File System Organization

The Art of Automounting

Version 1.4

Martien F. van Steenbergen
Sun Microsystems Nederland B.V.

Last modified on August 1, 1991 16:00

The OPEN LOOK and the Sun Graphical User Interfaces were developed by Sun Microsystems, Inc. for its users and licensees. Sun acknowledges the pioneering efforts of Xerox in researching and developing the concept of visual or graphical user interfaces for the computer industry. Sun holds a non-exclusive license from Xerox to the Xerox Graphical User Interface, which license also covers Sun's licensees.

TRADEMARKS

The Sun logo, Sun Microsystems, Sun Workstation, NeWS, and SunLink are registered trademarks of Sun Microsystems, Inc. in the United States and other countries.

Sun, Sun-2, Sun-3, Sun-4, Sun386*i*, SunCD, SunInstall, SunOS, SunView, NFS, and OpenWindows are trademarks of Sun Microsystems, Inc.

UNIX and OPEN LOOK are registered trademarks of AT&T.

PostScript is a registered trademark of Adobe Systems Incorporated. Adobe also owns copyrights related to the PostScript language and the PostScript interpreter. The trademark PostScript is used herein only to refer to material supplied by Adobe or to programs written in the PostScript language as defined by Adobe.

The X Window System is a product of the Massachusetts Institute of Technology.

SPARC is a registered trademark of SPARC International, Inc. Products bearing the SPARC trademark are based on an architecture developed by Sun Microsystems, Inc. SPARCstation is a trademark of SPARC International, Inc., licensed exclusively to Sun Microsystems, Inc.

All other products or services mentioned in this document are identified by the trademarks, service marks, or product names as designated by the companies who market those products. Inquiries concerning such trademarks should be made directly to those companies.

LIABILITY

The ideas and opinions set out by the author in this document do not necessarily reflect or represent the ideas and opinions of Sun Microsystems Nederland B.V. or Sun Microsystems Inc.

Neither the author nor Sun Microsystems Nederland B.V. accepts liability for any possible damage arising out, or in connection with the usage and application of the concepts specified in this document.

COLOFON

This document was made using FrameMaker 2.1 on a SPARCstation 2 GX in the OpenWindows 2.0 environment. Artwork was done by Nathalie Obst, scanned and imported. Other illustrations are done with FrameMaker. Body text is set in 10 point Palatino. Proof printing was done on a 400 dpi SPARCprinter and the NeWSprint software.

The document layout is a mixture of that used in the Sun Desktop SPARC Documentation and the Xerox Publishing Standards.

Contents

Preface

Much of the work described in this publication finds its roots in the system administration concepts used for the Sun386*i*. One of the documents related to the concepts that form the basis of the Sun386*i* system administration is the Sun386*i* Cookbook.

Reading the cookbook learned that some nice concepts were wrought out. However, my goal was to apply these concepts beyond the Sun386*i* environment, better still, in a heterogeneous environment consisting of Sun machines with different architectures, and even non-Suns. The objective lead to simplification of the concepts that form the basis of the Sun386*i* file system organization.

The conventions and guidelines laid out in this publication have been implemented with success in a number of environments and have proven to be very helpful, alleviating the confused view that users and system administrators often have of their environment.

I hope this document will be useful to others and I would welcome any comments and suggestions that could improve the quality of this document.

You can reach me via ordinary mail at

Martien F. van Steenbergen
Sun Microsystems Nederland B.V.
PO Box 1270
3800 BG Amersfoort
The Netherlands

or via electronic mail at

Martien.van.Steenbergen@Holland.Sun.COM[1]

or via telephone at

+31 33 501234

Acknowledgments

I would like to thank Hein Konert and Hans Bouw from Philips Consumer Electronics, Eindhoven, Holland. At Philips, the initial draft versions of this document and a publication with an even wider scope than just file system organization were crafted during a consulting job in 1990.

I would also like to thank all my colleagues at Sun for their patience and effort they've shown reviewing this document and for having me say the things I had to say. Thanks to Dirk van Ginneken and Maarten

1 A more efficient way to reach me if you are in the Dutch region is via Martien.van.Steenbergen@sun.nl.

Westenberg. I also would like to thank my manager, Mathieu Lebens, for allowing me to spend the time and the effort to write this document.

And last but not least, special thanks to Nathalie Obst for creating the artwork and reviewing the text on such a short notice.

Introduction

Now, where was it where I could find that file?!

Wouldn't it be nice if, when working with computers on a day to day basis, we had a general, uniform and consistent computing environment, especially with respect to the file system layout. Furthermore, we would also like this environment to be scalable from stand-alone systems to large heterogeneous environments.

Besides that, one would also like to be able to use the available disk space in the network as efficient as possible, avoiding fragmentation of this valuable resource. It is very frustrating to discover that you cannot save your file because a file system is filled up, while on other places in the network more than 1 GB appears to be available!

Also, being able to find what you are looking for in the file system on an intuitive fashion would reduce confusion in your user community. Separating the various classes of data available on the network and providing this data in a logical and consistent way reduces confusion and requires less help from the system administration department. A complete separation between private data (stored in home directories) and more project related data would reduce the need to snoop around in someone else's home directory. After all, most of us also don't like others to peek and poke in our private drawers, let alone suitcases.

The frustration of having to change your command search path every time a new (version of a) software application is installed, is causing headaches sometimes. Or, even worse, the search path depends on the underlying hardware architecture, e.g. Sun-3, Sun-4 and Sun386*i*.

All this and more is covered in this publication. Just by setting up some conventions and guidelines about where to put what and why is 90% of the job. And please note that setting up the system described here does *not* require any additional software to be bought or installed. You can just use the tools and technologies available at your fingertips right now.

Three main objectives that have been the most important during the definition of the guidelines are the same as those used for the design of the OPEN LOOK GUI: *simplicity*, *consistency* and *efficiency*.

Simplicity Conventions and guidelines must be simple to understand and implement. Applying those guidelines must also be very simple. So simple, that even non-administrators have a clear picture of what is going on. So simple, that even non-administrators can execute the steps required to perform a certain operation with success.

Consistency Having a consistent environment for both end-users and system administrators will alleviate their work. If the same concepts apply in all circumstances, this will greatly enhance the overall quality and understanding of the environment, and concepts once learned can be applied everywhere.

Efficiency Besides being simple and consistent, the system must also be efficient to use and apply. The overall performance and usability of the system is not allowed to suffer too much from the first two objectives. This means that in some cases exceptions have to be applied in order to ensure better performance.

Initially, the first two goals mentioned above are the most important and the document will focus on those. However, research has shown that if these rules are strictly applied, performance loss may result. In general you could say that in order to improve performance, you should try to minimize the number of symbolic links that you use.

