The following paper was originally published in the
Proceedings of the Eleventh Systems Administration Conference (LISA '97)
San Diego, California, October 1997

For more information about USENIX Association contact:

1. Phone: 510 528-8649
2. FAX: 510 548-5738
3. Email: office@usenix.org
4. WWW URL: http://www.usenix.org

An Analysis of UNIX System Configuration

Rémy Evard – Argonne National Laboratory

ABSTRACT

Management of operating system configuration files files is an essential part of UNIX systems administration. It is particularly difficult in environments with a large number of computers.

This paper presents a study of UNIX configuration file management. It compares existing systems and tools from the literature, presents several case studies of configuration file management in practice, examines one site in depth, and makes numerous observations on the configuration process.

Introduction

Systems administration is hard, and is getting harder. This may be the computing world's single biggest problem. There are certainly others: security, privacy, improving performance, standards enforced by potential monopolies, the year 2000, etc.; the list can go on and on. But none of these matters if computers aren't useable in the first place.

In our modern distributed systems, each desktop is becoming increasingly more powerful and is expected to provide more and more functionality. Borrowing a metaphor from Rob Kolstad, the ''service knob'' is being cranked up, and systems administrators and users are paying by spending more time configuring systems, installing software, tuning networks, fighting fires, and trying to convince the environment to just work right. In the corporate world, this problem is usually referred to as part of the ''total cost of ownership'' or TCO, and it is a growing concern.

Simply stated – it is difficult to keep a computing system up to date and performing correctly. This has traditionally been the role of the systems administrator, and, as the requirements for computers continue to grow, the systems become more complex to administer. It is imperative that we make systems administration easier.

In computer science and engineering disciplines, complexity is often managed by abstraction. For example, source code is organized into functions, procedures, or objects with well-defined interfaces. Information is stored in data structures, allowing algorithms to be developed to manage abstract data structures. Abstraction methods are often used in systems administration as well. We often create a set of scripts or a tool for performing some particular function. As evidenced by the growing complexity in our field, we need to investigate more powerful abstraction mechanisms.

The work in this paper is part of an ongoing project to understand the underlying principles of systems administration. It is hoped that a deeper understanding will result in tools and methods that can be used to build stronger abstractions, and in new administration models that help to reduce the complexity of managing large and diverse sites of all different types.

The particular area discussed in this paper is that of operating system configuration files – the files in a UNIX system that control how the operating system and its constituent services perform. Classic examples are /etc/passwd, root's crontab file, and /etc/inetd.conf. The number of files configured on any particular system varies dramatically from one site to another and one architecture to another, but can range from a small handful to perhaps a hundred. Ultimately, these are the files that determine who can use the machine, how it can be used, and what takes place on it.

These configuration files are a good area of study because they are relatively simple but can lead to complex issues. They are quite well understood at the single-system level, but they require a very carefully planned strategy in a network of several thousand hosts. Each configuration file is a self-contained problem; but the files are typically grouped together, making them a choice candidate for an abstraction that encapsulates all configuration management in a system. In understanding how configuration files are created, managed, and distributed at a site, one will typically have to understand the site's management model (and, often, the political intricacies). In this way, configuration file study becomes a platform for understanding the other aspects of systems administration.

The goal of this study is to understand the operating system configuration process and the problems associated with it, to look at how different sites have approached this problem, and to consider various abstractions for managing the configurations of multiple hosts.

Although the problem of the complexity of systems administration spans all different types of computers, organizations, and management approaches,

this study was limited in scope in order to make it feasible. The discussions in this paper are principally applicable to heterogeneous networks of UNIX machines.

Configuration Management Background

Configuration file management is not a new topic to the systems administration community. Yet, despite multiple papers on the topic, there does not yet appear to be a commonly accepted approach to building new machines, configuring existing systems, or managing the files used in the process. While this may be a problem for systems administrators, it also means that there is a wealth of information from which to draw potential solutions.

As part of the background for this study, I spent some time reviewing the history of configuration systems. A detailed discussion of this review is in itself quite interesting but beyond the scope of this paper. A quick summary, however, may be help set the context of the study.

Interest and work in this topic dates at least as far back as the days of LISA I (all the way back to the Reagan years), when Ken Stone [Stone] presented a paper that described HP workstation disk cloning, making initial modifications with sed, and then performing later updates with rdist. Ironically, nearly the same method is used today in several very large sites.

Over the next several years, Sun Microsystems' NIS [NIS] became more widely used, due in part to the 1991 publication of the book *Managing NFS and NIS* by Hal Stern [Stern]. Other solutions from vendors appeared, including the Tivoli Systems *Management Environment* [Tivoli].

Several configuration systems and cloning scripts were detailed in various LISA proceedings. Then in 1994, in LISA VIII, the community nearly exploded with four configuration systems:

- Anderson's lcfg [Anderson]
- Harlander's GeNUAdmin [Harlander]
- Imazu's OMNICONF [Imazu]
- Rouillard and Martin's Config [Rouillard]

Each of these is quite different, but they share some interesting similarities. First, each grew out of a need for a more powerful tool than was currently available to the author. Second, each maintains a central description or database of the configurations that should be installed on individual hosts. I recommend that scholars in this area examine Anderson's paper for an excellent summary of the state of host configuration at this time.

In following years, configuration systems were used in increasingly sophisticated ways, or perhaps more accurately, were seriously discussed as a part of other processes for the first time. Shaddock and fellow authors [Shaddock] discussed a use of their sasify system to do a massive upgrade of 1500 workstations. Fisk [Fisk] the rather hazy barrier between machine configuration and software distribution, and described a system that tackled both areas as part of the same problem.

The general approach taken by the administrative community over this time period has been to develop a host cloning process and then to distribute updates directly to hosts from a central repository. The diversity of solutions developed illustrates that this is a basic problem for many sites with – as is not surprising – a wide range of requirements.

Site Case Studies

During the past two years, I moved from a site where we had rigorous configuration management to a site that had ad-hoc methods of keeping machines up to date with good informal methods but no formal structures in place. The difference between the two sites struck me as remarkable. This was one of my primary motivations for examining configuration files in detail.

Initially, I thought that my new site would be much more difficult to manage at the host level, requiring a lot more hands-on management, but that was usually not the case. Instead, the differences were really about how easy it was to manage the entire environment.

At the first site, it was easier to delegate management of machines to different people, because no single person had the configuration of an architecture in their head: it was all kept in the central configuration files and build scripts. Global changes such as an inetd replacement or a new shell could be easily performed, and so they often were, making for a rich environment.

