation (out of ***):

Useful: *
Accessible: **
Accurate: ***
Available: **

Flat-File Documentation

Pros and Cons

Pros:

- searchable
- intangible
- high information density
- selective printing
- low maintenance cost

- instantaneous change propagation
- easy to back up
- revision history

Cons:

- searchable
- intangible
- prerequisites for use

Flat-file or text-file documentation is the electronic equivalent of the pocket-sized paper notebook I mentioned earlier. It's easy to use, free-form, and ideal for quick notes on simple tasks such as configuration changes. What Web-based documentation is to notebooks and binders, flat-file documentation is to single sheets of paper and Post-Its. Flat-file documentation offers many of the same benefits as Web-based documentation and suffers from many of the same drawbacks.

So why consider it separately? Because sometimes a Web page is overkill. Sometimes you just don't need all the bells and whistles offered by HTML and associated protocols. For example, I often use a flat file to document changes to a system's configuration or for tracking problem history. Sure, I could set up a Web site dedicated to collecting and organizing the information, but the flat-file approach fits my needs better: it's digital and therefore searchable, it's easily backed up, it's easy to maintain, and it's immediately available when I need it. When I finish working on a system, I can open the text file I keep the information in (stored on the system in a location known to everyone), in the editor of my choice, and quickly document who I am, when it is, what I did, and why. Over time, this file grows to become an informal history of the system, and it is invaluable when troubleshooting new problems.

Here's a snippet to illustrate:

```
20021217 mcl
Compiled and installed mysql client for use with snort.
20021215 mcl
 Compiled and installed apache 1.3.27. Sourcetree is in
   /usr/local/src, and the new apache is in
   /usr/local/apache1.3.27.
 Created symlink /usr/local/apache, designed to point to current
   running version of apache, to make switching between
   versions easier in the future. New version installed to
   address security problems with previous version.

 Ran /usr/local/apache1.3.27/bin/apachectl configtest with
   success.
20021210 mcl
 Put checklinks.pl in /usr/local/scripts, and created the
   following:
   /usr/local/scripts/check-http-links.pl
   /usr/local/scripts/check-other-links.pl
   /usr/local/scripts/check-https-links.pl
```

```
    Created a cronjob to run each week and mail the output to
      myself.
```

Whenever I have something to add, I can open the file in an editor and add the new information at the top. It's a simple approach that balances the convenience of paper with the flexibility of electronic documentation. If necessary, I can use common OS tools (e.g., grep) or built-in editor functions to search the document. I can easily produce a hardcopy version if necessary. It's backed up along with other critical system information and user data, so it won't get lost.

You'll still need a computer to use flat-file documentation, but the approach circumvents some of the problems associated with Web-based documentation. For example, if you store the file(s) on the system on which you're working, or often work, you won't need to worry about accessing a remote system to obtain the documentation. If the documentation content is system maintenance information, as in the previous example, you'll have a document that stays with the system in question, easily accessible as you work on the system. Of course, if the system itself isn't usable, you'll be unable to access the files. Therefore you should make sure any documentation you keep on systems scattered throughout your organization is routinely collected and stored in a central location. An automated tool such as rsync is ideal for this.

I'd like to take a few minutes to discuss a special case of flat-file documentation: source code annotation (i.e., "comments"). As I've mentioned earlier, it's not uncommon for me to load some source I've written a year or more ago only to discover that I've not only forgotten how or why I did certain things, but that it looks so foreign to me I'm no longer sure I wrote it. I'm glad to say that's a rare occurrence for me these days, but it may still happen to you. Worse, someone else may look at your code and be completely lost. That's just fine if you're competing in the Obfuscated Perl contest (*http://www.sysadminmag.com/tpj/obfuscated/*), but it's not so great if you've got some critical code maintenance to do and you're staring at your code, completely lost. Remember to add comments describing the purpose of your functions and subroutines. Write short notes that explain the use of variables, and if you cut any corners or take any shortcuts (we promise we won't tell), be sure to note them as well, to aid in future debugging.

Similarly, adding a separate text file or three to your code explaining the code's purpose and use, the method of installation, and a history of changes made to the code (commonly found in many projects as README, INSTALL, and CHANGES files, respectively) can be very helpful to anyone who looks at the code later, including yourself. You may want to add another two files, BUGS and TODO, which are (surprisingly enough) lists of known problems, and changes yet to be made.

How Does Flat-File Documentation Rate?

As you can see, you can take advantage of simple flat-file documentation in a variety of ways. Let's review the four keys of good documentation as they apply to the flat-file approach. Flat-file documentation tends to be relatively short and addresses a fairly narrow scope, so it's *useful* and *accessible*. It tends to be maintained manually, so a certain amount of effort is needed to keep it *accurate*. However, its digital format places some limits on its *availability*, though typically not as many as for Web-based documentation.

Be sure to choose a format that is appropriate to the information you're storing. You wouldn't want to keep your entire department's troubleshooting procedures in a flat file, nor would you want to build a Web site around half a page of data that never changes. You wouldn't want to keep only paper copies of every field name and table definition for 200 terabyte-size databases, just as you wouldn't want to create electronic-only copies of system-critical information for disaster recovery.

You may find yourself using a combination of approaches; in fact, I'd be rather surprised if you were able to work effectively using just one of the three formats in all situations. Don't be afraid to replicate information in multiple formats; I'm a huge fan of flat-file READMEs and INSTALL documents, but I still find myself printing them out because I work better following a document next to the keyboard, rather than having it on a screen that may not be able to display the information when I need to look at it. I use flat-file documentation to track system changes and work performed, but I still centralize the information regularly and make it available via a Web page. You should do what works for you and for the others who must create, maintain, and use the documentation. Just remember the limits and drawbacks of each format, applying the four principles of good documentation: usefulness, accessibility, accuracy, and availability. If your approach violates one or more of these principles, consider ways to adjust the information to avoid the problem, or explore other formats that are more appropriate.

Flat-File Documentation (out of ***):

Useful: ***
Accessible: ***
Accurate: *
Available: *

	Useful	Accessible	Accurate	Available
Paper	1	2	1	3
Web	1	2	3	2
Flat-file	3	3	1	1

Table 1: Document Formats Ranked by the Four Keys

Recommendations

What do I recommend? I could take the easy way out: "It all depends on your particular situation." But I'm not afraid to take a position on a matter of such importance. It's true that you should sit down and decide for yourself what's best for your particular situation. My own experience has been that a combination of approaches works best. I prefer to present my documentation digitally (either Web-based or flat-file), with both magnetic and paper archival copies of the data, and I like to keep paper "working copies" of certain mission-critical or frequently used documents. This insures maximum availability, usefulness, and accessibility of the information. The multiple redundancy just about guarantees that some form of the documentation I need can be retrieved, and paper copies give me exactly the information I need (making it more accessible for me) in the form I'm most comfortable following, thus making it more available and useful. (I've already admitted it to you: I'm a documentation Luddite. I prefer to work from dead-tree documentation.) The electronic presentation allows others in the target audience to generate their own paper copies of any documentation, offering them the same benefits I derive from such wanton tree-killing.