For more information on performance issues, please refer to the article "NFS Client Server Performance" by Jos van der Meer, Frans Wessels and Maarten Westenberg from Sun Microsystems Nederland B.V.

Who should read this document

This document is written for both novice and experienced system administrators. It is assumed that you are familiar with UNIX and SunOS in general, the Network File System (NFS), the Network Information Services (NIS), tools like the Source Code Control System (SCCS) and Revision Control System (RCS).

These tools and technologies will not be explained in this document. For more information about SunOS, NFS and NIS, please refer to *System & Network Administration*. For more information about SCCS, refer to the corresponding manual pages (*sccs*(1)). RCS is free software. So if you need more information on RCS, you must first get a copy of this software. Ask a local friend or colleague if he or she knows were to get it.

How to use this document

This manual is divided into the following parts: *Choosing a name for your computer*, *System management as a real project* and *The art of automounting*.

***Choosing a name for your computer* helps you picking the right names**

Choosing the right names for the things that you work with in your environment is the first step in organizing your environment and it will make it clear to work with. This chapter details on the do's and don'ts when choosing names.

Treat *System management as a real project*

Instead of directly modifying administrative files on an ad hoc basis, you could also treat system and network administration as a real project, including its own separate project directory. A primer on how to set it up is provided in this chapter.

***The art of automounting* explains the where, what, why and how**

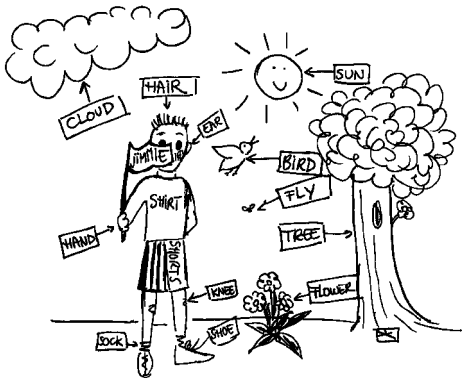
The key to success in file system organization is storing things on the right places and making them available on the right places, which are two different things. This chapter tells you all about that, including the underlying motivation for doing so. This chapter details about the real work that has to be done, up to the general commands required to implement it.

Besides the general concepts and guidelines explained in this chapter, it will go into some specific requirements that apply when installing or building applications that nicely fit in the frame work. It details about resolving architecture dependencies, different versions and variants of the same application etc. As an example, the building and installation of GNU Emacs is completely described.

1 Choosing a name for your computer

This chapter discusses guidelines regarding what to do and what not to do when naming “things” or objects in general. These objects range from computers to printers and perhaps even disks.

First of all, the article of Don Libes is included in this chapter. It has a somewhat narrow focus on computer names only, but his article has a wider scope than just computer names and can be applied to more things than just computers.



Don Libes’ memo

This is an almost exact copy of Don Libes’ RFC 1178 from the “Integrated Systems Group/NIST”, August 1990. It has only be adapted to the layout and style of this document and a few headers have had minor changes applied to them.

Status of this memo

This FYI RFC is a republication of a Communications of the ACM article on guidelines on what to do and what not to do when naming your computer [1]. This memo provides information for the Internet community. It does not specify any standard.

Distribution of this memo is unlimited.

Abstract

In order to easily distinguish between multiple computers, we give them names. Experience has taught us that it is as easy to choose bad names as it is to choose good ones. This essay presents guidelines for deciding what makes a name good or bad.

Keywords: domain name system, naming conventions, computer administration, computer network management

Choosing a name

As soon as you deal with more than one computer, you need to distinguish between them. For example, to tell your system administrator that your computer is busted, you might say, "Hey Ken. Goon is down!"

Computers also have to be able to distinguish between themselves. Thus, when sending mail to a colleague at another computer, you might use the command "mail libes@goon".

In both cases, "goon" refers to a particular computer. How the name is actually dereferenced by a human or computer need not concern us here. This essay is only concerned with choosing a "good" name. (It is assumed that the reader has a basic understanding of the domain name system as described by [2].)

By picking a "good" name for your computer, you can avoid a number of problems that people stumble over again and again.

What not to do

Here are some guidelines on what *not* to do.

Don't overload other terms already in common use

Using a word that has strong semantic implications in the current context will cause confusion. This is especially true in conversation where punctuation is not obvious and grammar is often incorrect.

For example, a distributed database had been built on top of several computers. Each one had a different name. One machine was named "up", as it was the only one that accepted updates. Conversations would sound like this: "Is up down?" and "Boot the machine up." followed by "Which machine?"

While it didn't take long to catch on and get used to this zaniness, it was annoying when occasionally your mind would stumble, and you would have to stop and think about each word in a sentence. It is as if, all of a sudden, English has become a foreign language[1].

Don't choose a name after a project unique to that machine

A manufacturing project had named a machine "shop" since it was going to be used to control a number of machines on a shop floor. A while later, a new machine was acquired to help with some of the processing. Needless to say, it couldn't be called "shop" as well. Indeed, both machines ended up performing more specific tasks, allowing more precision in naming. A year later, five new machines were installed and the original one was moved to an unrelated project. It is simply impossible to choose generic names that remain appropriate for very long.

Of course, they could have called the second one "shop2" and so on. But then one is really only distinguishing machines by their number. You might as well just call them "1", "2", and "3". The only time this kind of naming scheme is appropriate is when you have a lot of machines and there are no reasons for any human to distinguish between them. For example, a master computer might be controlling an array of one hundred computers. In this case, it makes sense to refer to them with the array indices.

While computers aren't quite analogous to people, their names are. Nobody expects to learn much about a person by their name. Just because a person is named "Don" doesn't mean he is the ruler of the world (despite what the "Choosing a Name for your Baby" books say). In reality, names are just arbitrary tags. You cannot tell what a person does for a living, what their hobbies are, and so on.

1 As a matter of fact, it is for the author...

Don't use your own name

Even if a computer is sitting on your desktop, it is a mistake to name it after yourself. This is another case of overloading, in which statements become ambiguous. Does "give the disk drive to don" refer to a person or computer?

Even using your initials (or some other moniker) is unsatisfactory. What happens if I get a different machine after a year? Someone else gets stuck with "don" and I end up living with "jim". The machines can be renamed, but that is excess work and besides, a program that used a special peripheral or database on "don" would start failing when it wasn't found on the "new don".

It is especially tempting to name your first computer after yourself, but think about it. Do you name any of your other possessions after yourself? No. Your dog has its own name, as do your children. If you are one of those who feel so inclined to name your car and other objects, you certainly don't reuse your own name. Otherwise you would have a great deal of trouble distinguishing between them in speech.

For the same reason, it follows that naming your computer the same thing as your car or another possession is a mistake.

Don't use long names

This is hard to quantify, but experience has shown that names longer than eight characters simply annoy people.

