



---

# Software Distribution and Repositories

Paul Anderson

Department of Computer Science  
University of Edinburgh  
[paul@dcs.ed.ac.uk](mailto:paul@dcs.ed.ac.uk)

## Introduction



- In a large site, it is useful to have a single master copy of each application package which is configured and maintained centrally
- The package can then be supported efficiently and a consistent version can be distributed to all local users
- This talk surveys a number of approaches to the problem of storing, distributing and updating such packages

- ✍ A package is any collection of software or data which is normally installed on a machine as a single unit. Configuration of a package may involve setting pathnames and other variables appropriately for the local site, application of patches, local modifications, and compilation from source.
- ✍ “Maintained centrally” means that only one “master” copy of each package is stored, rather than having multiple variants of the same package, supported by different people. This does not imply that all packages are maintained by the same person, or even stored in the same physical location.
- ✍ There is probably no one ideal solution to the problem. We will discuss some implementations which are intended to illustrate the variety of approaches and trade-offs available, rather than present the “best” practical system. Together with the references, this provides a basis for evaluating, selecting, or designing a suitable implementation for a particular site.
- ✍ Only Unix-based implementations are discussed, and commercial products (g. TME) are not considered.

## Overview

---



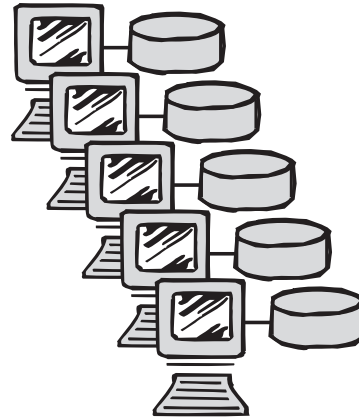
- Package provision
  - Standalone, file servers, software distribution
- Repository organisation
  - By directory, by package, special formats
- Software Distribution
  - Distribution mechanisms, Pre- and post- installation, Configuration, Security
- A Wish List
- References

- ✍ Looking at the problems with traditional ways of supplying packages to network clients provides the motivation for a repository/distribution approach. It also identifies some properties that we would like an implementation to have.
- ✍ The organisation of the repository and the distribution of the software are often treated independently - it is even possible to have a repository which is not distributed. However, the characteristics of the repository affect the choice of any distribution mechanism and visa-versa.

## Packages on Clients



- Packages are held locally on each client
- No guarantee of consistency between systems
- Time-consuming to update and install
- Per-client customisation is easy (but not per-user)

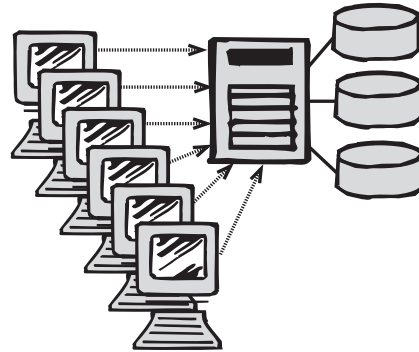


- ✍ This configuration is still common among small PC clusters. Considerable effort can be wasted by installing packages individually on each machine. This effort may be “hidden” if it is performed by the end-users themselves.
- ✍ The lack of consistency between machines is a problem in any situation where users need to use different machines: eg. a public laboratory. It also makes user-support more difficult.

## Packages on a Fileserver



- All clients mount packages from a single fileserver
- The fileserver forms a single point of failure
- This does not scale because of server performance
- The client view is the same as the server



- ✍ This is a typical configuration for a small cluster, but performance and resilience considerations mean that it is not suitable for larger networks.
- ✍ Larger networks are often formed from a number of such clusters. Different servers frequently hold different sets of packages and there is a tendency for inter-cluster links to develop which considerably reduce the robustness of the system.