This narrows the key documentation problems down to the single dimension of accuracy, and then only when dealing with the paper copies. The paper archival documentation can be rotated on a regular schedule, with new documentation produced to replace the old, which is recycled. The paper copies are generally recycled as soon as I'm done using them, or are kept as source material for document revisions (you might be surprised how much useful information winds up as marginal notations on working copies of documentation), then recycled.

4. Documentation Tools

No matter what methods you choose for documentation delivery, you'll need some mechanism to transfer your knowledge, and the knowledge of others, into the formats you've decided on. Here, we'll briefly explore a few options for each format.

Flat-File and Paper Documentation Tools

If you're working with flat files, your first instinct is to whip out your favorite text editor and start writing. This is certainly a valid approach, and it gives you certain abilities such as search-and-replace. However, there are several other tools you can use that offer both convenience and flexibility.

First, if you're working within a UNIX-variant operating system, chances are you have access to the command "script." From the manpage: "script makes a record of everything printed on your screen." In other words, while script is in effect, all interaction with the shell is logged to a text file. This can be incredibly useful in a number of scenarios, particularly when creating documentation. For example, let's say you need to document the installation and configuration of a new software package. You could use pen and paper to write down each step along the way, or even try to recall each step after you're done. But with "script," you can have a verbatim record of the entire process, which you can then edit for inclusion in documentation.

Suppose you have a short series of commands you want to include in your documentation, along with any output. You'll execute the command "script":

```
$ script
Script started, file is typescript
```

. . . and proceed with the work you need to perform and capture for documentation. When you finish, CONTROL-D (^D) will end your script session and close the textfile being used to capture your work:

```
$ ^D
script done on Mon May 19 12:26:59 2003
$
```

Now you have the entire session available for editing and inclusion. Below is the output of a very brief script session, complete with errors:

```
$ cat typescript
Script started on Mon May 19 12:31:03 2003
$ tar xf dnsparse.tar
$ cd dnsparse
$ gcc -o dnslex dnslex.c
gcc: not found
$ /usr/local/bin/gcc -o dnslex dnslex.c
$ ls -la dnslex
-rwxr-xr-x 1 root other  12338 May 19 12:31 dnslex
$ script done on Mon May 19 12:31:34 2003
$
```

The inclusion of typos and errors may initially seem wasteful to you, increasing the amount of editing you must do. I tend to incorporate the errors into the documentation, pointing out common pitfalls of the process I'm explaining. In the example above, gcc is not in root's path, so when including this series of commands in my documentation, I'd be sure to remind the reader to call gcc with the full path (as I've lazily done above) or else to modify the appropriate environment variable before beginning the work.

Other UNIX shell functionality, such as redirection, can be quite useful. If you need to append new information to an existing flat file, such as a new entry in a running log of work performed on a system, there's really no need to load up a full-blown text editor. Just take advantage of "echo" or "cat" in conjunction with redirection:

```
$ echo "20031207 mcl restarted apache" >>/var/log/maintenance
```

. . . will append the text in double quotes to the end of the file /var/log/maintenance (assuming I have the privileges necessary to write to that file).

If I've got a bit more to say, I could continue using "echo", but I prefer "cat":

```
$ cat >>/var/log/messages
20031207 mcl
   Added domain example.net to virtual hosts
   Migrated all domains from name-based to IP-based virtual
   hosting
   Enabled ssl for example.net, certificate stored in
    /usr/local/ssl/certs/example.net.crt
^D
$
```

This allows me to easily enter multi-line text and redirect it to the end of an existing file. Of course, I have to be a bit careful—I don't have the ability to return to a previous line to correct errors. If I'm not sure what I want to enter, or if I think I'll need the ability to edit as I go, I'll use a real text editor instead. But if I know what I need to say, I'll use one of these time-savers.

Another trick I've used makes flat-file data entry simple for multiple authors, by coupling the task of documentation with the ease of sending email. You can set up an email account to which you send all entries in the file. If your mail server's UNIX-based, you'll end up with a flat file containing any information sent to that address, in standard mbox format. You could stop there, but you'll probably want to use something like procmail to process the mailbox a bit further, separating the entries from the mail headers and appending them to a non-mailbox file, along with a timestamp and the author's name.

Common UNIX commands such as grep, sed, or awk give you even more flexibility. If you're ambitious, you may want to build a set of shell scripts (or short programs in the language of your choice) to assist you in your data entry. Don't feel you're constrained to a text editor if you decide to use the flat-file approach to documentation. The sky (or your operating system) is the limit!

These are all good methods for collecting and editing your information. But what about organization? For the documentation to be truly useful and accessible, the information needs to be organized sensibly. If you're working with flat-file documentation, organization can be as easy as establishing an agreed-upon format for the information (e.g., ANSI-standard date format and initials, followed by description of work performed, appended to end of existing log, terminated with a blank line) and insuring that everyone adheres to it.

If you've decided that paper documentation is the way to go, you'll also want to create additional groupings of information, possibly by date, or system, or subject matter, or some other set of criteria that's relevant to your environment. You'll want to create a Table of Contents for the information, to assist readers in quickly locating the information they need, and you may also want to create an Index that specifies the location of specific items in the text. Here, a text editor alone won't suffice, unless you're a glutton for punishment. If you need a table of contents and an index, you also need a word processor to facilitate the creation of these two items.

It's important to remember that a word processor won't magically organize your text for you. You'll have to do that yourself. A writing course is beyond the scope of this document, but I should at least remind you that much of your document's utility rests on its organization. Even the most informative document can be rendered inaccessible by poor organization. Pay close attention to the four keys to good documentation, remember your audience, and organize your documentation accordingly. The document organization should facilitate the use to which the documentation will be put. If it's meant to be an encyclopedic reference, it may fall short of its mark if you organize it as a narrative. If it's a record of your organization's network architecture, you'll probably want to include a few well-labeled diagrams and a list of all entities depicted.

Once you've decided on an organizational scheme, a word processor can assist you in creating and maintaining a table of contents and an index. Don't go overboard, though. If you're only writing a five-page document, you can probably manage without either a table of contents or an index; the section headings should suffice. Your goal is to insure the reader is able to navigate quickly through your document, finding what they need. A five-page document's structure and content should be obvious to anyone glancing at it. It's only when you're working on long documents that the reader will require a map of your organizational scheme and content.

If in doubt, test. Once you've printed your documentation, give it to people to try out, and see how they react. Take notes, and modify the document accordingly. But be careful! The people to whom you gave the document may not be part of the target audience and may be trying to use the document in a way you didn't anticipate. That's not necessarily a bad thing, but you should consider the possibility. If you think the unexpected use is valid, you may need to rethink the organization of the document. Someone using a network diagram to troubleshoot connectivity issues is probably making valid use of documentation you might have created only for archival purposes. Someone using a quarterly budget to infer network architecture is using the document incorrectly.

The most robust document-testing approach should allow the document to be tested by the following people, in this order:

1. Yourself/the author
2. One or more members of the target audience
3. At least one person outside the target audience

This will give you or the author a chance to catch any glaring errors, oversights, or omissions, before the document is circulated. The review by a target audience member will help insure that your document will be understood by those who must use it. The final review, by someone outside the target audience, will help you identify any errors that slipped unnoticed past those too familiar with the content to catch them.

Web-based Documentation Tools

Web-based documentation tools are useful even if your ultimate documentation format is not Web-based. There are three categories of Web-based tools you can use to facilitate the collection and organization of information to be documented: Web-enabled applications, ticketing/bug-tracking systems, and collaborative systems.