Most systems will allow prespecified abbreviations, but why not choose a name that you don't have to abbreviate to begin with? This removes any chance of confusion.

Avoid alternate spellings

Once we called a machine "czek". In discussion, people continually thought we were talking about a machine called "check". Indeed, "czek" isn't even a word (although "Czech" is).

Purposely incorrect (but cute) spellings also tend to annoy a large subset of people. Also, people who have learned English as a second language often question their own knowledge upon seeing a word that they know but spelled differently. ("I guess I've always been spelling "funxion" incorrectly. How embarrassing!")

By now you may be saying to yourself, "This is all very silly...people who have to know how to spell a name will learn it and that's that." While it is true that some people will learn the spelling, it will eventually cause problems somewhere.

For example, one day a machine named "pythagoris" (sic) went awry and began sending a tremendous number of messages to the site administrator's computer. The administrator, who wasn't a very good speller to begin with, had never seen this machine before (someone else had set it up and named it), but he had to deal with it since it was clogging up the network as well as bogging down his own machine which was logging all the errors. Needless to say, he had to look it up every time he needed to spell "pythagoris". (He suspected there was an abbreviation, but he would have had to log into yet another computer (the local nameserver) to find out and the network was too jammed to waste time doing that.)

Avoid domain names

For technical reasons, domain names should be avoided. In particular, name resolution of non-absolute hostnames is problematic. Resolvers will check names against domains before checking them against hostnames. But we have seen instances of mailers that refuse to treat single token names as domains. For example, assume that you mail to *libes@rutgers* from *yale.edu*. Depending upon the implementation, the mail may go to *rutgers.edu* or *rutgers.yale.edu* (assuming both exist).

Avoid domain-like names

Domain names are either organizational (e.g., *cia.gov*) or geographical (e.g., *dallas.tx.us*). Using anything like these tends to imply some connection. For example, the name "tahiti" sounds like it means you are located there. This is confusing if it is really somewhere else (e.g., "tahiti.cia.gov is located in Langley, Virginia? I thought it was the CIA's

Tahiti office!"). If it really is located there, the name implies that it is the only computer there. If this isn't wrong now, it inevitably will be.

There are some organizational and geographical names that work fine. These are exactly the ones that do not function well as domain names. For example, amorphous names such as rivers, mythological places and other impossibilities are very suitable. ("earth" is not yet a domain name.)

Don't use antagonistic or otherwise embarrassing names

Words like "moron" or "twit" are good names if no one else is going to see them. But if you ever give someone a demo on your machine, you may find that they are distracted by seeing a nasty word on your screen. (Maybe their spouse called them that this morning.) Why bother taking the chance that they will be turned off by something completely irrelevant to your demo.

Don't use digits at the beginning of the name

Many programs accept a numerical internet address as well as a name. Unfortunately, some programs do not correctly distinguish between the two and may be fooled, for example, by a string beginning with a decimal digit.

Names consisting entirely of hexadecimal digits, such as "beef", are also problematic, since they can be interpreted entirely as hexadecimal numbers as well as alphabetic strings.

Don't use non-alphanumeric characters in a name

Your own computer may handle punctuation or control characters in a name, but most others do not. If you ever expect to connect your computer to a heterogeneous network, you can count on a variety of interpretations of non-alphanumeric characters in names. Network conventions on this are surprisingly nonstandard.

Don't expect case to be preserved

Upper and lowercase characters look the same to a great deal of internet software, often under the assumption that it is doing you a favor. It may seem appropriate to capitalize a name the same way you might do it in English, but convention dictates that computer names appear all lowercase. (And it saves holding down the shift key.)

Names that work well

Now that we've heard what not to do, here are some suggestions on names that work well.

Use words/names that are rarely used

While a word like "typical" or "up" (see above) isn't computer jargon, it is just too likely to arise in discussion and throw off one's concentration while determining the correct referent. Instead, use words like "lurch" or "squire" which are unlikely to cause any confusion.

You might feel it is safe to use the name "jose" just because no one is named that in your group, but you will have a problem if you should happen to hire Jose. A name like "sphinx" will be less likely to conflict with new hires.

Use theme names

Naming groups of machines in a common way is very popular, and enhances communality while displaying depth of knowledge as well as imagination. A simple example is to use colors, such as "red" and "blue". Personality can be injected by choices such as "aqua" and "crimson".

Certain sets are finite, such as the seven dwarfs. When you order your first seven computers, keep in mind that you will probably get more next year. Colors will never run out.

Some more suggestions are: mythical places (e.g., *Midgard*, *Styx*, *Paradise*), mythical people (e.g., *Procne*, *Tereus*, *Zeus*), killers (e.g., *Cain*, *Burr*, *Boleyn*), babies (e.g., *colt*, *puppy*, *tadpole*, *elver*), collectives (e.g., *passel*, *plague*, *bevy*, *covey*), elements (e.g., *helium*, *argon*, *zinc*), flowers (e.g., *tulip*, *peony*, *lilac*, *arbutus*). Get the idea?

Use real words

Random strings are inappropriate for the same reason that they are so useful for passwords. They are hard to remember. Use real words.

Don't worry about reusing someone else's hostname

Extremely well-known hostnames such as "sri-nic" and "uunet" should be avoided since they are understood in conversation as absolute addresses even without a domain. In all other cases, the local domain is assumed to qualify single-part hostnames. This is similar to the way phone numbers are qualified by an area code when dialed from another area.

In other words, if you have chosen a reasonable name, you do not have to worry that it has already been used in another domain. The number of hosts in a bottom-level domain is small, so it shouldn't be hard to pick a name unique only to that domain.

There is always room for an exception

I don't think any explanation is needed here. However, let me add that if you later decide to change a name (to something sensible like you should have chosen in the first place), you are going to be amazed at the amount of pain awaiting you. No matter how easy the manuals suggest it is to change a name, you will find that lots of obscure software has rapidly accumulated which refers to that computer using that now-ugly name. It all has to be found and changed. People mailing to you from other sites have to be told. And you will have to remember that names on old backup media labels correspond to different names.

I could go on but it would be easier just to forget this guideline exists.

Conclusion

Most people don't have the opportunity to name more than one or two computers, while site administrators name large numbers of them. By choosing a name wisely, both user and administrator will have an easier time of remembering, discussing and typing the names of their computers.

I have tried to formalize useful guidelines for naming computers, along with plenty of examples to make my points obvious. Having been both a user and site administrator, many of these anecdotes come from real experiences which I have no desire to relive. Hopefully, you will avoid all of the pitfalls I have discussed by choosing your computer's name wisely.

Credits

Thanks to the following people for suggesting some of these guidelines and participating in numerous discussions on computer naming: Ed Barkmeyer, Peter Brown, Chuck Hedrick, Ken Manheimer, and Scott Paisley.