On the other hand, at my new site, it was much simpler to handle new architectures, because there was no overhead in assimilating them into a global system. One simply set up the machine, tweaked it until it worked, warned new users that it had a minimal environment, and then left it alone. This resulted in more flexibility at the host level and less in the larger environment.

Intrigued by these differences, I started to talk to administrators at other sites to learn how they handle configuration management. During the past year, I've talked about the issues with approximately thirty different groups. These studies were informal, usually occurring as a series of conversations on the phone, around a whiteboard, or over lunch.

I present a summary of a number of these discussions here in order to impart a general idea of the range of the sites and strategies. These sites are not intended to be representative of the industry as a whole; a far larger study would be required for that. Instead, they provide insight into how other sites do configuration management, and the general state of systems administration at a number of different sites.

All sites and participants have been kept anonymous except for Northeastern University. I worked there and played a large role in the design of its systems, and feel that I should acknowledge my own role in the evaluation of its environment.

Case Study 1 – Northeastern University

The College of Computer Science at Northeastern University runs a network of approximately 70 Suns, 40 Alphas, 50 PCs running Windows variants, 50 Macintoshes, and a number of special purpose UNIX machines. These are managed by a central administration group that is responsible for all aspects of the technical environment.

At NU, a new machine is built by installing an operating system from media, following a set of instructions to configure it, and then applying modifications from the network. NIS is used to coordinate most of the files that it can support. All other configuration files are maintained in a central location under RCS. The configuration directory is NFS exported to all hosts. Machines are updated manually by using a homegrown system based on a root rsh mechanism from a central server that then installs the correct file onto that host. A number of tools have been built around this mechanism to automate the distribution of files. In general, changes to local machines are kept to a minimum through this mechanism, even though several machines have very different configurations from others (in part because the central repository is able to store different configurations for different machines).

When the system was first installed, it solved a number of important problems. Over the years, the environment has grown more complicated. The administrators have identified new requirements for the system, such as keeping changes to the OS and other configuration information on all client machines in sync, even when machines are down temporarily. This is especially important to them in order to keep all machines current the latest vendor security patches. They expect to completely rework the system soon.

Case Study 2

Site 2 is a computer science department with about 70 UNIX-based computers. The majority of these are dataless machines (with just swap on the local disk) that get their filesystems over the network from an Auspex NFS server. The remaining computers are SGIs, and can't boot from the Auspex, so use their own local disks and a centralized /usr/local-like scheme.

If a machine is a client of the Auspex, building is pretty simple: an additional set of directories is created for that machine, and it is configured to netboot from the Auspex. Changes to those machines' configurations are done on the Auspex, by editing the file directly on the Auspex's file system and then copying changes to the other clients. RCS is used for change management of key system files.

The other machines in the department are set up individually, each by hand. If changes need to take place to them, the administrator logs in and makes those changes. There is some expectation that this method won't scale to large numbers of machines, but that's not seen as an important issue at this time.

Case Study 3

Site 3 is a large Fortune 100 corporation. There are many groups within the company who are responsible for different parts of the infrastructure. The particular group that was interviewed is responsible for the environment for a large development and engineering segment of the company. The set of machines that they are responsible for includes 1000 Sun workstations and 5000 X terminals. In addition, some people in the group have responsibility for other architectures within the company including HPs and SGIs. The approach to managing these other computers is completely different than the management of the Suns, and was not discussed during the interview due to a lack of time. The group is responsible for the operating system and the applications on the Suns, but does not manage the network, or some networked applications like email. Their users and the machines they manage are in multiple locations spread around the world.

This group has divided their computers into small modules, each consisting of a machine for general logins, an application server, and a number of compute servers. Users are associated with one particular module and use X-terminals to access the resources.

The machine configurations are kept on a set of master hard drives. New machines are cloned from these hard drives, and then the initial boot-up script asks a series of questions in order to initialize the host. The master configurations are rigorously maintained, with files documenting all changes kept in critical directories.

NIS is used to distribute password, group, and netgroup information. In order to scale NIS to their environment, the group rewrote the NIS transfer mechanism to introduce another layer of hierarchy. Some files, such as /etc/passwd, are pushed out using an rdist mechanism, while other files, such as /etc/printcap, are maintained largely by hand and distributed by complex scripts to each system architecture. OS patches are kept on designated OS masters, with changes tracked in RCS control files. These patches are distributed using a combination of rdist and ftp.

In some cases, the group has to do direct hands-on management. Notably, the administration group had to install the 5000 X-terminals and configure them by hand because of security concerns and an SNMP bug.

The group is having troubles with the disk cloning strategy because of an increasing number of variables: operating system versions, different sizes of disks, different types of computers, and, most importantly, organizational changes.

Case Study 4

Site 4 is a growing company currently expanding to multiple campuses. The primary UNIX computer users are engineers using CAD applications. A central administration group is responsible for all aspects of managing the computing infrastructure, and is divided into several different groups with separate areas of authority, such as UNIX, PC desktops, and networking. The environment consists of 1500 Intel-based PCs and 900 UNIX machines, most of which are Suns running Solaris 2.5.x. There are also a dozen HPs and SGIs that are managed independently by specialists within the UNIX group.

The Suns are built by using Sun's JumpStart [JumpStart], which solves the build and initialization issues. The group uses NIS to manage password and other changes. Further configuration of machines almost never takes place, due to a very strong emphasis on centralized servers. When changes do need to take place, they are pushed out from a server using a script wrapped around rdist, which takes advantage of clever hostnaming conventions in order to make decisions about what hosts to affect. Central files are not kept under revision control, but backup copies of critical files will typically be maintained.

The group uses approximately 30 servers to support the 900 Suns. Those servers are managed in a more ad-hoc way, with a lot of hands-on modification of configurations, primarily because the servers span a wide range of services and hardware.

Interestingly, the smaller HP and SGI environments are managed in a much looser way, with individual host configuration typically taking place directly on the host. Thus, the centrally managed approach of the Suns comes from a need to manage on a large-scale, not from a mandate from management.

The group anticipates that the next operating system upgrade may be very difficult and, despite the fact that machines are well behaved in this system, is nervous that things are on the verge of getting complicated.

Case Study 5

Site 5 is a small university department serving a combination of computer science and art graduate students. Their network consists of some thirty SGIs, a couple of Suns, and a scattering of Intel-based and Macintosh personal computers. The direction of the infrastructure is determined almost entirely on the availability of funding and the need for project development and demos.