Web-enabled applications are exactly that: applications you can interact with via a Web browser. Of particular interest to you in your documentation tasks are those that provide some form of automated discovery and/or monitoring of various parts of your environment. Network monitoring tools, for example, often contain a Web-based interface and status report facility. You could take advantage of such interfaces via "scraper" scripts (scripts designed to parse a remote Web page and extract information of interest for further processing), or you could consider the page itself as documentation. The status display of a Web-based network monitoring tool serves as automatically maintained, Web-based documentation delivery. (Nagios [*http://www.nagios.org*] is an excellent example of such an application.)

Web-based ticketing, help desk, and bug-tracking tools have become popular in recent years, and many have found their way into regular use in IT departments. Although their names differ, their purposes are similar: to centralize information from multiple sources in a structured format and to notify certain individuals and/or groups when certain conditions have been met, such as ticket resolution, bug elimination, or expiry of certain time periods. Information in these systems has a definite lifespan, and the systems are designed to track certain events within this lifespan. They tend to be modal, marking information as either "open" or "closed," "resolved" or "unresolved."

They are designed to take action when these modes change, or when they remain unchanged for a specified time. They also provide organizational tools that facilitate (in some cases, require) the categorization of both information and users of the system. The extraction of information from these systems is often accomplished via some reporting facility, and it tends to be either depth-oriented (providing the history of a single "ticket" or individual) or breadth-oriented (providing an overview of one or more categories or classes of information or users). These systems can be extremely useful for reporting problems and documenting the steps taken in solving the problem, and for building work histories of a system. Request Tracker (*http://www.bestpractical.com/rt/index.html*) is a good example of a Web-based ticketing/helpdesk tool, and Bugzilla (*http://www.bugzilla.org/*) is a prime example of a bug-tracking tool.

Collaborative systems are similar to the Web-based ticketing systems discussed above, but I want to distinguish between the two categories to highlight the difference between the structured-format and result-oriented ticketing systems, and the open-ended approach of collaborative systems. Whereas ticketing and tracking systems expect information of a certain type, in a certain form, collaborative systems tend to be free-form, permitting information to be entered as the user sees fit. Whereas ticketing systems tend to be modal, collaborative systems are more general information repositories, designed mainly for storing information rather than facilitating actions based on the information. The Wiki Project (*http://wiki.org*) is a perfect example of a collaborative system. It allows multiple users to create and modify information in a central repository, which itself serves as the delivery mechanism.

This list is by no means complete; many other Web-based documentation tools, both free and commercial, are available to you. In this instance, Google is, as they say, your friend. Spend some time exploring your options. Do you want to write a tool in-house, or would it be better to pay for a commercial package? Would you prefer to bring in an open-source package and customize it to meet your needs? Whatever you decide, make sure you're aware of your needs before you begin your search. Without a clearly defined set of requirements, you may end up with exactly the wrong tool for the job.

5. Documentation Strategy

Now it's time to discuss documentation strategy. You're probably in one of several potential situations as you start your documentation efforts:

- No existing documentation, and no documentation policy
- No existing documentation, but a documentation policy exists
- Existing documentation, and existing policy
- Existing documentation, but no existing policy

Documentation policy generally applies groupwide or organizationwide, defining document content, storage, and retention parameters. You should always begin by checking to see whether a policy already exists in your organization. It may save you a lot of hassle later.

Dealing with existing policy opens a complex tangle of technical and political issues that this booklet isn't suited to address. If you're in an organization with an existing policy that governs any documentation you may produce, my advice is to follow that policy to the best of your ability, working with others in your organization to adapt any parts of the policy that hamper your efforts. The policy may be very thorough, restricting content and format, or it may be as simple as, "All Web-based documentation must work with Internet Explorer 6.1, and only Internet Explorer 6.1."

We will hope that you're relatively free from policy restrictions as you set forth to document your corner of the world. This is the best possible situation: you've discovered a wonderful opportunity to create policy! (Those of you ready to lynch me for advocating the creation of yet more policy should consider the benefits of being a rule-maker. You have the chance to create policy that's sensible, rational, technically feasible, and—best of all—under your own control!)

The Most Common Scenario: Documentation Without Policy

Now that I've appealed to your megalomaniacal side, I'd like to focus on situation #4 above: documentation exists, but no policy has been set. This is the scenario many of us find ourselves in. We've inherited an infrastructure, with all its bits and bobs, nooks and crannies, bugs, quirks, and secrets. We've probably also inherited a lot of ill-documented in-house code, peculiar configurations, and, if you're very lucky, a lot of half-written or barely maintained documentation (those of you who inherited well-oiled, perfectly running, thoroughly and recently documented sites should have stopped reading around page 5). How do you start making sense of this sort of environment?

You can take control of this type of situation in four easy steps (Call now! Operators standing by!):

1. Identify what's necessary.
2. Identify what's available.
3. Identify what's important.
4. Establish procedure.

Identify What's Necessary

First, identify what's necessary. Make a list of the tasks associated with system administration in your sphere of influence (whether that's a single system or piece of code, an entire datacenter, or a team of sysadmins). Tick off each task you think should be documented. Now go back and ask yourself, "Why should this be documented?" for each task you marked.

If you're having difficulty compiling the list, spend a week or two taking notes on the various activities you're called upon to undertake during the course of your work. At the end of this period, you should have a fairly representative list. Don't forget periodic tasks that may not have occurred during this data-gathering phase, however.

Let's say your list looks like this:

- Daily backups, tape rotation, and offsite storage
- Routine checks of system status
- Configuration and installation of new systems
- Maintenance of financial databases

It's a fairly short list, so let's assume you marked each line as needing documentation. You should start by asking yourself, "Why do I need to document the daily backup routine? What purpose will this documentation serve?" You may find you have more than one answer for that question, which is fine. Each answer is more than likely a separate document. You might answer, "Well, I need a procedure for the junior admins to follow when checking the backups, changing the tapes, and getting the archived backups ready for pickup by our offsite storage company." You might also answer, "I need a policy and procedure for disaster recovery with respect to these backups and the systems they're designed to protect." That last answer's actually two separate documents. You might continue, "I need a document describing the system configuration, in case we need to rebuild the backup server from scratch." Finally, you might say, "I need a description of the correct troubleshooting procedure for the backup systems."

From that one daily task, you've identified five unique documents you need in your environment. You should continue down your list of tasks, asking and answering the same question each time: "Why?" You'll start to see certain common answers—you need a policy for this, a procedure for that, a list of information about the other. Once you've determined what you need, categorize the list according to document type. It'll make documentation development easier, by allowing you quickly to identify sets of documents that may be best addressed by a certain person, group, procedure, and/or tool.

Identify What's Available

Second, identify what's available. We're assuming you've got some documentation lying about somewhere; go find it. Gather it all together and sort it into the document type categories you set up in the previous step. Compare the two sets of documentation, desired and actual. Is there any overlap? If you find existing documentation that matches an item on your "desired documentation" list, examine it closely. Is it adequate for your needs? If so, cross that item off your list of desired documentation. Don't celebrate just yet, though. It may be sufficient, but is it good documentation? Reexamine it, this time grading it according to the Four Keys of Good Documentation: Is it useful? Is it accessible? Is it accurate? Is (was) it available? If it fails this test, you've got some rewriting ahead of you.