This essay first appeared in the Communications of the ACM, November, 1989, along with a Gary Larson cartoon reprinted with permission of United Press Syndicate. The text is not subject to copyright, since it is work of the National Institute of Standards and Technology. However, the author, CACM, and NIST request that this credit appear with the article whenever it is reprinted.

References

[1] Libes, D., "Choosing a Name for Your Computer", Communications of the ACM, Vol. 32, No. 11, Pg. 1289, November 1989.

[2] Mockapetris, P., "Domain Names - Concepts and Facilities", RFC 1034, USC/Information Sciences Institute, November 1987.

Security considerations

Security issues are not discussed in this memo.

Author's address

Don Libes Integrated Systems Group National Institute of Standards and Technology Gaithersburg, MD 20899

Phone: (301) 975-3535
EMail: libes@cme.nist.gov

A sample configuration

During the discussion in this document, the sample environment tries to use the same kind of names that we use in our day to day life. For example, people are used to districts or neighborhoods that have the names of flowers, planets and mills, say. The same can be applied to objects in a networked computer environment. Picking the right name classes for the right object collections can further improve the acceptance and clarity of both the individual or local environment and the overall network with all its resources like printers, networks, domains, computers, terminal servers, gateways et cetera. You should pick name classes which somehow relate to the objects that you are naming.

For instance, since more and more powerful PostScript printers appear in computerized environments and since PostScript printers are capable of producing "art work", you could pick the name class of famous painters for your printers. Printer names like *van Gogh*, *Rembrandt*, *Dali* and *Vermeer* are good examples. For less powerful printers like simple matrix or inkjet printers which normally only produce text, you could choose the name class of famous writers or poets. E.g. *Robbins* and *Ludlum*. Network name class: spider names. NIS domain names: same as department, i.e. *user@dept.company.topleveldomain* e.g. **`Martien.van.Steenbergen@Holland.Sun.COM`**.

A small table of the entities that administrators typically manage is included to get you started.

Entity	Name class	Example
computer	composers	beethoven, chopin, mozart, liszt, bach
printer	painters	rembrandt, dali, vangogh, vermeer, chagal
partition	elements	krypton, xenon, helium, oxygen
user	self	preferable the user's first name
network	spiders	tarantula, birdspider
group		department and project names
NIS domain		*department.company.topleveldomain*
application		application base name, e.g. **frame** for FrameMaker, **emacs** for GNU Emacs

2 System management as a real project

Instead of changing administrative files like **/etc/hosts** directly and on an ad hoc basis, you could consider creating a real project environment for system and network administration. This will help you organize and structure your job and you will be able to do more in less time and improve the overall quality of your work.

This chapter explains the system administration project directory and the conventions that apply to it. It covers the directory structure and organization and the use of revision control applied to the various files.

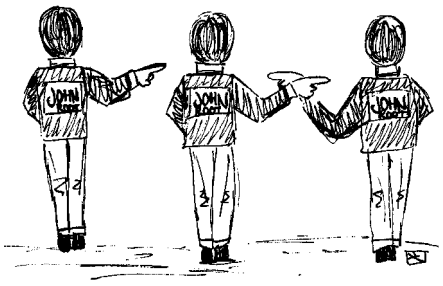
You should consider this chapter as a primer for setting up your project environment. It is by no means meant as a complete solution.

The system administration project directory

System management and administration is considered to be a real project. As with all projects, it has its own directory. In this directory, all data related to system management and administration is stored. It ranges from original copies of **hosts**, **group** and **aliases** to documentation, shell scripts, backup logs and other statistical data that needs to be maintained.

The project directory is separated from the normal location of the administrative files: **/etc**. Having a separate directory that holds all administrative information has major benefits:

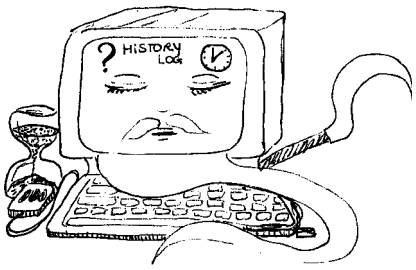
1. *Complete separation of local data and data supplied by other parties.* This improves maintainability and makes future installations of new versions of other party software very easy: you don't have to find out which files you must save before performing an upgrade; you can more or less install the new version without thought; after the upgrade, you only have to copy the relevant files to the correct destinations. You could consider to apply this separation not just for system administration, but also to application management.
2. *Original files are not stored on a local root file system, but in network wide accessible project directory.* This means that you will have the data available anywhere and almost anytime without having to login to the machine were the data is actually stored.
3. *Original files are maintained by real users instead of the anonymous root.* Besides that, it reduces the need to have super user privileges during the actual update. You will only require super user privileges during the installation and effectuation of the modified files. This can improve security. Furthermore, if revision control is applied (see below), a complete history of events, including an audit trail, is maintained at almost no extra cost. This improves consistency and reliability.



Who dunnit?!

It is up to you how you structure your directory. You could consider storing information in an "object oriented" way, e.g. having separate files or directories for the entities—hosts, printers, users, etc.—that you have to manage. The ultimate solution would be using a real database management system to store the relevant information. The required system files could then be *generated* from that database.

Revision control



Who did what, when and why?

You could consider to put all source files, like NIS maps and other administrative files that normally reside in the **/etc** directory, under revision control. Of course there are some disadvantages to this approach. For instance, it introduces overhead because for every modification first the source file has to be locked, then edited and installed in the correct place, tested, and finally checked in again. Furthermore, direct changes to the installed files (e.g. **/etc/group**) must be avoided.

This requires strict discipline from the administrative staff. Despite the disadvantages, it has some major benefits including:

1. Supporting multiple revisions of text—changes no longer destroy the original, because the previous revisions remain accessible.
2. History log—it is easy to find out who did what and why.
3. Resolution of access conflicts—two or more administrators wanting to change the same revision are alerted, preventing one modification from corrupting the other.

SCCS versus RCS

Two major tools are available to support revision control: SCCS and RCS. Both have pros and cons. Instead of going into a detailed discussion about which one is best you could consider picking RCS for the following reasons (the complement applies to SCCS):

1. Allows multiple comment lines on check in.
2. Retains execute permissions on files.
3. Supports symbolic names for revisions.
4. If needed, a lock can be removed by another user than the locker. This avoid blocking work when the locker is unavailable. The original locker is notified through mail.

The major disadvantage of using RCS in favor of SCCS is that you have to build, install and support it yourself.

Of course you could always decide to use SCCS instead and provide workarounds for the cons.

Revision numbering scheme

The revision numbering will use a four level scheme, *R.L.B.S*. *R.L* denote the *release* and *level* number and is kept in lock step with the release and level number of the operating system. For instance, if you are running SunOS 4.1 on the majority of systems, *R* must equal 4 and *L* must equal 1. Of course, this applies to all files under revision control. The *B.S* combination denotes the *branch* and *sequence* number of a file. All files must use the same branch at a given time. The sequence number however is incremented on every change made to a file. Branch numbers are incremented whenever major changes are made to the system (in this case “the network” is the system).