## A Distributed Approach

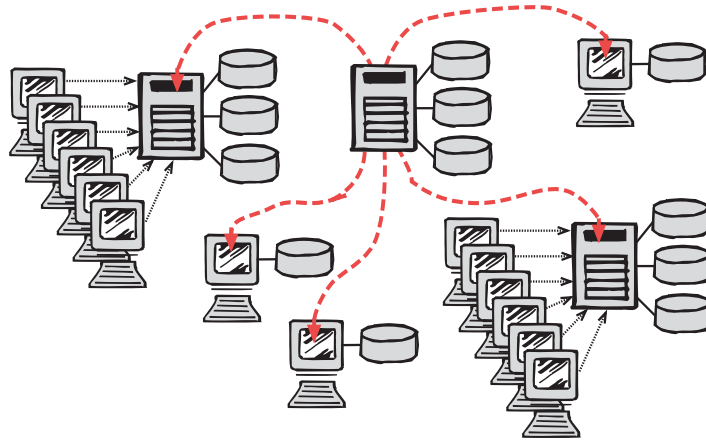
---



- A Master server contains a repository of packages which are distributed to servers and client disks
- Periodically, the packages are synchronised with the master copy in the repository
- This attempts to address the problems of scalability and resilience while retaining the administrative advantages of a single server

- ✍ The repository may hold considerably more than any one server, including binaries for multiple platforms and multiple versions of single packages. The repository may also be spread across more than one physical server, and the individual packages may be managed by different people.
- ✍ Since clients can operate independently from the network, except during updates, this approach allows portable machines to share a consistent environment with the main network.

## A Distributed Approach



SoftwareDistribution and Repositories: 7

Paul Anderson: 24/4/97

- ✓ In a large system, there may be more than one level of distribution to reduce the load on the master repository.
- ✓ Practical systems tend to involve a hybrid approach where clients mount some files from a server and synchronise local copies of other files from the repository.
- ✓ File caching is also common to reduce server load, although this does not improve resilience. This can be achieved with a caching file system such as Sun's **cacheFS**<sub>[33]</sub>, or with a program such as **lfu**<sub>[3]</sub> or **nightly**<sub>[21]</sub>.

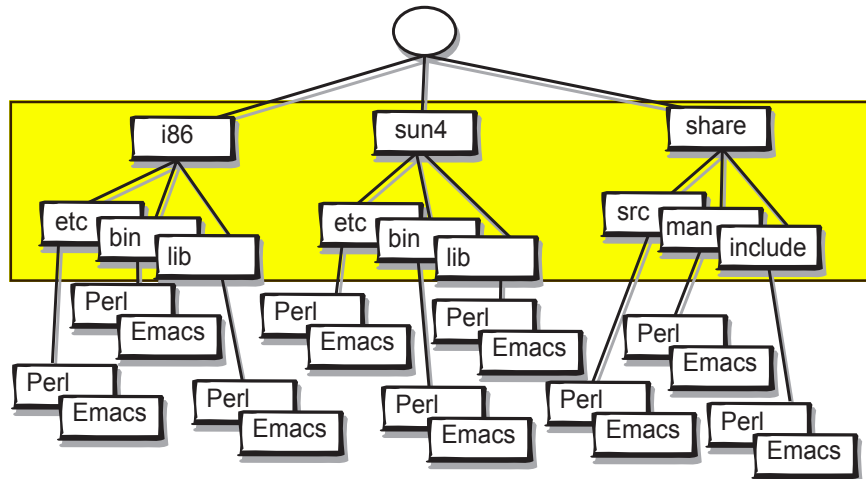
## Repository organisation



- Simple repositories are organised in a directory structure which is similar to the structure on the file server (or client)
- Repositories which are structured by package are easier to manage
- Many systems now use special formats for the packages which include additional information

- ✍ If the repository structure is similar to the directory structure on the fileserver, then distribution is simply a matter of copying appropriate sections of the directory hierarchy to the server. However, as with a simple server it can be difficult to identify which files belong to which packages, because they are spread throughout the filesystem.
- ✍ This problem is solved if the repository is organised “by package”, but distribution is then no longer a case of simply copying directories to the server - there needs to be some way of installing the files on the server so that they appear in suitable locations - eg. in the user’s path.
- ✍ Neither of the above approaches can cope well with packages that require significantly more than copying files - eg. different files may need copying for different hardware variants, and some post-installation operations may be required.