Now that you've compared the two lists for possible overlap, let's consider disjoint: What existing documentation is not on your list of desired documentation? More important, why does that document exist? You need to decide whether to add it to your list of necessary documentation or to eliminate that document from your environment. Someone at some point thought people needed the document you now hold. Before you dismiss it as unnecessary, you should try to discover the reasons behind its creation. Perhaps some vice president once asked that the document be created, but she hasn't touched it since. This is, of course, the universe's way of trapping you: a week or so after you destroy the only copy of that document, the aforementioned VP will come looking for the head of the person who destroyed it, because she needs it urgently for an important meeting. The moral: Always research the history of a document before eliminating it from your set of documentation.

When you're finished with this step, you should have three piles of documentation:

1. Documents that are necessary and adequate
2. Documents that are necessary but need work
3. Unnecessary documents

The documents still left on your list of desired but nonexistent documents need to be created, and those in category #2, above, need to be modified.

Identify What's Important

The third step is to identify what's important. Now you will begin to prioritize your documentation work. What documents could bring the world to a halt if they aren't created immediately? What documents can wait until next quarter? What documents will take the least time to complete? What approval process, if any, must your documents go through before they are accepted within your organization (check existing organizational policy to answer this question)? Rank both the desired and the existing but inadequate documentation according to priority, highest to lowest.

The prioritization scheme is up to you. You may decide to tackle the quick work first, getting it out of the way so you can focus on the meatier projects. You may instead decide that all of the procedural documents for your junior staff have top priority, because it's imperative that you establish some sort of consistency and clarity in your service delivery model (I promise I won't write like a manager again in this booklet). You might decide that disaster recovery or security policy documents have top billing. There may be a pet project of senior management that requires your full attention. This is an instance where the answer really does depend on your unique situation. Whatever you decide, list your priorities and proceed accordingly.

One item I urge you to place near the top of your list of priorities, however, is "anything documented only in someone's head." "Walking documentation" can be an incredibly useful tool in almost any environment. Until the person leaves, that is, or gets hit by a train, or takes a sabbatical to sail around the world. When this wonderful person, this font of wisdom and knowledge, is suddenly and unexpectedly unavailable, he becomes no more useful than a remote Web server over which you have no control. If you've got someone in your organization whose head is chockfull of unique and valuable information, getting it into a more readily available form should be one of your primary goals.

Establish Policy and Procedure

Finally, establish documentation procedure and policy. Regardless of what else is on your prioritized list of documents, these two documents should be at the very top. Setting policy and establishing procedure for your group or organization's documentation will insure a consistent product and a clearly defined set of tasks for everyone involved in the effort.

Policy

Your policies should establish guidelines and boundaries for the creation and maintenance of all your documents. Some items you should include in your policies:

- Acceptable rationales for documentation
- Storage locations and methodology
- How to determine responsibility for maintenance
- Revision methods and timeline
- Acceptable or required formats
- Acceptable or required content
- Requirements for acceptable procedures (a reference to the procedures you are going to write)
- Acceptable tools for document creation and presentation

These policies will serve as a framework to help people find and use your documentation.

Myriad examples are available online, from the brief (*http://whpo.ucsd.edu/policies/doc.htm*), to the practical (*http://www.debian.org/doc/docpolicy, http://www.ucar.edu/ncab/Policies/documentationpolicy.html*), to the formal (*http://web.library.uiuc.edu/ahx/documpol.htm*). Google around a bit to get a feeling for what others have done.

Documentation Procedure

For a long time, I felt confused about the difference between a policy and a procedure. Finally, I understood. A policy is designed to establish boundaries and guidelines, to outline requirements and restrictions, to set forth principles and standards. A procedure is a method for implementing a policy.

So, now that you've defined a documentation policy, you need to establish a set of procedures that implement your policy. The procedures should address:

- How and when to create documents
- How and when to store and to delete them
- How to verify their completeness and accuracy

The last item may work better as a separate review procedure, leaving the first two items as the creation procedure.

Your descriptions should be clear enough that everyone involved in the process understands exactly what needs to be done, and how.

Take, for example, the creation and maintenance of an online FAQ database. Let's assume the document policy is already in place, but that it places no significant restrictions on this work. The procedure may read something like this:

1. When answering a request for technical help in person or via email, phone, or the online ticketing system, check the request against the list of IT FAQs at *http://our-intranet.example.com/faqs.cgi.*
2. If the request is addressed by one of the existing online FAQs, refer the user to the appropriate URL and consider the incident closed.
3. If the request is not addressed by one of the existing online FAQs, resolve the incident to the best of your ability, then:
   a. If no FAQ exists for this issue, create one using the admin interface at *http://our-intranet.example.com/faq-admin.cgi* and supply the answer and/or steps that successfully resolved the incident.
   b. If this incident was a special or more general case of an existing FAQ, edit that FAQ entry using the admin interface at *http://our-intranet.example.com/faq-admin.cgi*, adding the solution to the end of the existing FAQ and noting it as a special or a more general case of that FAQ.
   c. In either case, be sure to include all relevant hostnames, IP addresses, software names and versions, vendor names, absolute paths, and commands or menu selections relevant to the solution of the incident.

Alternately, let's say you write a procedure for documenting the hardware and software configuration of a standard Microsoft Windows desktop system in your organization. It may look something like this:

1. Record all BIOS settings. Enter the PC BIOS according to the manual that came with the motherboard, or according to the instructions presented during the system POST.
2. Remove the PC case to provide full access to the motherboard. Record all jumper settings for the motherboard. Use the motherboard manual as a guide to locate each jumper and determine the jumper's purpose.
3. Sketch the rear panel, and draw and label all external connections. You may use the FakeGraphingProgram templates stored on *http://soopersekrit-it-server.example.com/templates/* to facilitate your work.
4. Draw and label all internal connections. Use the templates from the source in #3 to aid you in your work.
5. Make a list of all manuals associated with this system, and where they are located.
6. Create an inventory of all components, including vendor names and model numbers for each CPU, heatsink, fan, peripheral card, bay-installed peripheral, and external peripheral. List all cables by type and number.
7. Make a copy of all system files, including C:\AUTOEXEC.BAT, C:\CONFIG.SYS, C:\WINDOWS\WIN.INI, and C:\WINDOWS\SYSTEM.INI.
8. List all software installed on the system, including the absolute path to the software, vendor name, software name, and version number. If there are any license keys associated with that software, list those as well.
9. Store all information collected in steps 1, 2, 5, 6, 7, and 8 in a single ASCII file, listing each step, followed by the information obtained in that step. For steps 3 and 4, provide a URL to the two image files containing the diagrams.

   All information should be stored by office number and floor in the form X###-##, where X is the alphabetic office designation, the first ### are the office number (if the number is only two digits or a single digit, the leading #'s should be 0s) and the last ## are the floor number (if the floor number is a single digit, the first # should be 0).

Together, the policy and procedure define the boundaries within which documentation will be created. Now that you've put together a policy and a procedure (I'm assuming you've taken the steps necessary to get both approved for use in your organization), you must become an evangelist for them, but remain dispassionate: They're documents, not children.