Below, you will find a sample history log. The time stamps have been omitted for convenience.

```
3.5.1.1 martien initial revision
3.5.1.2 martien user john added (also sysadmin)
3.5.1.3 martien user mary added
3.5.1.4 john    user jim added
3.5.1.5 john    user jim removed
3.5.1.6 john    user carl added
...     etc.
3.5.2.1 martien file system reorganization
3.5.2.2 john    user deborah added
...     etc.
3.5.2.23martien user john removed (moved to ABC Corp.)
4.0.1.1 martien conversion to SunOS 4.0
...     etc.
4.1.1.1 martien conversion to SunOS 4.1.1
...     etc.
```

This way, it is easy to mark certain major events in time and it keeps the revision number in sync with the operating system version.

Files under revision control must have a default branch assigned to it so that check out/check in sequences automatically increment the file's sequence number (as opposed to the release and level number).

General procedures

The general procedure to modify these files is:

lock → modify → install → test → accept→ unlock

All files are normally read-only. This means that they cannot be changed without special action (although root can, but shouldn't). In order to modify a file, it needs to be locked first.

Whenever a file is locked, only the locker can modify it, so care must be taken to keep the file locked for short times only. After modification, the file is installed in its normal place so that other tools and utilities can find it. Its new contents is tested against any requirements and finally accepted.

Eventually, the file is unlocked (or checked in) which increments its sequence number. During the check in, you specify the reason for change in a short comment.

3 The art of automounting

Organizing your file system so that you always know what is stored where on your disks and why, could be considered as an art. However, once you have done that, you cannot live without it anymore.

This chapter contains the guidelines and procedures that help you set up and structure the data that you have to maintain for your customers—the end users (note that you yourself are also an end user).

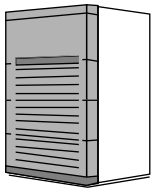
A golden rule for successful system management is: *document the conventions that you use and the things you change and the reasons why.*

The sample configuration

The sections below are based on the following configuration of computer systems interconnected by Ethernet.

Host name:	**bach**
Usage:	Main application server
Type:	SPARCserver 2
Partition:	**/xenon**
Usage:	Third party and unbundled software
Partition:	**/argon**
Usage:	Free and locally built software
Partition:	**/helium**
Usage:	Source code of free and local software

Host name:	**chopin**
Usage:	Mirror application server
Type:	SPARCserver 2
Partition:	**/neon**
Usage:	Redundant copy of **bach:/xenon**
Partition:	**/fluor**
Usage:	Redundant copy of **bach:/argon**, beta software and older versions of software for compatibility



Host name:	**mozart**
Usage:	Production data server
Type:	SPARCserver 490
Partition:	**/krypton**
Usage:	Home and project directories

Host name:	**liszt**
Usage:	Your workstation
Type:	SPARCstation 2 GX
Partition:	**/gold**
Usage:	Very old software and private data

The problem defined

/krypton: file system full

First of all, what are your most important problems right now.

Probably you run out of disk space somewhere in your network almost every day. Besides that, you know that there is a lot of unused disk space available in the network on the various drives local to workstations that you would like to use.

You will probably also need to change the user's command search path once in a while, in order to make a new application available to them. With tens or even hundreds of users and without some organization, this might become a nightmare.

Looking for the files you need all over the place, and not being able to find them is a major frustration. Even worse, if you have to peek in your colleague's private directories—because you know that he or she stores things there instead of a project or workgroup directory—you might feel uncomfortable and hope that you will not be caught snooping.

Plug and Play

Also, installing (*and* uninstalling *and* moving them around) applications, users, systems and other entities on your network is sometimes a hassle. You would probably like to plug and play, meaning that you unpack your new hot box, plug it into the mains and into the network and run! The setup should be so, that users and applications do not suffer from the various underlying hardware platforms and operating systems, users should be able to use them transparently.

Data classes and instances

Most of the problems are caused by the various classes, versions and variants and locations of data. Below, you will find a few examples of data classes and some considerations about how to use them.

Home directories: /home/*user*

To start with, home directories are a good example of a distinct data class. Preferably, a home directory looks something like **/home/mary**. This is easy to remember, refers to a real person and does not have any "foreign" data clobbering it.

For instance, in the conventional Sun file system set up, you use home directories of the form **/home/mozart/mary** where **mozart** is the name of the NFS server of the home directory. However, you will have to do a lot of work if you decide to move Mary's home directory to another NFS server. So why don't you leave out this sensitive information in the first place?

This means that in general, home directories have the form **/home/***user*, where *user* is replaced with the login name of that user.

Note that if you are planning to implement this concept, you will have to be careful not overmounting the conventional **/home** directory, which is normally used as mount point for the home partition, e.g. **sd0h**.

Project directories: /project/*project*

Another specific data class could be the various projects that are being done in your environment. You should apply the term "project" for a wide scope of activities, not just software engineering projects.

For example, a project that studies NFS performance could be called **nfstune**. Another project that deals with personnel administration could be called **humadm**. Yet another project is responsible for system administration (yes, that's you), let's call it **sysadm**.

Having these examples, it would be no more than logical (and intuitive) to use separate project directories for them, and their

corresponding names then would be **/project/nfstune**, **/project/humadm** and **/project/sysadm**.

In general you will have **/project/***project*.

Applications, tools utilities and other read-mostly data: /vol/*volume*

As a third example, consider the various applications, tools and utilities that you make available for your users. In general, these "things" are used in a read-only fashion. For example, you could support tools like FrameMaker, GNU Emacs, Console Tool, Rolo Tool and Catcher[1] on your network. Also, you want these tools to be available for everyone on the network.

These applications are normally made available through a path name that contains the application's base name as last part. FrameMaker is then called **frame**, GNU Emacs is called **emacs**, and the others **contool**, **rolotool** and **catcher** respectively.

Following the trend set in the first two examples, you could consider providing these applications through, say, **/applic**, **/tool** and **/util**. However, most of these applications are used in more or less the same way and perhaps even some other data (i.e. non-applications, like include files, icons, FrameMaker document templates) are used in the same way. So why not just keep it simple and make them available as *volumes* through **/vol**: a clod of data that can be treated as *one single unit*.

To complete our example, you would have directories like **/vol/frame**, **/vol/emacs**, **/vol/contool**, **/vol/rolotool** and **/vol/catcher** respectively.

In general you would have **/vol/***volume*.