Each of the SGIs is generally built from CD-ROM or by doing a byte-for-byte copy of the system disk of a previously built system. Since performance for demos is a big concern, patches are applied very sparingly, and considerable work is done to verify that the vendor patches do not break or slow down existing code.

Each machine is individually maintained by hand. This approach is taken to avoid having a central machine that, if compromised, would allow for easy compromise of others. In this dynamic university environment, security is a big issue. Each systems administrator has an individual root account in /etc/passwd on a given machine. Various people in the environment, beyond the system administrator, have root access to selected machines in order to facilitate research and development by installing software, changing kernel configurations, and permissions for /dev devices.

NIS is used to allow department-wide logins. The system administrators of this network control their NIS maps, but send email to another group for updates to their DNS tables. It is felt that the time required to set up a DNS server would be better spent on immediate pressing issues.

Any systems other than SGI are maintained fitfully or not at all; attention is given to them only in the case of a particular user need or security incident.

Backups of system areas of critical machines are performed, but users are expected to back up their own files as the user deems necessary. DAT tape drives are provided in public areas for this purpose. There is no change management for configuration files; copies of relevant files can usually be found on similarly configured machines.

The administrator of this environment is well past the point where he can keep up with all of the changes that need to take place.

Case Study 6

Site 6 is a financial company that is spread across several cities. As with many large sites, the infrastructure is managed by several different groups, who are divided both according to function (i.e., networking) and according to company directions (i.e., all activity based around one type of interaction with clients). The focus of this study was a part of their computing infrastructure used to build, maintain, and run one particular application, where uptime during business hours is the prime directive. This environment consists of about 350 Suns, all running Solaris. 200 of these are used for running the application, 100 of these are development and support machines, and the remainder are servers of various types.

NIS is used within this environment to deliver passwords, automount maps, and some special maps used by administrative applications. NFS is barely used, because of the importance of minimizing dependencies.

The application machines are critical and are carefully controlled. They are built either from JumpStart or from a cloned disk that boots up into an interactive initialize phase. The developer machines are less carefully managed, and will typically be built by hand. The servers run on a number of different types of Sun hardware and have all been custom-built. The group uses an internal web page to maintain a checklist of things that should be done when building a machine. Over the past year, one of the group's major projects has been to get the servers and the developer's machines ''rationalized'' or similarly configured.

Each machine has a separate root password, and there is no centrally authoritative machine. However, the group uses a ''master root cron'' mechanism to achieve the same effect. Every half hour, the cron job checks to see whether there is a new crontab available on any of several replicated NFS servers. If so, it is copied in as the new crontab, which is, of course, executed as root. The group uses this mechanism to install carefully crafted patches, to update configuration files, and to make global changes as necessary.

The group is pretty happy with their system. Other than the dependence on NFS for some central functions, the environment is quite failsafe and reliable. There is some dissatisfaction with the server and development environments, but those are being fixed during the reconciliation process. The hardest problem they have is finding all the machines.

Case Study 7

Site 7 is a research lab with an emphasis on computational science. The infrastructure consists of several supercomputers, a UNIX-based workstation network with over 100 UNIX machines of many different types, a growing number of PCs, and a production network based on ethernet and ATM. Most of the infrastructure is managed by a central group, with some of the experimental labs being managed by individuals focused in that area. This study focused on the machines managed by the central group.

There is one NIS server for the department, and all machines are a member of the NIS domain. NFS is the primary remote file system in use, although AFS and DFS are used minimally. The build process for a new machine depends on what type of computer is being built, but the group is working to standardize methodology. Typically, one will install the operating system onto a machine from CD-ROM, then follow written instructions to get the machine onto the network. After that, a script applies relevant patches and makes changes to the local machine.

Many changes are handled through NIS, but occasionally changes must be pushed out to all machines. When this happens, the group generates a list of machines and then does an rsh from a central server to push out the changes. Until recently, no precautions were taken to check for machines that were down, or to use revision control on the sources of the files. Some of the machines in the environment are special purpose or specially configured, and the set of machines is constantly moving and being reconfigured, so a hands-on approach was the simplest to develop.

This approach resulted in a somewhat inconsistent environment and was too difficult to use for all but the most serious modifications, so individual hosts weren't tuned often to match new requirements. When they were updated, the build scripts weren't necessarily changed to reflect that update, so machines built after the change might or might not have that new change.

The group is moving to a centrally managed set of configuration files, and a standard mechanism for installing new hosts based on these files and centralized sets of OS patches. There are two main concerns with this system: first, it must support individual machine idiosyncrasies, and second, it must be able to handle machines that are down or disconnected from the network.

Case Study 8

Site 8 is an engineering department in a university. A lot of the administration work is done by students, and the policies and procedures reflect this. The large environment consists of many different types of UNIX machines, including BSDI, NetBSD, Solaris, SunOS, Alphas, HPs, and some Windows boxes and Macintoshes added for flavor.

Many machines are built by students by hand. Others are built by doing a network boot and then getting the latest set of modifications.

A set of HPs is used for most of the central management. The HPs use both NIS and rdist to distribute files into the environment. In many cases, the source files are built by using either Perl or m4 macros, because the environment is complicated enough that the source files are hairy. The rdist files are built using gnumake, and the source files are kept under RCS. They've found that, because of the number of new students they work with, detailed logging is important.

This is a rather complicated system, and one of the most difficult tasks is to incorporate new architectures into it. The administrators would also like the ability to put comments and documentation within the source of files, and feel the need for a comprehensive database of hosts.

Case Study 9

Site 9 is a research lab with a focus on computer science. The UNIX environment consists of about fifty DEC Alphas running Digital UNIX, along with a few SGIs.

Builds of new machines are done by using a customized version of the Digital UNIX install process. It builds the local machine, makes some modifications, and then invokes an rdist on the machine to add files

from a central collection. The administrators can build a number of machines simultaneously, spending only about five minutes per machine. The entire process takes about two hours.

The site uses a rdist system to manage configurations on all of the machines. It is used to push out aliases, fstab, automount files, printcap, and others. The rdist scripts are run nightly, and not invoked directly by the administrator. Maintaining the list of target hosts for rdist is one of the bigger problems. The files that are pushed out are generally maintained by hand, although some of them have special rules that are applied on distribution. For example, fstab incorporates any fstab.local it finds on the target machine.

The site does not use NIS, so all password changes must take place on a central machine. New accounts and password changes are pushed out using the rdist system.