Insure that those affected by your policy and procedures are aware of them, understand them, and don't have any major objections to them. Sure, you'll run into a few people who are steadfastly opposed to any documentation effort whatsoever, but putting their complaints aside, you may nonetheless discover that either or both documents do not fit your environment very well. Take such criticism to heart and make appropriate modifications.

Observe people as they attempt to implement the procedures you've defined. Watch for problem areas in the procedure itself or in the person's workflow. Your goal is to make using documentation routine. If your procedures are awkward or burdensome or the policy doesn't suit your environment, you may inadvertently be building resistance and resentment instead of cooperation.

Procedure documentation shouldn't become an exercise in creation by committee, however. A certain degree of resistance to this sort of change in the workflow is inevitable, from inertia if nothing else. It's important to emphasize the value of the effort. One trick I've used with some success is to highlight a workday situation that could clearly benefit from the documentation you're advocating. For example, if your goal is to capture the steps necessary to administer a certain system, every time you hear someone grumbling about having to reverse-engineer a procedure or second-guess another admin on that system, you're being offered a perfect opportunity to promote documentation of the process. Playing to the ego works well here, too—mention that the admin who documents the process can insure that the process is completed to her satisfaction. There's a bit of control freak in each of us. Preaching to your team's inner control freak can work wonders when trying to get buy-in for documentation work.

Documentation Maintenance and the DRI

Soon the documentation habit will take root, and you'll start building a repository of useful documentation. Once your team discovers that the documentation is useful in their daily lives, that repository may grow more quickly. This kind of growth, by voluntary contribution, should be your longterm goal. The pitfall is that the growth must be *controlled*. Too much, and you'll end up with irrelevant, incoherent, and/or over-specific documentation. Too little, and you'll have a very sparse set of documents that convey too little information about too few subjects.

This is where the person responsible for a document, the Designated Responsible Individual (DRI), becomes crucial. It's her job to insure that contributions are complete, accurate, and necessary. If the DRI is the sole contributor to the document, that's a fairly simple task, part of the normal writing process. If, however, the DRI is only a co-contributor or is overseeing the other contributors, she must assume the role of editor, making sure that the document adheres to the Four Keys to Good Documentation, that the procedures are being followed, and that the implementation of the procedures complies with the documentation policy. The DRI should also take responsibility for regular document review, according to the review guidelines set forth in your policy. In short, the DRI is responsible for the entire document life cycle, from creation, through revisions, to deletion.

The role of the DRI may be as formal or informal as you wish; there's no need to create a special title and modify someone's job description. The important point here is that certain steps need to be taken to guarantee that documentation is useful and well-maintained. Soon after documents become neglected, they generally become worthless. Architectural blueprints are a good example. They are used to specify the parameters for construction. Assuming the blueprints are followed faithfully, at the moment of completion, reality and documentation match. But if an alteration is made and the blueprints aren't updated accordingly, the next attempt to use the blueprints as documentation could result in disaster. In the tech industry, changes occur much more frequently than they do in buildings, so it's particularly crucial that someone take responsibility for maintaining the accuracy of your documentation.

6. Examples

Now that we've covered everything you need to get started, I'd like to give you a few examples of the types of documentation we as system administrators are likely to encounter. These are not meant to be skeletons or models, but, rather, to give you some impression of the breadth of documentation our field encounters and creates.

The Run Book (Procedural)

A run book is a document that lists items to be monitored and operational tasks to be performed regularly. Its purpose is to provide a clear, systematic, and comprehensive roadmap for monitoring and maintaining the systems in an environment. You'll often find these in larger shops, and they're common in Network Operations Centers (NOCs).

Here is a sample table of contents from a run book for the maintenance of a large-scale, dedicated UNIX SMTP server:

Application architecture	1
Configuring the system and network	7
Starting and stopping the mail server	14
Managing accounts	15
Managing domains	19
Reviewing sidelined mail	21
Handling mail in the error directory	22
Maintaining/monitoring log files	24
Defragmenting the Message File system	26
Performing backup and recovery	28
Processing bad messages	31
Monitoring syslogd output	32
Monitoring cron jobs	33
Monitoring file system usage	34
Monitoring network availability and usage	36
Configuring and running imsysmon	40
Enforcing message aging policy	41
Managing the configuration database	43
Disaster recovery	51

Note that the run book isn't a step-by-step guide to the workday. Rather, it's a comprehensive document broken into sections, with each section describing in detail the procedures necessary to deal with a certain aspect of the job, specific to that system. It's used as a reference manual, to guide each admin in the proper handling of every possible situation regarding that particular system or set of systems. (To further illustrate the contents of a run book, Chapter 7 offers a case study of Amanda backup administration.)

The Build Book (Procedural)

A build book is a document that details the procedures for installing equipment, systems, and/or software. It's frequently considered a companion to a run book, but the build book is commonly used by a different group of admins than a run book (except in small environments, where admin responsibilities often overlap). The build book contains information such as:

- Parts inventory
- Hardware assembly and configuration
- Pre-installation considerations
- Installation procedures, hardware
- Installation procedures, software
- Software configuration
- System integration procedure

Together, these sections thoroughly document the steps necessary to build, configure, and install a system compliant with policy. A build book is a much more sequential document than a run book; it is designed to be followed step by step. Once the work in the build book is complete, the procedures in the run book can be used to bring the system up and keep it running smoothly.

System Configuration Documentation (Informational)

If you've implemented a build book in your environment and your systems adhere to it, you probably have already captured most of your system configuration information. However, even in environments with religiously obeyed build books, system configurations change: software gets added or removed, configuration files are modified, and so forth. It's therefore a good idea to create and maintain a set of documents that capture all the configuration information for your systems. These documents often contain:

- Network configuration by interface
- Disk partition layout
- Installed software, with any significant configuration information
- Hardware and peripherals inventory
- Physical location of system
- Authentication information (method used, configurations for method, etc.)

- Network integration information (e.g., ntp configuration, DNS resolver configuration)
- List of authorized superusers
- List of authorized sudo users
- List of individual(s) responsible for system, with contact information (preferably via multiple communication channels)
- OS version information and installation idiosyncrasies
- Patches installed

With good system configuration documentation, you can rebuild a system from scratch, as well as handle other disaster recovery tasks. It's also a great reference for software version and OS patch maintenance. Most of the information can be captured programmatically, so it's an excellent candidate for Web-based or other digital documentation.

Maintenance Logs (Informational and Procedural)

A close cousin to the system configuration document is the maintenance log. A maintenance log is a document (often relatively simple) that records who did what, when, and why. An example was provided on p. 25. This very simple example, although it lacks justification for the actions taken, captures the essence of a maintenance log. The example in Chapter 3 is taken from an operating system/software maintenance log, but hardware maintenance logs are not uncommon. Maintenance logs are extremely useful for troubleshooting recurring or obscure problems, as they provide a record of all work performed on the system and may shed light on hard-to-spot interactions between seemingly unrelated symptoms. In short, they're the history of the system. As in life, it's important to learn from history, lest we repeat it.