A major advantage of the single unit approach is that you can move these units around on the network to the most convenient location very easily, while still maintaining the same logical access path. (If you are wondering how? Read on.) Besides that, treating applications and the like as units, means that they do not "infect" your file system on other places than just the installation directory. This means that uninstallation becomes a piece of cake: just remove the installation directory and any references to it. You don't have to track down any other places in your file system that may have been touched during the initial installation of the application.

Software distributions: /distrib/*distribution*

Suppose your business includes distributing software. For instance, you deal in FrameMaker, you have your own developed software that you want to cut tapes from, you support a few distributions for friends, etc. Consider making these things available through **/distrib/***distribution*. For instance **/distrib/frame** (note the difference between this version and **/vol/frame**), **/distrib/nfstune** (the product that results from the corresponding project discussed above), **/distrib/xview** (the XView source code that you want to be able to give your friends).

Again, these distributions are used in a read-only fashion and they can be treated as units. You could develop scripts that follow this convention in order to create tapes or floppies ready for distribution. You could also consider putting less frequently used distributions on partitions that are normally not used to store production data. For instance, you could use the partitions local to workstations and that are normally not used for other purposes.

And more: /source, /beta...

Using the same concept, you could go on. Supporting **/source** for building and installing public domain and free software, **/beta** if you want to distinguish between general (or "production") applications and beta releases of software that you want to make available.

In general: /*class***/***instance*

In general, you will have **/***class***/***instance* as a path name to get to a specific instance of data in your network. Note that it is only two levels deep!

1 Catcher is a nice utility developed by Chuck Musciano that supports the drag and drop of files on custom applications and shell scripts.

This is easy to remember, especially if you use the right names and set it up consistently.

You can promote data at the instance level to class level if the need is there. For example, you start out with supporting **/vol/distrib** for software distributions. But after a while, this directory becomes so large and unmanageable that you decide to split it up into separate distributions. This is the moment that you promote **/vol/distrib** to **/distrib**. Of course, the other way around is also possible.

Versions and variants

Other major problems are the various versions and variants of applications.

For example, at a certain point in time you must support FrameMaker 1.3b SunView (because there are still users that did not have the upgrade training to 2.0), FrameMaker 2.1 SunView (because there are still users who are not running OpenWindows) and FrameMaker 2.1 for the X Windows System, the default.

At the same time, OpenWindows 2.0 is the default windowing environment and OpenWindows 3.0 beta is available for those who cannot wait.

Besides that, you need to support a few free software tools—like GNU Emacs—on multiple architectures, Sun-4, Sun-3 and Sun386*i*, say.

All of these instances are variations of a functional equal application. How do you organize these things? A very simple approach would be to always let */class/instance* be the default or current version of a data instance. For example, **/vol/frame** corresponds with the FrameMaker 2.1 X Window System version, **/vol/openwin** corresponds with OpenWindows version 2.0 and **/distrib/xview** corresponds with the XView 2.0 source code contribution of Sun to the MIT X Windows System distribution.

Now, if you also want to make non-default versions available and still be able to distinguish between them, add the version number to the default name, separated by a hyphen. So, FrameMaker 1.3b (the SunView version) would then be available via **/vol/frame-1.3b** and **/vol/openwin-3.0-beta** is the beta release of OpenWindows version 3.0.

The hyphen that separates the base name from the version number helps you distinguish between the two, e.g. consider using Lotus 1-2-3 version 1.0: **/vol/lotus1231.0** versus **/vol/lotus123-1.0**[1].

In order to maintain consistency, you should also support explicit version numbering for the default version. This means that in the example above, you would have both **/vol/frame** and **/vol/frame-2.1** available (which of course both refer to the same installation).

Putting it all together

The conventions described in the previous section can be realized by using techniques and tools available in SunOS, NFS and NIS[2]. To be specific, it requires both servers and clients to be able to use NFS and NIS and the program that glues them together and exploits their capabilities: *automount*(8)[3]. The automount process, once started from **/etc/rc.local**

1 You will have a slight problem if you decide to use the base name **lotus-1-2-3**, this leads to the name **/vol/lotus-1-2-3-1.0**.

2 Of course you can implement these concepts in other similar environments as well, provided you have the appropriate tools and technologies at hand.

3 If you do not have the automounter available, you could consider using *amd*. This is available as free software and has the same functionality as the automounter.

during boot time, acts like a name to location server. Its behavior can be almost completely controlled by the contents of some specific NIS maps. The next section, “How to set up your system” shows you the details on how to do this.

How to set up your system

This section gives you a look under the hood. It details on how to partition your disk, exporting and importing[1] data and how to glue it all together using the automounter, NFS and NIS.

This section applies to the general case and does not describe on how to organize your own applications or public domain or free software. For more information on the latter topic, please refer to “Application management” on page 3-10.

Partitioning disks

When partitioning your disks, you should consider wether or not the data that you are going to store in that partition is going to be used in a read-write or read-only fashion. If it is going to be used as read-only in most of the cases, then you will never have to include that partition in you backup scheme. You only need to archive the data once, so that you can recover from loss of data.

Note that installed third party applications and public domain and free software, as well as software distributions, local include files, etc., are used read-only most of the time. Besides that, these installations can total up to tens, even hundreds of megabytes. Megabytes that you do not have to backup! You only have to backup “production” data like home and project directories and perhaps databases.

Another point of consideration is separating your data from third party data. Third party data is data that you get from some other party, for example a new version of the operating system, or a new version of your publishing software, but also a new version of a locally built tool. After having installed third party software, you often have to customize it in order to match your local needs. If you need to customize it, try to avoid changing files in the installation directory, or keep the required changes to a minimum. Every time that you re-install the software you have to re-apply the changes. Instead, try storing those changes in a directory separate from the installation directory and tell the third party software to look there for specific files, e.g. by setting the appropriate environment variables or by creating the appropriate symbolic links in the installation directory.

As an example, consider the customization of FrameMaker with respect to the use of locally developed document templates. FrameMaker normally looks for document templates in its own installation directory. If you replace the directory that FrameMaker looks in by a symbolic link to a place that *you* maintain, FrameMaker will use the contents of that location instead. In order to be compatible with the document templates that FrameMaker supplies, you could create a symbolic link back to the, now renamed, original template directory. Although you create two extra

1 Note that in this document the term *import* is used in favor of the term *mount* in order the emphasize the function rather than the operation.

symbolic links, you have the advantage that the next time you install FrameMaker, you only have to create one symbolic link, and everything works as before. You don't have to worry about saving your templates before removing the old installation of FrameMaker.

Mounting local file systems

After having partitioned your disks and having created file systems in the appropriate partitions, you have to mount them at boot time to make them available to you local system. This is done by means of the file **/etc/fstab**.