The administrators aren't particularly happy with the mechanism, although it works. Among other things, they would like to see a pull mechanism rather than a centralized push. The system has been in place for quite some time, and given the staffing levels, they are unlikely to be able to change it for some time.

Case Study Observations

As I mentioned above, this sample set is too small to generalize to the entire industry. Nonetheless, a number of interesting observations can be made, some of which may help to understand what is needed in a stronger abstraction method.

- Almost every site uses NIS, although some use it to distribute for only a few maps, while others used it for every map intended.
- No site uses NIS+, not even the Sun-only sites.
- No one seems to settle into a definitive way of doing things on every host. Most sites have more than one method that they use for building machines and more than one method to configure them. Site 6 is a good example of this; they use JumpStart in some cases and disk cloning in others (once they even participated in a race between two administrators who favored different approaches). Site 7, while having a build script for some architectures, doesn't have a build script for others. some architectures, didn't have a build script for others.
- The large sites typically have a very controlled method for managing most of their machines, and a more ad-hoc method of managing their servers. The way they manage their thirty servers often is similar to how a thirty-machine site manages its entire environment. This fact has some very interesting ramifications.
- Centralized management and automated building of some type or another are done at nearly every site with fifty or more machines. In the cases where this isn't true, building is done by giving a cd-rom and a set of instructions to a student (which is nearly the same as automated building).
- Once a site has a strategy, it is stuck. Whether the staff have invested heavily in one vendor's build mechanism (like JumpStart), or an Auspex, or a management scheme, they find it very difficult to move beyond the restrictions imposed by that scheme. This is one reason that a major OS upgrade or the installation of a new architecture is so hard. Not only must the administrators learn the nuances of the new system, they have to modify their existing practices in order to support it.
- At some sites, changes take place constantly, at others rarely. This appears to be a function of a number of variables: how comprehensive the build process is, the ways in which the machines are used, and the need for the environment to stay modern. Schools and research labs seem to have more dynamic environments, while corporations seem to focus on supporting one type of application and then not changing once the application works well.
- A surprising number of people feel that keeping track of machines is the hardest problem they have with configuration. If a complete list of machines could be generated, it would be much easier to keep them up to date.

Site	Environment	Build	Configure	Revision
1	100 various	Media + Script	NIS, file push	RCS
2	70 dataless Suns	Copying	NIS, edits	RCS
3	1000 Suns	Disk clones	NIS, rdist	-
4	900 Suns	JumpStart	NIS, rdist	.bak
5	30 SGI	Media	NIS, edits	-
6	350 Suns	JumpStart / Clone	NIS, cron copies	-
7	100 various	Media + Script	NIS, edits	-
8	100 various	Media + Script	NIS, rdist	RCS
9	50 Alpha	Digital UNIX install	rdist	RCS

Figure 1: Environment attributes.

- Some sites differentiate between the build process and the configure process. Others don't touch a machine after building it, except in extreme cases, while still others don't have any more formal build process than a set of notes from the last time they did it. Again, this comes down to what the computers are being used for.
- Everyone in the survey who has an update system uses a push mechanism. In some cases, the hosts pull down the files, but that pull is initiated from some central spot. No one is doing an explicit pull, where the action on the host is initiated by the host or the user on the host. (This may come from the fact that I always spoke with an administrator who was part of some kind of central support organization, not with a user who was responsible for their own machine.)

Also, a few notes of non-technical nature:

- Everyone has a different definition of what "server" means.
- Nearly everyone feels overworked and said something similar to "I've been too busy to take the time to go back and fix that."
- If you've seen one machine room, you've seen them all. But it's still a lot of fun to see them all.

An In-Depth Look at One Site

For four years, Northeastern University's College of Computer Science (Site 1 in the above section) has been using a central configuration mechanism to manage most of its files. I have studied the files in this system in some depth in order to understand what was being changed and how often those changes took place.

A bit of background on the NU configuration system will be helpful. The system is based on a central NFS repository, where all UNIX machines, regardless of architecture, retrieve their files. Multiple copies of a single type of file can be kept, with specifications based on hostname and architecture type. So, for example, if a sun4 named "sol" were to look for a

File	Different Versions	Revisions	Type	File	Different Versions	Revisions	Type
amd	2	1-2	admin tool	svc.conf	2	1	OS
cops.cf	3	1	admin tool	syslog.conf	8	1-7	OS
etherdown	1	1	admin tool	termcap	2	3	OS
newsyslog	2	1-3	admin tool	ttys	2	1	OS
rotlogs	1	1	admin tool	ttytab	10	1-2	OS
staticroutes	1	1	admin tool	rc	5	2-3	OS bootup
sudoers	1	1	admin tool	rc.local	11	4-14	OS bootup
super-users	3	50	admin tool	rc.priv	25	1-17	OS bootup
watchmerc	11	1-4	admin tool	hosts.lpd	1	1	printer
crontab	7	1-4	cron related	printcap	10	1-10	printer
daily	16	2-26	cron related	aliases	1	2	service
hourly	5	3-11	cron related	ftpusers	1	1	service
monthly	4	1	cron related	hosts.allow	9	6-16	service
weekly	10	2-11	cron related	hosts.deny	6	3-6	service
bootparams	1	1	OS	lbcd	1	1	service
bootptab	1	5	OS	mrouted.conf	4	1-6	service
exports	3	1	OS	ntp.conf	4	3	service
format.dat	2	2/3	OS	resolv.conf	6	1-4	service
fstab	1	1	OS	sendmail	2	1	service
group	4	1-2	OS	sendmail.cf	2	2	service
hosts.equiv	1	1	OS	zshenv	1	5	shell
inetd.conf	11	4-15	OS	profile	1	5	shell
magic	1	1	OS	profile.bash	1	4	shell
nis	1	6	OS	tcsh.cshrc	1	6	shell
passwd	45	1-10	OS	Xconfig	1	1	X config
securenets	1	3	OS	xlogin	1	1	X config
securettys	3	1	OS	Xsession	1	1	X config
services	4	1-2	OS	Xsetup	1	1	X config
shells	2	1-3	OS	Xstartup	1	1	X config

Figure 2: Northeastern's config files.

passwd file, it would first select the file ‘‘passwd.sol’’ if it existed. If not, it would select ‘‘passwd.sun4.’’ If that didn’t exist, it would copy ‘‘passwd.’’ This mechanism allows the administrators to set up defaults for the system, override those for specific architectures, and then override those for individual hosts. Thus, if one file will suffice for the entire system, there will only be one copy of it in the repository.