## Organisation by Directory

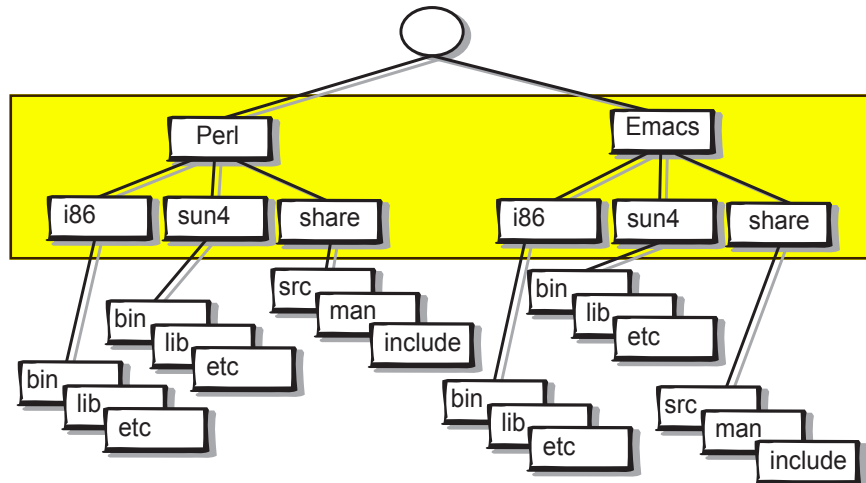


SoftwareDistribution and Repositories: 9

Paul Anderson: 24/4/97

- ✍ In this case, the structure is very similar to the structure of the fileserver itself. Since repositories usually serve more than one platform, it is normal to split the contents into architecture-specific and architecture-independent directories. The “share” directory and the directory for the appropriate architecture are simply copied to the fileserver.
- ✍ With this organisation, it is difficult to determine which files belong to which packages. One possibility is to use explicit fileset lists, although these require maintenance and can easily become out of synchronisation. `lfu`<sup>[3]</sup> uses a different user-id as the owner of files belonging to each different package - this works well except that certain applications require a specific owner for certain files - eg. `setuid root`.
- ✍ Installation of most packages into the repository is comparatively simple, since it is very similar to installing on a normal server. However, it is difficult to hold multiple versions of the same package, or other packages which share the same pathname for different files.

## Organisation by Package



SoftwareDistribution and Repositories: 10

Paul Anderson: 24/4/97

- With this organisation, files belonging to a particular package are easy to locate, and multiple versions and conflicting filenames are easily handled. It is also reasonably straightforward to install packages into the repository, because the hierarchy for an individual package appears very similar to a conventional server. The modularity of packages means that they can easily be shared between sites.
- However, the structure of the repository is now different from the directory structure of a normal client. The files could be relocated as they are copied, or they could simply be copied into the same hierarchy on the client, with some mechanism such as **Modules**<sub>[29]</sub> used to ensure that the appropriate packages appear in the user's path. Alternatively, a system of links and/or loopback mounts can make this structure appear more like a conventional client (eg. **Slink**<sub>[9,10]</sub>).
- Numerous systems have been developed around these general principles. **Depot**<sub>[17,6]</sub> is a popular example. Others include: **Lude**<sub>[11]</sub>, **ASIS**<sub>[12]</sub>, **The corporate software bank**<sub>[15]</sub> and **Ericsson**<sub>[16]</sub>

## Special Package Formats



- Additional information needs to be stored to enable more complex installations
- This information is often stored in some special format together with a compact version of the package files
- Disadvantages:
  - Formats can be difficult to create
  - Incremental updates are not normally possible
  - There are many different incompatible formats
  - Special software is needed to unpack them

- ✍ Complex installation requirements are not well satisfied by simple file-copying schemes - different files may need to be selected depending on specific features of the server, and complex pre- and post- installation operations may be needed (see later).
- ✍ Many formats have been developed to store packages together with additional installation information. Special software is usually required to create these package formats, and to extract the data and perform the installation. Such systems are typically used for installing stand-alone PC packages.. Examples in the Unix world include the **Posix** standard<sup>[34]</sup>, System V **pkgadd**<sup>[32]</sup> and Linux **RPM**<sup>[30]</sup>.
- ✍ These formats provide a compact representation of the package data and it's installation requirements, and there is often some mechanism for maintaining a database of packages installed on a particular machine. However, they suffer from the disadvantages listed above and are not normally designed to be used remotely (although one version of RPM can retrieve packages by URL).