Solution Databases (Informational and Procedural)

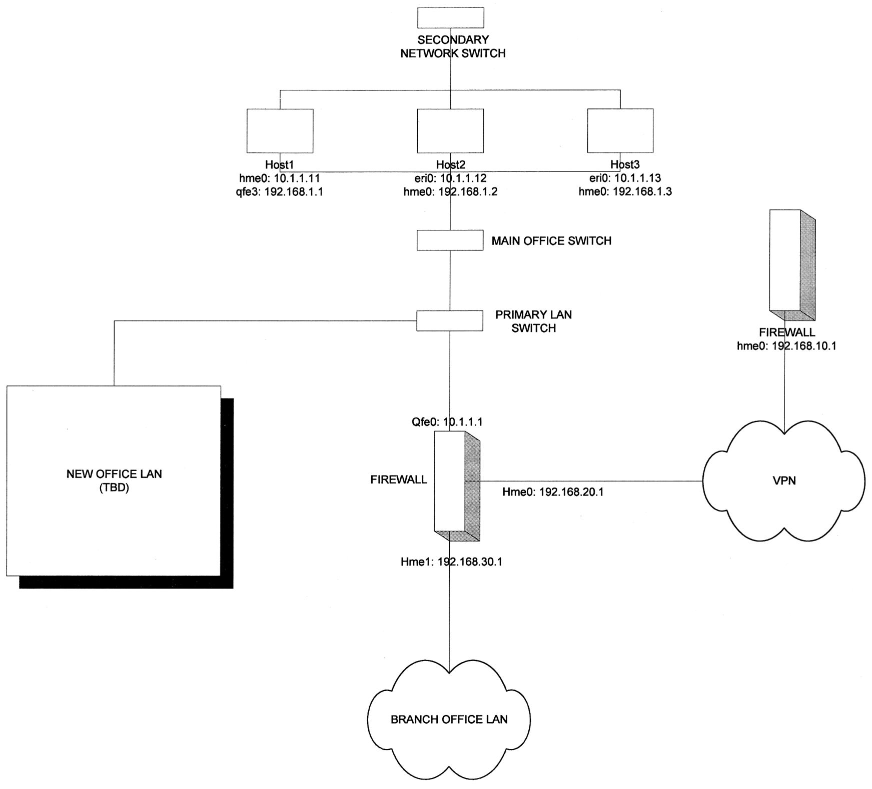
A solution database is an electronically accessible, searchable collection of descriptions of problems and their solutions, often with a Web-based front end. You could use a general-purpose bug-tracking or ticketing system as a solution database, if you wished; the only real requirement for such a system is that the problems and solutions provide enough detail that someone other than the original author could replicate the solution.

At first glance, this may seem similar in concept to a FAQ. However, a FAQ tends to contain less specific information than a solution database. Usage differs as well: A FAQ is often written by a technical person for a somewhat less technical audience, whereas solution databases contain information written by technical people for other technical people (often, team members or peers). Further, FAQs tend to be read before the seeker pursues other information; they seldom offer a definitive answer. A good solution database should work in the opposite way, containing enough information to be the ultimate resource for answers to a particular problem.

A search function is not mandatory, but, as with any large collection of information, it's often necessary. A search capability allows you to mine the database quickly for specific information about a given problem, solution, system, or component. Over time, a log of such searches will provide a de facto history of work performed on a given component or system and will give you quick insight into problems that tend to recur in your environment.

Network Maps (Informational)

Network maps, whether automatically generated or manually constructed, are invaluable. Whenever you're faced with a complex network or an unfamiliar one, a network map can provide you with a quick overview of the logical topology, allowing you to troubleshoot a variety of problems, from host misconfiguration to packet transit problems. A network map contains a logical diagram of all relevant network-addressable devices and any pertinent information about them, such as interface names, IP addresses, MAC addresses, routes, and so forth. The amount of detail to be included is up to you, but I tend to err on the side of verbosity, as my maps must occasionally be used by people with only a passing familiarity with the actual architecture of the network.



Here's a sanitized map taken directly from real-life daily use. Note that several systems have multiple interfaces, and information about each is included. This can be extremely useful when troubleshooting routing problems.

This particular map was created by hand, and it represents a system that must often be maintained remotely. (It's on a tropical island. Seriously. You do not want to be paged at 3 a.m. about a problem that's 4,000 miles away.) When you don't have physical access to the devices with which you must work, a network map is essential (and so are wiring diagrams, which are the physical counterpart of the information represented logically in a network map).

Various products can create network maps automatically, for example, HP OpenView (*http://www.openview.hp.com*), IP Sonar (*http://www.lumeta.com*), and Tkined (*http://wwwhome.cs.utwente.nl/~schoenw/scotty/*). These and similar tools locate resources on your networks and represent them in a graphical form. Often, such tools may also be used to provide near-real-time monitoring of those resources (OpenView, IP Sonar, and Tkined offer that option).

It's important to understand the difference between creating a map by hand and generating a map automatically. The former results in a representation only of what you feel needs to be represented. The latter, however, creates a representation of every accessible device, leaving you the task of whittling it down to something relevant to your needs. This can be handy when attempting to make an exhaustive map of an infrastructure, but I find that manual generation is often more efficient, particularly when the map in question is a small subset of an entire network.

Disaster Recovery Procedures (Procedural)

Every IT group these days should have some form of disaster recovery policy in place. If you don't, you should. Establishing expectations of what will happen should the worst occur is crucial for everybody's sanity—yours, so you'll know what's expected when Something Very Bad™ happens, and everyone else's, so they'll know Something Is Being Done™.

But disaster recovery policy is only half the picture. Recall the definition I provided a bit earlier, for policy versus procedure: A policy is designed to establish boundaries and guidelines, to outline requirements and restrictions, to set forth principles and standards. A procedure is a method for implementing a policy. Without the procedure, you'll know what should be done, but not how or when to do it. That's where the procedure comes in. It will explain, step by-step, what to do when, and to what, in an emergency.

Let's look at a simple example. Assume the disaster recovery policy states, "If our main office is incapacitated for any reason, our satellite office in Hillsborough, NC, should provide redundant functions for all main office activities within eight hours of the loss of the main office." That's pretty simple and straightforward. But for you, the person responsible for insuring that it happens, it's not nearly enough information. To make reality match policy, you need a disaster recovery procedure that contains such information as:

- How information in the main office will be replicated to the satellite office before any disaster
- How all functionality in the main office will be moved to the satellite office in the event of a disaster, taking into consideration that main office resources may not be available
- How to deal with any problems that may occur at the satellite office during and after the transition, as a direct or indirect result of the transition
- How any information regarding the transition will be propagated, both to network entities and to employees and other affected individuals
- In what order all of this will occur

Looking at this list, it should be obvious to you that you'll have to organize the process well in advance of actually needing it. As with any procedural document, you'll also want to test the procedure before actual implementation, to verify that what you've specified works the way you think it will.

The following chapter offers a case study of procedural documentation development. Earlier, I described this case study as part of a run book, but the process of developing the document may be used as a model for developing any procedural guide.

7. Documentation Case Study: Amanda Backup Administration

At my place of work, we currently use several methods to back up our critical data, including an installation of Amanda (*http://www.amanda.org*). The Amanda installation performs nightly full and/or incremental backups of various systems throughout our organization, and can occasionally be quirky—the daemons won't terminate properly on occasion, a tape may break or jam, the wrong tape may be loaded, etc. I often found myself answering questions about various aspects of maintaining the system—questions that would have been much less frequent if there had been a standardized procedural document detailing the routine tasks surrounding the maintenance of this system.