I’ve grouped these files by their function as I perceive them. For each file, I’ve noted two pieces of data:

- ‘‘Versions’’ is the number of different copies of that file are kept in the repository (so for the above example there would be three copies of the passwd file).
- ‘‘Revisions’’ is the number of times that file has been modified during the last four years. Because each version of the file might have multiple revisions, I’ve given the range of revisions.

The files are listed in Figure 2.

Some of the entries in the figure require some explanation or deserve some comment:

- After five revisions, bootptab has moved out of the config system because it is being autogenerated from a database of hosts.
- exports and fstab are in the repository but aren’t actually distributed by the system. Instead, these are managed by hand on all hosts.
- group is the copy of /etc/group with NIS hooks in it.
- passwd has an enormous number of different versions. All differences amount to which netgroups are in which files, since this environment has restrictions over who can log in to which machines. Even with this number of files, it is possible to change the root password everywhere by running a sed script to change all of these files, and then typing "pushfile /etc/passwd".
- aliases and ftpusers are no longer used, since they are maintained as part of the central mail and ftp servers.
- hosts.allow and hosts.deny are part of the tcpd program, which controls access to various ports. These files have been changed extensively.
- sendmail is the actual sendmail binary. This is an interesting change for the config system, which, other than this file, is used only for ASCII files.
- amd is a script used to startup and shutdown the amd automounter.
- etherdown, rotlogs, and staticroutes are homegrown utilities.
- super-users is a file used to list who can have super-user access on a machine. At one point, there were many more versions of this file than just three, but the differing copies were recently eliminated.
- hourly, daily, weekly, and monthly are scripts executed by the root crontab at the frequencies you would expect. These are typically used to do maintenance on various kinds of servers, such as rotating logs, cleaning up tmp, and so on. The high number of daily and weekly files reflects the number of machines running customized services.
- rc, rc.local are used on suns to replace /etc/rc and /etc/rc.local.
- rc.priv is an augmentation of /etc/rc.local that is typically host specific, often used to start services on that particular machine such as a web server. Again, the high number of these reflects the number of machines running customized services.
- tcsh.cshrc, profile, profile.bash, and zshenv are the central files used to control all of the shells in the environment. These are changed only to make major changes to the default environment. Most PATH changes and other modifications are set using a different system.
- The various X* files are part of the Alpha CDE environment, and have a small number of revisions because they are newly added to the repository.

From this information, it is possible to make some statements about the role of the configuration management scheme at Northeastern. It is not clear how many of these observations will be relevant to other sites, but they nonetheless provide some insight into what kinds of patterns may be observed in the configuration of a network.

- A file that only has one copy and a small number of revisions is typically a file that was shipped with the OS but changed just a bit. The build process copies in the changed version of that file.
- The repository is used to change existing OS files, to add new files to the OS, to distribute a binary, to distribute scripts, and to distribute local configuration files for various processes. This is more than a simple configuration mechanism, but less than a complete solution, since it doesn’t really handle software distribution, operating system patches, and other types of modifications. The system seems to have evolved into a mechanism for pushing out ‘‘things on individual machines that change or need to be kept locally.’’
- A number of files in the repository aren’t distributed to hosts anymore, and some never were.
- It is not possible to gather this information from the table, but is worth mentioning that over twenty different people made modifications to the files. The system uses RCS for change

management, and this seems to have worked.

- A very small number of files had changes that were made and then reversed. RCS was used to detect a problem and move back to the last known good configuration. This does not seem to have been a common occurrence.
- At a first glance, it's difficult to generalize the number of times a file is likely to be changed. However, there are some distinct patterns. Files like super-users and passwd were changed quite often, due primarily because of changing roles of users. New students would arrive and be given root privileges, new machines were purchased on grants and had to number of people who could use them had to be constrained. In these cases, the files have some direct relationship to the role of the people who use the computers, so they changed more often than files that didn't.
- Another type of file that changed relatively often were those relating to some aspect of the physical environment, such as printcap, which had to be updated every time a new printer was purchased or an existing printer was moved.
- Files relating to the purposes of a specific machine, notably inetd.conf and the daily cron jobs, tended to have a high number of different versions, with some of those files having a high number of revisions.
- For the most part, the number of times that files change over the course of four years is pretty small. This may imply that emphasis should be put into the process of building a machine and getting it up to date with respect to the rest of the environment, rather than working on the process of distributing files.
- Some files aren't in this repository at all, or were in it but then their functionality moved away. This reflects a move toward centralizing a service, or encapsulating the functionality of the file into a different system. For example, sendmail.cf and aliases were minimally updated because major changes took place on the central mail hub. (In contrast, the aliases file on the hub has been updated 961 times during the same period.) Files like group, passwd and bootparams are generally updated via NIS. The central files for the shells are only updated minimally because they use another systems to set global paths (which has been updated 62 times). /etc/motd is not updated via the configuration system because it is never changed. Instead, the group uses a msgs-like system to make announcements. In every case, the move to a centralized system made the function provided by that file easier to manage and support.

Finally, to complete the story, the primary maps that are distributed via NIS at Northeastern are listed in Figure 3. In this figure, the ''revisions'' column also refers to the number of times the file has been updated since 1993. A few entries are worth describing in detail.

- The amd.* files refer to various automount maps used by the AMD automounter.
- The amd.home file has a line in it for each user of the system, specifying the mapping of their home directory.
- The amd.net and amd.home are actually in their

File	Versions
amd.ftp	5
amd.home	3241
amd.net	132
amd.proj	127
archtree	4
bootparams	3
ethers	2
group	415
hosts	4
netgroup	2000+
netgroups.aux	3
netmasks	1
networks	1
passwd	3913
protocols	1
publickey	1
rpc	3
services	22
ypservers	38

Figure 3: Primary maps at Northeastern.

second and third major releases, using a feature of RCS that lets one change major version numbers. This facility was used to mark major changes in the environment. The revision number given is a sum of all changes in all releases.

- The netgroup file is created using a script that collates several source files, some of which come from a database. The actual number of revisions is very difficult to calculate, but the number of revisions to the source files is approximately 2000.
- The hosts map is not used to distribute hosts information. It's essentially empty. All name lookups are done via DNS, which is largely independent of the hosts map. The database that is used to build DNS files has been changed 502 times in the last two years, and probably 1000 times in the last four.