## Distribution Mechanisms



- A distribution mechanism is required to transfer files from the repository to the servers
- Some implementations push files from the source machine. Others pull files from the destination
- Incremental updates are usually necessary so that files are only copied when they are changed
- Updates may occur periodically, or on-demand

- ✍ In the simplest case, a distribution mechanism just needs to perform a remote file copy, but the distribution mechanism often performs a number of other tasks well - eg. incremental updating, pre- and post- installation operations, and file caching. **Track**<sup>[18,19]</sup>, **Sup**<sup>[26]</sup>, **RDist**<sup>[7,8]</sup> and **Ifu**<sup>[2]</sup> are typical programs. RDist is probably the most common and has been used as the basis for several more complex schemes (eg. **FDist**<sup>[24]</sup>).
- ✍ Some systems initiate the copy by “pushing” from the source (RDist) and some initiate the copy by “pulling” from the destination (Track). The choice of method has implications for configuration and security (see later).
- ✍ Most implementations are capable of performing an incremental update by only copying files that have changed. However, if the number of files is large, simply performing the comparison can take a considerable amount of time and bandwidth, and some systems (eg. Track) maintain a state-database of file changes, to speed up the process. Systems which store packages in a special format cannot normally handle these fine-grain incremental updates.

## Distribution Protocols



- Distribution implementations may use custom protocols, or they may make use of existing file-transfer protocols.
- Some systems do not address the remote transfer issue at all and rely on remote filesystem access
- Robustness is important
- Parallel updates are useful
- Wide-area networks pose special problems

- ✍ Some implementations (Track) use their own protocols and some use existing protocols (RDist uses rsh, RPM can use FTP). Some systems are designed to use an underlying remote filesystem protocol such as NFS (Ifu). The choice of protocol has implications for robustness, performance and security.
- ✍ Failure of a software update can leave the destination machine in an unusable, or insecure state, so the behaviour of the protocol in response to failures is important. For example, some programs can check the available disk space in advance (RDist), and Ifu can automatically link files to a remote server if there is no space to copy them locally.
- ✍ For systems which operate in the “push” mode, it is useful to be able to update remote systems simultaneously. Some versions of RDist are capable of this. However, the load on a single repository server can easily become too great. A multi-cast protocol for file comparison and distribution would be very useful.
- ✍ Both the comparison and the transfer of files may need special attention (eg. compression) on a wide-area network<sup>[20]</sup>.

# Pre- and Post- Installation



- Pre-installation
  - Hardware compatibility
  - Pre-requisite package availability
  - Resource availability
- Post-installation
  - customisation
  - System file changes
  - Driver installation

- ✍ In many cases, it is not sufficient just to transfer the package files from the repository to the server - some additional operations are needed:
- ✍ Pre-installation operations are required to ensure that any pre-requisites are met - eg. processor type, disk space, memory, dependencies on other packages etc.
- ✍ Post-installation operations include any further actions necessary to make the package available and configure it for the particular host - eg. configuration files might need editing to include information about the host - system files such as `inetd.conf` might need extra entries - special device drives or system configuration parameters might need setting.
- ✍ Most distribution mechanisms provide some facility for executing arbitrary scripts before and after updating a package. Systems that use special package formats usually include specific provision for these operations (Posix, RPM). However, constructing good, portable scripts for these operations can be very difficult<sup>[31]</sup>.

# Configuration



- Configuration information is needed to specify which machines carry which packages
- Changes in the configuration can initiate package installs and removals
- Configuration files which refer to individual files and directories can become very complex
- Configuration can be centralised or distributed
- Distributed configurations may refer to central databases

- ✍ In a large installation, not all servers carry the same packages. In the simplest case, packages have to be explicitly added or removed. However, it is usually desirable to have a declarative configuration file which lists the packages that a server should carry. A periodic update program then adds or removes packages automatically to track changes in the configuration file.
- ✍ Configuration file formats are often oriented towards specification of individual directories and files - eg. RDist uses a "distfile" which is similar to a makefile. In complex networks, these configurations become too large and detailed. Some mechanism is needed to simply specify the required package names.
- ✍ In a "pushing" implementation, configuration files are normally stored and controlled from the source (RDist). In a "pulling" implementation (Ifu), each server usually has its own configuration file.
- ✍ Even if the configuration files are distributed, it is useful if they can refer to some central configuration database - eg. Ifu configuration files can use NIS netgroups to specify sets of packages. Changes to one of these netgroups will cause all affected servers to add and remove the appropriate packages.