By convention, the mount points that are used in this document use element names from the periodic table, like "krypton", "xenon" and "helium". The advantage of choosing real names that are unique within your domain or network is that you can uniquely identify a partition (or file system) in the network. There is never any doubt about which partition you mean when you talk about the fact that, say, **/krypton** is full again. The use of the name class of elements from the periodic system should provide for enough distinct names in your network, it contains well over 100 element names.

The generic name */element* always denotes one single physical file system. Mapping this physical name to a more logical name is done during the export and import of (parts of) the file system by the automounter.

Subdividing partitions

Within the local file systems, you can create the subdirectories for the data classes that you want to store on these partitions. For example, if you want to support your applications and other read-only data on **/xenon**, then you should create the directory **/xenon/vol**. And if you want to support home and project directories on a partition called **/krypton**, then you should create the directories **/krypton/home** and **/krypton/project**.

Exporting data

If you have created directories and installed software, users and projects, and you want to make this data available to the network, you have to export them. If you do not export this data, others cannot import it.

Exporting data can be done at three levels: at the partition level, at the class level and at the instance level. For example, you could export the complete **/gold** partition. You could also export home directories at the class level by exporting **/krypton/home**, for example. Finally, you could export volumes (applications, etc.) at the instance level: **/xenon/vol/openwin-2.0**.

The level at which you export data depends on the granularity of control you want to have on your data. The lower the level, the more control you have. For instance, if you export volumes at the instance level, then you can specify the export options on the finest level. In this case you can specify for each instance if it is exported read-only, read-mostly, with root access, which clients can access it, and so forth. Root file systems and swap files for diskless clients, for instance, are exported at the finest level. You could also use it in combination with *netgroup*(5) in order to allow or disallow certain groups of machines or users access to specific data.

Importing data

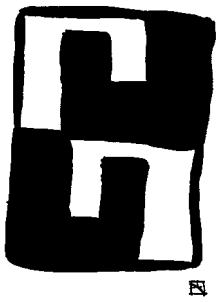
If you are a client of data provided by file servers on the network, you should follow the conventions defined in the previous sections. This means that if, for example, you need FrameMaker to be available on your system, you have to import that volume from the server. This importing is done by issuing the *mount*(8) command or by specifying the appropriate entries in **/etc/fstab**. A sample command to import FrameMaker would look like:

```
mount bach:/xenon/vol/frame-2.1 /vol/frame
```

assumed that FrameMaker is installed on the NFS server **bach** on a partition that is mounted on **/xenon** and that the local directory **/vol/frame** exists[1]. In general you would use:

```
mount server:/partition/class/instance /class/instance
```

Instead of doing these mounts manually or during boot time, you can exploit the automounter's capabilities to automate this process. How to do this is covered in the following sections.



Simplicity & Consistency

Please note that NFS servers can be NFS clients at the same time, even of them selves. In order to maintain consistency and simplicity, you should set up your system so, that NFS servers are treated the same way NFS clients are. There should be no difference except for the fact that servers serve files.

Using the automounter

Instead of importing or mounting NFS directory hierarchies manually or during boot time, you can use the automounter to automate this process. For a description of how the automounter works, please refer to the manual page: *automount*(8). For an in-depth discussion of the automounter, you should refer to chapter 15, *Using the NFS automounter* from *System & Network Administration*.

By default, the automounter looks for a map with the name **auto.master** in the current NIS domain. If this map exists, it will consult this map and use this map as a list of initial automount maps (consider it a meta map). The layout of the **auto.master** map is as follows:

```
/mountpoint       mapname  [mount options]
```

By convention, all the maps that are specific for the automounter have **auto.** as a common prefix.

For instance, to follow the examples used in the previous section, our **auto.master** could look like this:

```
/vol     auto.vol          -ro,nosuid,hard,intr
/project auto.project      -rw,nosuid,hard,intr
/home    auto.home         -rw,nosuid,hard,intr
/distrib auto.distrib      -ro,nosuid,hard,intr
/net     -hosts            -ro,nosuid,hard,intr
/-       auto.direct       -ro,nosuid,hard,intr
```

As you will notice, most of the imports are done read-only and without set UID on execution by default. Importing data read-only is cheaper than importing it read-write, and importing data without the set UID on

1 To take it one step further, you could even create an alias in the hosts map that aliases a hostname to a partition. In this case you would map the host alias **xenon** to **bach** and the appropriate mount command would then be **mount xenon:/xenon/vol/frame-2.1 /vol/frame**.

execution semantics avoids one of the security violations that Trojan horse attacks like to use.

As we will see in later examples, you can override this default on special cases in the appropriate maps.

All of the maps used in the above example will be explained below.

To start with, the contents of the **auto.vol** map could look like this:

```
emacs              bach:/argon/vol/emacs-18.55\
                   chopin:/neon/vol/emacs-18.55
frame              bach:/xenon/vol/frame-2.1\
                   chopin:/neon/vol/frame-2.1
bin                bach:/argon/vol/${ARCH}/bin\
                   chopin:/neon/vol/${ARCH}/bin
man      -rw,nosuid,hard,intr\
                   bach:/argon/vol/man\
                   chopin:/neon/vol/man
openwin  -ro,suid,hard,intr\
                   bach:/xenon/vol/openwin-2.0\
                   chopin:/neon/vol/openwin-2.0
emacs-18.55        bach:/argon/vol/emacs-18.55\
                   chopin:/neon/vol/emacs-18.55
frame-2.1          bach:/xenon/vol/frame-2.1\
                   chopin:/neon/vol/frame-2.1
frame-1.3b         chopin:/fluor/vol/frame-1.3b
openwin-2.0 -ro,suid,hard,intr\
                   bach:/xenon/vol/openwin-2.0\
                   chopin:/neon/vol/openwin-2.0
openwin-3.0-beta -ro,suid,hard,intr\
                   chopin:/fluor/vol/openwin-3.0-beta
```

Taking the first entry, **emacs**, as an example, here is how the automounter works. The automounter acts like a NFS file server. In this example, it intercepts any reference to **/vol**. As soon as a process refers to something (**emacs**) under this directory, the automounter searches the corresponding map, **auto.vol** in this case. When found, it will import the directory hierarchy from the location(s) specified in the last column. What it will do exactly in this example is shown in the following commands (assuming the reply to the import request is received from **bach** first):

```
mount -o ro,nosuid,hard,intr\
         bach:/argon/vol/emacs-18.55 /tmp_mnt/vol/emacs
ln -s /tmp_mnt/vol/emacs /vol/emacs
```

The **bin** entry will be explained in the section "Application management" on page 3-10. Note that the **man** entry will be mounted read-write in order to be able to store formatted pages in the corresponding **cat** directories if necessary. Also note that **openwin** is mounted with the set UID on execution option turned on in order to support the "MIT-MAGIC-COOKIE" security.