Again, a number of observations specific to Northeastern can be made:

- There are several orders of magnitude difference between some of these maps and others, and several of the maps have been changed far more often than any files in the configuration system. It's clear from the number of changes in the passwd and amd.home maps that NIS is being used to support those files in the environment that change the most.
- For the most part, the files can be categorized by order of magnitude of their revisions.
  - O(1000) files include amd.home, passwd and netgroup. These are the files that must be changed in order to create a new account and arrange for it to be usable. These files reflect daily changes in the system, and they are very tightly coupled with the users of the environment.
  - O(100) files include amd.net, amd.proj, and group. The amd files are changed when the group adds new disks to some computer somewhere and must make them visible to the environment. The group files are changed most often when students enter or leave groups. These correspond with the super-users and password file changes in the central configuration system. These files reflect ways in which different parts of the organization use the environment.
  - O(10) files include services and ypservers. These change when some new service or function needs to be added to the environment, or when the network structure is modified. They correspond with centrally managed files like inetd.conf, rc.priv, and daily. Changes in these files reflect modifications to the network and to machines used as servers, and indicate a new function or a major architectural change in the network.
  - O(1) files are generally either unused or setup once and then forgotten, reflecting some part of the environment that rarely changes. Ethers, for example, never changes because this site does no diskless booting. Further changes typically reflect minor fixes. This pattern is generally true with the centrally managed files as well, although in some cases, such as the shells, the small number of modifications is because the mechanism for change has been delegated to another part of the system.
  - Some files do not fit perfectly into this ranking. For example, the hosts database has been updated around 1000 times, making it an O(1000) file. The hosts database at this site is comprehensive, including information such as architecture type, IP address, user, and OS version. Due to the expansive nature of this

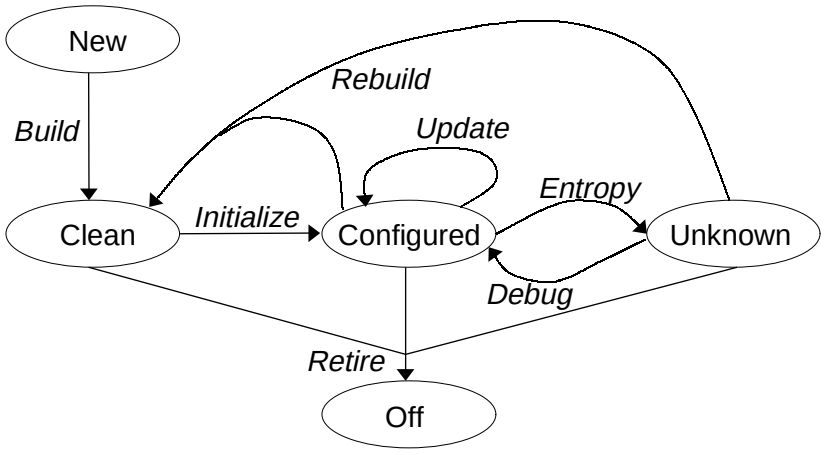

Figure 4: The lifecycle of a machine.

information, it is hard to generalize when changes are made to the hosts database, other than to say that it is changed whenever something relating to the identity of a machine is updated. More research into this area is required.

Keeping in mind that this data spans four years, the actual number of revisions of files is less important than how often they change relative to each other.

- It is possible for the password files and the amd.home files to be maintained as actual files on every host. This would require that new files be pushed out on the average of three times a day, which would pose several problems for the configuration system, including delivery to all machines (including ones that are down) and speed of distribution. In general, the NIS mechanism scales better to a high frequency of changes than their existing configuration system.
- The NIS system has achieved a certain degree of abstraction at this site. The administrators do not think of ‘‘adding a new user to the entire network by changing a file in NIS,’’ they simply think of ‘‘adding a new user to the NIS maps.’’

Whether these statistics and observations will correlate with those of other sites remains to be seen. Regardless, it is hoped that this data will provide some insight into where and how often changes are made in a real-world environment.

General Observations and Theories

The real killer for systems administrators is change. Requirements change, new versions of applications appear, hardware becomes outdated, users change their mind about what they need . . . everything’s a moving target. A configuration management system, in part, is the process by which an administrator controls and manages operating system and host changes.

From the literature survey, the site interviews, and the in-depth study of Northeastern’s configuration file changes, I have collected a lot of observations and developed a few conjectures about the patterns of changes that take place on a host and in a network of computers. It is important to understand these patterns because they can be helpful in understanding the issues and requirements in a configuration system. Furthermore, they can be of help in developing stronger models and abstractions that may eventually result in an improvement in systems administration methods.

Changes Within A Machine Life Cycle

The life cycle of an individual machine is an interesting place to investigate the role of change. This cycle in itself is a complicated process and worthy of further study, but not in this paper. A sufficiently detailed version of the cycle is given in Figure 4. In the figure, a machine moves between these states:

- New. A new machine.
- Clean. A computer with the OS installed, but not configured to work in the environment. (For example, its network is unconfigured.)
- Configured. A computer that is configured correctly according to the requirements of the computing environment.
- Unknown. A computer that has been misconfigured, or has gotten out of date, or perhaps been borrowed by an intern and returned with stains on it.
- Off. All done. Retired. Excessed. A dead parrot.

The interesting part of the figure are the changes that a machine goes through as it moves between states. These are ‘‘processes,’’ and consist of:

- Build. During the build process, the operating system is installed on the machine.
- Initialize. This occurs directly after build, and at many sites, is thought of as part of the same process. This is the initial set of modifications to the OS image that are required to have the computer operate in the environment. This will typically include network configuration, and may include OS patches and other changes. Once initialization is complete, the computer is (theoretically) a functional citizen of the computing environment.
- Update. At some point after the initialization, the computer will probably have to be modified. Perhaps the network configuration has changed, or a user needs to be added, or an OS patch needs to be applied, or the machine needs some kind of new functionality. Whatever the cause, the computer needs to be updated in order to bring the machine into conformance with the requirements. In most cases, this will happen continually for the lifetime of the computer.
- Entropy. This refers to the gradual process of change that results in a computer that has an unknown state. The causes for this are numerous; they include, for example, undisciplined changes made to the machine, major changes in the environment, or unexplained problems.
- Debug. This refers to the process of debugging an ‘‘unknown’’ machine, and getting it back into spec. This is usually an intensive, hands-on experience. Debugging can often involve updating as well.
- Rebuild. In some cases, a machine will need to be rebuilt, either because of some kind of problem or because the changes to be made are so drastric that simple updates make no sense. For example, a rebuild is typically done when upgrading from one major revision of an OS to

the next. The rebuild process usually consists simply of reapplying the build and initialization processes to the machine.