# Security



- Repository security
  - Different packages may be maintained by different people, who may not have root access
- Distribution security
  - Security during distribution depends on the underlying protocol
- Package verification
  - Trojan horses can easily be installed unless there is some way of verifying installed packages

- ✍ Packages in the repository can be owned by a specific user-id, allowing one person (or group) to maintain the package. In this case, the distribution mechanism needs to be able to change user-ids during installation for specific files - eg. setuid root files. Implementations which store packages separately and require users to add them explicitly to their path are more secure than systems which place all available packages in a common "bin" directory.
- ✍ Security during distribution depends on the underlying protocol. Many implementations are weak in this area - eg. using NFS or rsh/rhosts authentication. Systems which "pull" only need read permission on the repository, whereas "pushing" requires write permission on all the servers.
- ✍ Some mechanism for verifying the package installation is useful - eg. taking a checksum or signature of the package files (RPM). This helps to prevent "trojan horses".
- ✍ Pre- and Post-installation scripts normally need to run as "root".

## A Wish List

---



- Standard, portable package formats
  - Simple to create
- Distribution
  - Incremental, parallel, secure
- Secure package management by non-root users
- High level configuration
- Lightweight and free!

- ✍ Some of the implementations mentioned include some of the above features. We are not aware of any implementation which addresses all of these issues.
- ✍ There is a tendency towards using special package formats, but considerable effort can be required to assemble a package into a suitable form. Common packages could easily be shared between sites if a standard format was accepted. It should also be easy to install packages that have only very simple requirements.
- ✍ Remote distribution and features such as incremental update are not an integral feature of any of the implementations which use special formats.
- ✍ Secure package management by non-root users is very difficult. handling pre- and post-installation requirements is particularly difficult.
- ✍ High level configuration should allow a system manager to specify sets of packages to be held on groups of machines.

## References (1)



- [1] Paul Anderson. **Ifu manual page**. Department of Computer Science, University of Edinburgh, 1991.
  
- [2] Paul Anderson. **Managing program binaries in a heterogeneous Unix network**. In Proceedings of the 5th Large Installations Systems Administration (LISA) Conference, pages 1-9, Berkeley, CA, 1991. Usenix.
  
- [3] Paul Anderson. **Effective use of local workstation disks in an NFS network**. In Proceedings of the 6th Large Installations Systems Administration (LISA) Conference, pages 1-7, Berkeley, CA, 1992. Usenix.
  
- [4] John D Bell. **A simple caching file system for application serving**. In Proceedings of the 10th Large Installations Systems Administration (LISA) Conference, pages 171-175, Berkeley, CA, 1996. Usenix.
  
- [5] David J Bianco, Travis L Priest, and David E Cordner. **Cicero: A package installation system for an integrated computing environment**. Technical report, NASA, 1994.
  
- [6] Wallace Colyer and Walter Yong. **Depot: A tool for managing software environments**. In Proceedings of the 6th Large Installations Systems Administration (LISA) Conference, pages 151-158, Berkeley, CA, 1992. Usenix.
  
- [7] UC Berkeley Computer Systems Research Group. 4.BSD System Manager's Manual, chapter **rdist**. O'Reilly & Associates, July 1994.
  
- [8] Michael A Cooper. **Overhauling rdist for the '90s**. In Proceedings of the 6th Large Installations Systems Administration (LISA) Conference, pages 175-188, Berkeley, CA, 1992. Usenix.

## References (2)



- [9] Alva Couch and Greg Owen. **Managing large software repositories with slink**. In Proceedings of the SANS V Conference, Berkeley, CA, 1995. Usenix.
- [10] Alva L Couch. **Slink: Simple, effective filesystem maintenance abstraction for community-based administration**. In Proceedings of the 10th Large Installations Systems Administration (LISA) Conference, pages 205-212, Berkeley, CA, 1996. Usenix.
- [11] Michel Dagenais, Stéphane Boucher, Robert Gérin-Lajoie, Pierre Laplante, and Pierre Mailhot. **Lude: A distributed software library**. In Proceedings of the 7th Large Installations Systems Administration (LISA) Conference, pages 25-32, Berkeley, CA, 1993. Usenix.
- [12] P Defert, E Fernandez, M Goosens, O Le Moigne, A Peyrat, and I Reguero. **Managing and distributing application software**. In Proceedings of the 10th Large Installations Systems Administration (LISA) Conference, pages 213-226, Berkeley, CA, 1996. Usenix.
- [13] Thomas Eirich. **Beam: A tool for flexible software update**. In Proceedings of the 8th Large Installations Systems Administration (LISA) Conference, pages 75-82, Berkeley, CA, 1994. Usenix.
- [14] Ola Lapidó. **A subscription-oriented software package update distribution system (spuds)**. In Proceedings of the 2nd Large Installations Systems Administration (LISA) Conference, pages 75-77, Berkeley, CA, 1988. Usenix.
- [15] Steven W Lodin. **The corporate software bank**. In Proceedings of the 7th Large Installations Systems Administration (LISA) Conference, pages 33-42, Berkeley, CA, 1993. Usenix.