So I set out to write one. First, I wrote down a list of all the daily tasks associated with maintaining the system:

- Check to insure that the nightly backup ran properly
- Address any errors from last night's run
- Prepare the system for tonight's run

That's a pretty short list. However, the second item is hiding a great deal of documentation. In effect, it requires that I document all possible failure modes, or at least those that have been seen in our environment.

The list above doesn't cover less routine, but nonetheless necessary, tasks, such as:

- Change the tape cartridges
- Replace an unreliable tape
- Recover data from a tape, in case Mark's not around to do it
- Randomly verify the integrity of data on a tape

They were added to the list, along with a note stating when they should be done. Data recovery is normally a task I handle personally as the need arises, but it's important to document any tasks that rely on the knowledge in one person's head, to eliminate that person as a single point of failure for the entire process.

Now that I had my list of tasks to be documented, I sat and thought about who was going to be reading and working from this document. The primary audience is an individual without much UNIX background, and indeed without the root password to the backup system, so the documentation would need to concentrate on physical interaction with the backup system, rather than interaction with the software itself. The tasks that require UNIX knowledge and/or root access to the backup system are also documented, though, to provide continuity should those possessing such knowledge be hit by a bus, or monkeypox, or a severe case of unemployment.

So, I had a list of tasks to be performed, and I had identified my audience. I knew that this was a procedural document designed to concretize our backup policy. I had identified the type of content to be included, based on my audience, and I had found a likely candidate to write the document: me—not only for the sake of expediency, but also because in this instance, I happen to be the expert on the subject of our backup procedures, and I'm comfortable with the documentation process.

How should this documentation be presented? We use a Web-based documentation system for our group, so the document would certainly be added to that repository. That system is backed up on a regular basis, so there would be archival copies of the document stored safely offsite. However, I also wanted to produce printed versions of the document, so those who must work from it could have a short, tangible reference to carry with them to the backup system. Not only does this satisfy my penchant for wanting both digital and deadtree copies of documentation, but it nicely sidesteps the problem of obtaining Web access in a room where a GUI is hard to come by. Yes, I could have avoided using our Web-based documentation system in favor of a flat ASCII file stored somewhere else, but that would have created new problems, such as an additional document repository, and it would not have eliminated the need to use a computer to access the information while working on the backup system.

Having worked through the necessary preparatory steps, I was finally ready to write the actual document. I used my list of tasks as a guideline, and my knowledge of daily routine and the history of common problems with the backup system helped me flesh out the content. Below is the final document used for dealing with daily backup maintenance. For brevity's sake, the sections dealing with the less routine issues have been left out.

Daily Backup Maintenance

1. You should have received an email message from the backup system sometime during the night. The email was sent from "*backup@example.com.*" Find this email and read the report.
   The first thing to note is the "Subject:" line of the email. It should contain the current month, day, and year. If it contains something else (such as "BogusMonth 0,0"), contact Mark immediately.
   If there was a problem with the backup, the beginning of the email body will state: "*** THE DUMPS DID NOT FINISH PROPERLY!" and proceed to explain the nature of the problem.
   If the mail states, "THESE DUMPS WERE TO DISK. Flush them onto tape [$TAPE_NAME] or a new tape," where [$TAPE_NAME] is the name of one of the tapes we use for backups, take the following steps:
   a. Go into the server room, to the rack where the backup system is held. Locate the tape jukebox. Determine which tape magazine

is in use, by observing which magazine case is empty. The label on the outside of the magazine case will contain a number. This number is the first digit in the tape name. The tape's position in the tape magazine is the second digit in the tape name. That is, if the digit on the case is a 3 and the case holds a 6-tape magazine, you know that the magazine currently in the jukebox holds tapes 31 through 36, in slots 1 through 6, respectively.

b. Look at the LCD display on the jukebox. It will state which tape from the magazine is currently loaded in the tape drive. If the tape currently loaded is the one requested in the email, proceed to step g.

c. If the tape currently loaded in the tape drive is not the tape requested in the email, press the button marked "EJECT" on the front panel of the jukebox. The tape in the drive should be unloaded and placed back in the magazine.

d. If, for some reason, the tape does not unload properly when you press EJECT, notify Mark or his superior immediately.

e. Once the tape has been placed back in the magazine, use the UP and DOWN arrows on the front panel to select the appropriate slot from which to load the requested tape, assuming the requested tape is in the magazine currently loaded. Once the proper slot has been selected, press the LOAD button. The tape should be removed from the magazine and loaded into the tape drive. If this does not occur, notify Mark or his superior immediately.

f. If the requested tape is not in the currently loaded magazine, you will need to eject the current magazine, place it in its appropriate case, and load the magazine containing the requested tape. To do this:

    i. Press and hold the EJECT button on the front panel of the jukebox. The tape currently in the drive, if any, should unload, and then the magazine itself should be ejected from the jukebox. Once this occurs, you may release the EJECT button.

    ii. Place the ejected magazine back in its case and retrieve the magazine containing the requested tape. Use the method described in step 1.a to locate the appropriate magazine.

    iii. Having found the correct magazine, remove it from its case and insert it completely into the slot on the front of the jukebox, being careful to insure that the green arrow on the magazine is facing up and pointed toward the jukebox magazine slot.

iv. Once the magazine has been successfully loaded into the jukebox, return to step 1.e to load the requested tape.

g. If Mark is available, inform him of the steps taken above and forward to him a complete copy of the morning's backup email. He will take the following actions. If he is unavailable and you have been given appropriate access to the backup system, perform the following steps yourself:

   i. `/usr/bin/mt -f /dev/rmt/0 rew`

   ii. `/usr/sbin/amflush Nightly`

h. Allow time for the command in step g.ii to complete. Once the command has completed, the shell from which you ran the command will return to its normal input prompt, and you should receive an email from *backup@example.com* with a "Subject:" header that includes the phrase, "AMFLUSH MAIL REPORT." This email should state, "The dumps were flushed to tape [$TAPE_NAME]," where [$TAPE_NAME] is the name of the tape you loaded in the steps above. If the mail claims there was an error during the flush process, notify Mark or his superior immediately and forward the person contacted a copy of the email containing the error.

2. Once step 1 has been completed, it's time to prepare for tonight's backups. In the email you received in step 1 (if the last night's backups finished successfully), or the email you received in step 1.h (if there was a problem with last night's backups), there was a line that contained the phrase, "Tonight's dumps should go onto 1 tape: [$TAPE_NAME]." Use the procedure defined in steps 1.a through 1.f to load the requested tape, as specified by [$TAPE_NAME].

3. At 1500 hours (Pacific), you should receive another email from *backup@example.com.* This email should either state that the proper tape for tonight's backup has been loaded or that the incorrect tape is still in the drive. If the email states that the incorrect tape has been loaded, retrace your steps from 1.a through 1.f to verify that you loaded the right tape. Make sure you've loaded both the correct magazine and the correct tape from that magazine.

   If, after retracing your steps, you find that you would have loaded exactly the same tape, notify Mark or his superior immediately and explain the problem and the steps taken to resolve the problem. Forward a copy of the pertinent email to the person you contacted as well. Chances are the tapes in the magazine are not in their proper order, or someone has placed tapes from another magazine in that one.