- Retire. This is the process of turning a machine off. In some sites, there is an official process for this, in others, it merely involves turning the computer off or forgetting it exists.

Obviously, these are generalizations. In some sites, and for some operating systems, the ‘‘build’’ and ‘‘initialize’’ processes are the same thing. At other sites, there is no particular definition that can be pointed at to say that a machine is ‘‘configured.’’ It may only be ‘‘working right’’ or ‘‘broken,’’ which, in effect, are the same thing as ‘‘configured’’ and ‘‘unknown.’’

In fact, this may be an important point. The literature and common knowledge implies that there is some strong definition of how a machine should be configured in an environment. Yet the majority of the sites that were interviewed could not say with certainty whether or not any of their hosts matched that definition; they could only say that they were working without complaint. It may be that we need a less rigorous concept of an environment definition.

In this life cycle, the desired state of a computer is the ‘‘configured’’ state, and almost all of the effort in the life cycle involves trying to get a machine configured and keeping it there. Changes to a machine only take place in the process portion of the diagram. So why do these processes take place, and what do they entail?

The ‘‘build’’ and ‘‘initialize’’ processes take place because a machine is in a known state (new or clean) and must be brought into conformance with the definition of a configured machine in the environment. Here the machine needs to be updated, but the requirements are stable.

The ‘‘update’’ process takes place because a new requirement has been identified, and the machine needs to reflect that requirement. In this case, the machine needs to be updated because the environment has changed.

The ‘‘entropy’’ process is perhaps the most interesting and least understood, and is the section of the graph that needs the most expansion. For our purposes, entropy includes any kind of situation where a machine is discovered to not have the correct configuration, and getting it back to where it should be will be difficult.

The ‘‘debug’’ process can be painful and time-intensive. It takes place only because entropy has occurred. (This is different from debugging a known configuration.)

Finally, the ‘‘rebuild’’ process takes place either as an alternative to debugging, or because the changes that must take place are so extensive that the ‘‘initialize’’ process is preferred. The rebuild process is a way of taking a machine in any state and moving into a known and understood state. It takes place either because of massive changes in the environment or massive changes in the machine.

In general, changes must take place on a machine for these reasons:

- The machine needs to conform to the environment.
- The environment has changed, and a machine needs to be modified to match it.

Using this model, we can identify the high-level requirements for an abstraction mechanism to manage configurations:

- It must contain or have access to the definition of the environment.
- It must be able to perform or replace each of the processes that result in a configured machine, i.e.:
  - build
  - initialize
  - update
  - debug

As noted above, all of the effort in the life cycle is involved in getting a machine into the ‘‘configured’‘ state. It may be worth considering a model in which the configuration abstraction simply takes a machine in *any* state and configures it.

Areas of Change

Modifications made to UNIX machine file space are often categorized on into the following areas:

- The operating system. The kernel, libraries, server processes, controlling files, initial applications and whatever else the vendor has decided is part of their OS release.
- Software. Any additional software installed on the machine beyond what was installed as part of the OS. In some places, new software installed only goes into /usr/local, while in others, it may go anywhere.
- User space. Home directories, files in /tmp, crontabs, and so on. On UNIX machines, the presence of these files doesn’t typically impact the configuration of the machine from an administration perspective.
- Glue. The part of the computer that makes the environment appear like one large system, including such things as hooks for file system mounts. User space and software is often accessible via these mounts rather than residing on local disk.

The distinction between the operating system and the software installed comes to us because of the traditional model in use at many large sites: each individual workstation has its own copy of the OS, while the software is delivered to them from a central server. User space is typically centrally served as well. Thus, the difference between OS and software can often be

"what is on the local machine" versus "what is on the server."

In this distinction, for the purposes of workstation change management, the centrally served software and user space repositories is usually considered to be a different kind of configuration management problem. They are not so much a part of machine configuration as environment configuration. However, software and user data is sometimes installed directly on machines in order to improve performance or reduce the dependency on the network. In these cases, as was seen in the examination of the configuration files at Northeastern, the differences between a configuration distribution scheme and a software distribution scheme can become quite unclear.

This categorization also emphasizes the issue of responsibility for change management, as shown in Figure 5. Interestingly, the systems administrator is involved in each area of change. Note that in the OS and software areas, the sysadmin must work to configure something that was created by someone else. Perhaps that's why systems administration is so hard.

The Role of the Environment Model in Host Changes

The above discussion assumed a model where a machine has its own copy of the operating system, and gets user data and software data from over the network. This is a simple model, useful for understanding changes on a local machine, but very few real world systems conform completely to it.

On one end of the spectrum is the environment where every machine has its own OS, all local software, and all local user space. This is often done because the performance of local disk is so much better than network disk, because the environment is so small that the systems administrator (if there is a designated administrator) hasn't had to discover the value of centralized servers, or simply because everyone at the site is very good at administration of their own machine.

At the other extreme is the diskless workstation or X terminal model, where absolutely no data whatsoever is kept on the individual machines, and everything is served from some set of central locations. This is usually done to make administration easier, but if it's not done right, it can still be quite difficult to manage.

The need for a configuration management system may be less pronounced in some models than in others. Ideally, an abstraction mechanism would be applicable to the entire spectrum of data distribution models. In examining existing systems, it appears that one constant goal is to achieve the reliability and performance of the independent machine model, while achieving the management simplicity and environmental consistency of the diskless model.

Change Magnitude Conjecture

Based on the observations of the orders of magnitude of changes in the Northeastern University configuration files, I propose this model for understanding change magnitude.

Assume an organization with a sufficiently large computing system, and the following sets of files:

- U – the files that contain information that relates to the way in which specific users can use the system
- G – the files that contain information that relates to the way in which a particular group of users can use the system
- E – the files that contain information that defines the services that function in the network and the architecture of the environment
- I – the files that are used to initialize services that then reference centralized information resources

In practical terms, files in set U change more often than files in set G, files in set G change more often than files in E, and files in set E change more often than files in set I. Furthermore, in my observations, the ratio of changes between U and G is approximately the same as the ratio of changes between G and E, and so on.

In psuedomathematical terms, if C(X) means "the number of times that a file of type X is changed," then there is a number k>1 such that

$$C(U) \geq kC(G) \geq k^2C(E) \geq k^3C(I).$$

This postulation has yet to be proven in any formal sense. In order to so, one would, at the very least, have to come up with a more rigorous definition of the sets of files.