## References (3)



- [16] Thomas Lundgren. **A file server directory tree hierarchy.** Technical report, Ericsson Telecom, 1992. ETX/TX/DD-91.
- [17] Kenneth Manheimer, Barry Warsaw, Stephen Clark, and Walter Rowe. **The depot: A framework for sharing software installation across organisational and Unix platform boundaries.** In Proceedings of the 4th Large Installations Systems Administration (LISA) Conference, pages 37-46, Berkeley, CA, 1990. Usenix.
- [18] Daniel Nachbar. **Track** manual pages.
- [19] Daniel Nachbar. **When network filesystems aren't enough: Automatic software distribution revisited.** In Proceedings of the Summer USENIX Conference, Atlanta, GA., Berkeley, CA, 1986. Usenix.
- [20] Peter Osel and Wilfried Gansheimer. **Incremental software distribution.** In Proceedings of the 9th Large Installations Systems Administration (LISA) Conference, pages 181-193, Berkeley, CA, 1995. Usenix.
- [21] Hal Pomeranz. **A new network for the cost of one scsi cable: A simple caching strategy for third party applications.** In Proceedings of the SANS III Conference, Berkeley, CA 1994. Usenix.
- [22] John Rouillard and Richard Martin. **Depot-lite: A mechanism for managing software.** In Proceedings of the 8th Large Installations Systems Administration (LISA) Conference, pages 83-91, Berkeley, CA, 1994. Usenix.

## References (4)



- [23] Bjorn Satdeva. **Methods for maintaining one source tree in a heterogeneous environment.** In Proceedings of the 7th Large Installations Systems Administration (LISA) Conference, pages 57-66, Berkeley, CA, 1993. Usenix.
- [24] Bjorn Satdeva and Paul M Moriarty. **Fdist: A domain based file distribution system for a heterogeneous environment.** In Proceedings of the 5th Large Installations Systems Administration (LISA) Conference, pages 109-126, Berkeley, CA, 1991. Usenix.
- [25] John Sellens. **Software maintenance in a campus environment: The xhier approach.** In Proceedings of the 5th Large Installations Systems Administration (LISA) Conference, pages 21-28, Berkeley, CA, 1991. Usenix.
- [26] Steven Shafer and Mary Thompson. **The SUP Software Upgrade Protocol.** Carnegie Mellon University, School of Computer Science, September 1989.
- [27] Ram R Vangala and Michael J Cripps. **Software distribution and management in a networked environment.** In Proceedings of the 6th Large Installations Systems Administration (LISA) Conference, pages 163-170, Berkeley, CA, 1992. Usenix.
- [28] Walter Wong. **Local disk Depot: Customizing the software environment.** In Proceedings of the 7th Large Installations Systems Administration (LISA) Conference, pages 51-56, Berkeley, CA, 1993. Usenix.
- [29] John L Furlani. **Modules: Providing a flexible user environment.** In Proceedings of the 5th Large Installations Systems Administration (LISA) Conference, pages 141-149, Berkeley, CA, 1991. Usenix.

## References (5)

---



- [30] Marc Ewing and Erik Troan. **The RPM packaging system.** Red Hat Software.
- [31] Paul Anderson. **Software installation on large systems.** In :login: the USENIX association newsletter, March 1993.
- [32]. **pkgadd.** SunOS reference manual, section 1M, systems administration commands, March 1995. SunSoft.
- [33] **Setting up and maintaining the cache file system.** In Solaris system administration guide, volume 1, pages 577-612 SunSoft
- [34] Barrie Archer. **Towards a Posix standard for software administration.** In Proceedings of the 7th Large Installations Systems Administration (LISA) Conference, pages 67-79, Berkeley, CA, 1993. Usenix.