With the document written, I needed to make sure it made sense, particularly to the people who would use it. First, I tested the procedure myself several times, to make sure I hadn't left out any needed information. I knew my testing would go fairly well, because when I write procedural documentation, I perform the steps as I'm writing. I've long since learned that my memory is, at best, faulty; when I can easily verify some memory by an action, I try to do so.

Then I printed a few copies and handed them around to individuals from my target audience. I explained the purpose of the document and asked them to read through it, marking each word, phrase, sentence, or section that confused them.

Once they'd done that, we sat down together to go over each of their marked-up documents. I made copious notes about each point of confusion. I was, in effect, allowing the end users of my documentation to play the role of editor.

One person suggested I provide an at-a-glance list of the steps, with references to the more verbose explanation for each step. Another mentioned that the description of how the tape magazine was to be inserted into the jukebox wasn't very clear. (Descriptions of physical actions are sometimes difficult to write unambiguously.) Another person complained that not all possible backup system states were covered in this document. I explained that the purpose of the document was not to cover all possible scenarios, but only those that seem to occur with some regularity in our environment—those likely to need attention by those gathered in that room. A document covering every possible system state would be quite a bit larger, and somewhat unwieldy as a day-to-day task guide. Perhaps a separate document covering this issue should be written someday. In the meantime, I added a blurb at the top of each page of the document, in boldface type: "If you encounter a situation that isn't covered by this document, contact Mark via email (*mark@example.com*) or phone (408-211-2111)."

With feedback in hand, I returned to my desk and reworked the initial document, incorporating the end users' comments and suggestions and addressing their complaints. Once I'd finished the rewrite, I went through another iteration of the feedback process. This time, I not only asked the reviewers to follow the procedure and make notes of any confusing or incorrect areas in the document, I also observed them as they worked their way through the procedure, to insure that I caught any problems they might not write down.

Finally, I asked someone from HR to review the document. He caught several grammatical and formatting errors which made the document more difficult to read than necessary; neither I nor the admins I had review the document thought these particular issues were serious, but it's quite possible that we've become inured to poorly written documentation. I incorporated his feedback as well, and was ready to loose my document upon the world!

Getting feedback prior to implementation is essential. It avoids otherwise costly mistakes, particularly when the document in question is a procedural document dealing with production infrastructure. Just as essential is monitoring the use of the document closely for the first few days or weeks, and instructing your readers to come to you with questions if anything at all seems unclear, rather than attempting to implement the confusing directives. This provides you with yet another round of feedback; a real-world implementation often uncovers problems people don't think about when just reading through or testing a procedure. As I'm sure you know, reality intervenes at the most inopportune times. I can almost guarantee something unforeseen will occur in the early days of your document's use in a live environment. Writing a procedural document is much like writing a program. In both cases, it behooves everyone to attempt to account for all possibilities, even the supposedly impossible ones. And in both cases, at least one real possibility is invariably missed.

After a week with the document in use, I incorporated the last feedback—derived from experiences using the procedure in the production environment—and gave my team the final version of the document. Since then, the backup system has been relatively problem-free, and the number of questions hitting my desk about the backups has decreased dramatically, allowing me to focus on other things. This is perhaps one of the greatest benefits of documentation. Much like automation, it allows you to streamline your environment and free yourself from trivial, repetitive tasks.

This document forms the basis of one section of a run book, as described earlier. The same procedure could be used to flesh out other sections of such a run book, creating an entire system maintenance document for this particular machine.

8. Conclusion

I hope this booklet has helped you understand the need for, and various approaches to, documentation in the field of system administration. I've attempted to define the various categories of documentation you may encounter or need to create in your environment. I've provided you with guidelines for determining the quality of a document. I've outlined a procedure for assessing your current state of documentation. I've provided a process by which you can begin to tackle your own documentation projects, along with some brief examples of the types of documentation that exist in our field. Finally, I've walked you through a case study of document creation.

This booklet is by no means the final word on documentation. What I've included is meant to illustrate the breadth of the topic; any single type of documentation is itself worthy of its own book. I encourage you to explore your options for documentation and to come to the fullest possible understanding of your documentation needs before starting an all-out documentation effort. Careful assessment of what your environment demands in terms of documentation will allow you to strike a balance between effort and reward, investment and return. Documentation works best when it offers sufficient content to reward its users, yet can be created and maintained with minimal effort.

9. Resources

Automated Network Discovery Tools:
- HP OpenView: *http://www.openview.hp.com*
- IP Sonar: *http://www.lumeta.com*
- Tkined: *http://wwwhome.cs.utwente.nl/~schoenw/scotty/*

Databases:
- MySQL: *http://www.mysql.com/*
- PostgreSQL: *http://www.postgresql.org/*

Documentation Policy Examples:
- *http://whpo.ucsd.edu/policies/doc.htm*
- *http://www.debian.org/doc/docpolicy*
- *http://www.ucar.edu/ncab/Policies/documentationpolicy.html*
- *http://web.library.uiuc.edu/ahx/documpol.htm*

Modular Documentation:
- Ament, Kurt. *Single Sourcing: Building Modular Documentation.* Noyes Publications, 2002.

Obfuscated Perl Contest:
- *http://www.sysadminmag.com/tpj/obfuscated/*

Revision Control Systems:
- CVS: *http://www.cvshome.org/*
- Documentation on using CVS: *http://citeseer.nj.nec.com/326583.html*
- RCS: *http://www.gnu.org/software/rcs/rcs.html*

Usability and Design:
- Jakob Nielsen: *http://useit.com/*
- Don Norman: *http://www.jnd.org/*

Web-Based Documentation Tools:
- Bugzilla: *http://bugzilla.org/*
- Nagios: *http://www.nagios.org*
- RT: *http://www.bestpractical.com/rt*
- Wiki: *http://c2.com/cgi/wiki?WikiWikiWeb*

About the Author

Mark C. Langston has been a professional jack of all trades for the past fifteen years. During that time, he's run mail systems, dealt with data retention issues, worn the white hat of the security trade, debugged routers, designed and maintained naming systems, written firewall rulesets, programmed, soldered, punched down, spliced, crimped, changed tape reels, replaced disk packs, interviewed, evangelized, bled, chanted, danced, pleaded, and connived, all in the name of Keeping Things Running Smoothly(™). And through it all, he's tried to find the time to write things down, so others could have some inkling of what he's done.

Mark holds several degrees in experimental cognitive psychology, into which he fell after deciding education was its own end, rather than a path to employment. During his academic career he studied artificial intelligence and artificial life, and the memory processes involved in comprehension. He's published numerous papers on the latter.

Mark has worked for a broad range of employers, from one of the largest power companies in the United States, to small R&D departments of consumer electronics manufacturers. He's worn many hats, from lab assistant to trainer to senior sysadmin, from consultant to Chief Technical Officer. Mark has been an invited researcher at the German Institute for Artificial Intelligence in Kaiserslautern, Germany, and he's been an invited guest on TechTV's "The Screen Savers" television show.

Mark is currently the Senior UNIX Sysadmin for the SETI Institute in Mountain View, California.

Mark lives in San Jose with his loving partner and cat, both of whom pretend to tolerate his obsession with computers, if only because it feeds their insatiable appetite for sushi. When Mark isn't juggling his workload, he's working on his five-ball cascade and four-club shower.