However, if it turns out to be generally true then it has important ramifications to those working on configuration management systems. In particular, systems must best support distribution of files with a high change value.

	Initial Responsibility	Configuration Responsibility
OS	Vendor	Sysadmin
Software	3rd party	Sysadmin
User	Sysadmin	User
Glue	Sysadmin	Sysadmin

Figure 5: Responsibilities.

This also points out an interesting problem for systems administrators. In this model, one would expect that files in group U, which are related to specific user information and change the most often, should only impact one user if there were a problem with the change. Likewise, files in group E, which configure system-wide services, should be more likely to impact a large number of users at one time. While this situation is true in general, it is most certainly not true every time. If there is a problem in the NIS passwd file or in the central DNS entries, it is entirely possible for it to take down an entire environment.

We must design our systems so that the changes that are made the most often have the least potential for negative widespread impact.

The State of the Community

There is a disturbing dichotomy in the systems administration community. The experienced administrators with whom I've discussed the contents of this paper generally feel that the area of systems configuration is well understood and that many of the points contained here are nothing new. This may well be true, since this has been an area of exploration for at least ten years.

At the same time, these administrators and nearly every one of subjects of the site survey indicated a strong dissatisfaction with the system they were using. None of the more sophisticated tools developed by the LISA community were being used at any of the sites that I visited. In fact, each of them used a home-grown tool, often layered on top of rdist or NIS. None of the newer administrators were aware of the work that has been done by the community, and may be doomed to putting out fires until they too have developed 20/20 hindsight and specialized scripts.

Even though our environments are changing like mad, our standard methods for handling the changes have remained largely the same. It is to be hoped that a deeper understanding of the area will help to solve this problem.

Towards a Stronger Abstraction

I began this paper by suggesting that systems administration community needs stronger abstraction models in order to manage complexity. Throughout this paper, I have made observations that could be factored into the creation of such an abstraction. I would like to close by summarizing a few key points about possible abstraction models.

A good abstraction model changes the way in which one thinks. It presents an interface and hides implementation details. One should be able to think in terms of ''updating the environment'' rather than in terms of pushing changes out to hosts.

It may be necessary to change the configuration model in order for it to support strong abstraction. A few ways to do this were suggested earlier:

- Migrate changes into the network and away from the host. The aliases file at Northeastern is a good example of this model. Rather than make changes on every host's aliases file, or even make changes to the NIS aliases map and push them out, all changes are made on and isolated to the primary mail server. The passwd map under NIS is a bad example of this. Even though the passwd map is the source of the vast majority of changes, the local /etc/passwd file must still be updated regularly on client machines in order to change root passwords or update the netgroups in the file.
- The usual model is ''configure from,'' i.e., one assumes certain information about a host (for example, that it is up) and makes changes to that configuration to get to the desired state. An alternate model is ''configure to,'' where one simply describes the desired final state, and it is up to the machine to figure out how to get there. The MIT Athena project [Rosenstein] used a version of this model rather extensively, but it seems not to have caught on much outside of the Athena environment.

Furthermore, it should be possible to instrument and evaluate any methods or tools being used to implement the abstraction. Libraries are often available with debugging and profiling information to allow programmers to improve the quality of the code that calls the library routines. One can compare library routines and see which performs better, even if one doesn't know the details of the code. We need to be able to measure our tools and understand whether or not they have improved the quality of our systems administration. This may require some kind of analysis tools or formal models of the abstraction, perhaps allowing one to describe the environment and the changes applied to it in some kind of state diagram.

The ultimate goal is to improve systems administration by making it easier to manage large and complex systems. Hopefully, this study of configuration mechanisms in practice today will help the systems administration community move one step closer to that goal.

Acknowledgements

This work was supported by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Computational and Technology Research, U.S. Department of Energy, under Contract W-31-109-Eng-38.

My sincere thanks goes out David Blank-Edelman, Michele Evard, Bill Nickless, Gail Pieper, Gene Rackow, and members of the MCS Support Group for moral support and suggestions for this paper. In addition, I would like to express my gratitude to the many anonymous folks who participated in the site case study.

Author Information

Rémy Evard is the Manager of Advanced Computing Technologies in the Mathematics and Computer Science Division of Argonne National Laboratory. Among other things, this means that he looks back fondly on the days when he had time to crash the system in spectacular ways while exploring weird administration ideas. He can be reached at evard@mcs.anl.gov.

References

[Anderson] Anderson, Paul, ''Towards a High-Level Machine Configuration System,'' *LISA VIII Proceedings*, 1994.

[Fisk] Fisk, Michael, ''Automating the Administration of Heterogeneous LANs,'' *LISA X Proceedings*, 1996.

[Harlander] Harlander, Dr. Magnus, ''Central System Administration in a Heterogeneous Unix Environment: GeNUAdmin,'' *LISA VII Proceedings*, 1994.

[Imazu] Imazu Hideyo, ''OMNICONF – Making OS Upgrades and Disk Crash Recover Easier,'' *LISA VIII Proceedings*, 1994.

[JumpStart] Sun Microsystems, http://www.sun.com/smcc/solaris-migration/tools/docs/cookbook/30.htm.

[NIS] Sun Microsystems Inc., ''The Networking Information Service,'' *System and Network Administration*, 1990.

[Rosenstein] Rosenstein, Mark A., and Geer, Daniel E., and Levine, Peter J., ''The Athena Service Management System,'' *Proceedings of 1988 USENIX Conference*, 1988.

[Rouillard] Rouillard, John P. and Martin, Richard B., ''Config: A Mechanism for Installing and Tracking System Configurations,'' *LISA VIII Proceedings*, 1994.

[Shaddock] Shaddock, Michael E. and Mitchell, Michael C. and Harrison, Helen E., ''How to Upgrade 1500 Workstations on Saturday, and Still Have Time to Mow the Yard on Sunday,'' *LISA IX Proceedings*, 1995.

[Stern] Stern, Hal, ''Managing NFS and NIS,'' O'Reilly and Associates Inc., 1991.

[Stone] Stone, Ken, ''System Cloning at hp-sdd,'' *LISA I Proceedings*, 1987.

[Tivoli] Tivoli Systems, *Tivoli Management Environment*, 1992.

[Zwicky] Zwicky, Elizabeth D., ''Typecast: Beyond Cloned Hosts,'' *LISA VI Proceedings,* 1992.