The map used in this example also provides redundant locations for most of the volumes. In this case, they can both be imported from either **bach** or **chopin**. This redundancy can lead to a more reliable and robust environment. This kind of redundancy is of course only useful for read-mostly data.

Finally, note that besides the default version, also the explicit versions are provided for those who need them.

The exceptions to the rules set in the **auto.master** map stand out against the other entries. This gives you an instant overview of those parts that may require special considerations during the installation phase.

The **auto.project** map looks like:

```
nfstune          mozart:/krypton/project/nfstune
humadm           mozart:/krypton/project/humadm
sysadm           mozart:/krypton/project/sysadm
```

Nothing special about this one, except that all projects appear to be served by host **mozart**.

The **auto.home** map looks like:

```
john             mozart:/krypton/home/john
mary             mozart:/krypton/home/mary
graig            mozart:/krypton/home/graig
```

Also quite simple. When John logs in, **/krypton/home/john** will be imported from **mozart**.

The **auto.distrib** map looks like this:

```
xview                    bach:/xenon/distrib/xview-2.0
emacs                    bach:/argon/distrib/emacs-18.55
xview-1.0                liszt:/gold/distrib/xview-1.0
xview-2.0                bach:/helium/distrib/xview-2.0
```

In this case, the XView version 1.0 distribution is also supported. However, it could be that this instance is stored on a file server that is normally not used for intensive file server operations. Perhaps it is stored on a disk local to a workstation somewhere in the network, just for convenience and without the need to make backups at the appropriate times (in this case **liszt**, *your* workstation).

The entry that specifies **/net -hosts** is special. It causes a reference to **/net/***host* to mount *all* directory hierarchies exported by that host. Please note this can lead to a high system and network load and can take a long time if the host specified exports many directory hierarchies.

The automounter consults the NIS map **hosts.byname** for the host specified on the command line.

Finally, the **auto.direct** map is used to make single directories available in between existing ones. As an example, consider a direct mount of **/var/spool/mail** and **/var/spool/calendar** from a central file server that serves your mailbox and network agenda. The entries in the **auto.direct** map would look like this:

```
/var/spool/mail          mailhost:/var/spool/mail
/var/spool/calendar      calendarhost:/var/spool/calendar
```

References to either directory by Mail Tool and Calendar Manager respectively, would cause the appropriate directory to be imported from the server.

As another example, consider a transition phase: your are in the process of switching from the old file system organization to the new one. In the old case, you supported a project that had its directory under **/usr3/zis**. In the new set up you are going to make this project directory available under **/project/zis**. What you can do is support both access paths during the transition phase so that the engineers can adapt their scripts and Makefiles to the new set up.

In order to do so, include the following line in **auto.project**:

```
zis                      mozart:/krypton/project/zis
```

and include the following line in **auto.direct**:

```
/usr3/zis                mozart:/krypton/project/zis
```

As you can see, both locations refer to the same project directory.

Another conveniency of the automounter is that it will create all directories needed for you. It will also remove them (and the corresponding links) automagically after a certain period of inactivity or when the system is shut down. You do not have to do a thing.

Automount inconveniences

There are a few inconveniences when using the automounter that must be mentioned.

First of all, the automounter mounts all directory hierarchies under **/tmp_mnt**, and creates a symbolic link to that location. This means to you will often see paths that start with the unaesthetic **/tmp_mnt**. Try the command *pwd*(1) for example.

Second, the locations that the automounter watches are initially empty. Only on reference, entries are created. This means that if you type, say,

```
ls /vol
```

you will not see anything the first time. As another result, file name completion supported by some utilities (C and Korn shell and Emacs for example) does not work in these directories initially. You must explicitly type in the full name that you need.

Last, since the logical location contains nothing but symbolic links to the actual mount point, commands like

```
cd /home/mary/../john
```

are likely to fail. Remedy: always use full path names in this case.

Application management

If you need to support software for more than one hardware platform—which is normally the case in a heterogeneous environment—a well designed directory structure is what you need.

This section discusses a model that allows you to do so.

First of all, remember that applications are made available as volumes in the network. The first part of this section describes how to organize such a volume.

The second part of this section describes a way to make the commands available to end users in such a way that the end user does not have to be bothered with the underlying hardware and operating system platform. It is completely transparent to him or her. Furthermore, the second part will discuss a concept that avoids the need to change the shell's search path for every command that you want to make available.

Application directory organization

This section describes the preferred directory structure for applications.

In general, applications normally have architecture dependent and shareable, architecture independent, data. Furthermore, it is considered

good behavior if the application does not require to be able to write in its own installation directory during normal operation.

The directory structure described below results in a complete separation between shareable and architecture dependent data. The directory structure is as follows:

bin.*arch* Architecture dependent directory containing executable end-user programs for that architecture. Contains symbolic links to end-user commands stored in **../script**. Real copies create unnecessary redundancy and should probably be avoided for the sake of consistency.

script[1] Directory containing scripts (shell, awk and others) that can be shared across architectures. These scripts are also part of the end-user commands.

etc.*arch* Architecture dependent directory containing executable programs for that architecture. These programs are normally not used by end-users directly, but rather by the application itself or by an application administrator.

lib.*arch* Architecture dependent libraries and other resource files that could be used by the application itself or perhaps by software developers.

Besides these general directories, you could of course support directories for help texts, fonts, lisp sources, etc. For more information, please refer to "A complete example: GNU Emacs" on page 3-18.

Having this application directory structure, it can be made available as a volume. GNU Emacs for example, would become available as **/vol/emacs** and it would support the directories **/vol/emacs/bin.sun4** **/vol/emacs/etc.sun4**, **/vol/emacs/info** and **/vol/emacs/lisp**, say.

If the application itself needs to access its own files during operation, it must be built so that it refers to its installation directory, and not to some other place in the file system. For instance, GNU Emacs refers to its own Lisp files during execution, and it should use the path **/vol/emacs/lisp** to access them. *Not* something like **/usr/local/emacs/lisp**.

Making the application's commands available

In general, applications, tools and utilities exist of one or more end user commands. These commands can be typed at the shell prompt, and the shell will try to execute that command. The shell searches for commands in the directories specified in the PATH environment variable[2].

This means that one way to tell the shell that you have added a set of new commands is to add an entry to its search path. For example, if you have added GNU Emacs as a volume, you could make the commands available by executing the following commands:

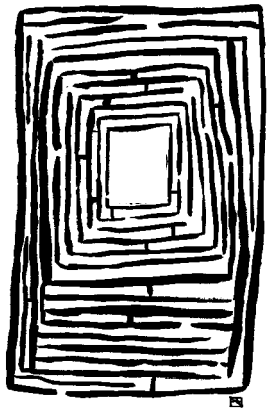
```
PATH="${PATH}:/vol/emacs/bin.'arch'"
export PATH
```

However, this has major disadvantages.

First of all, you need to do this for all the users and all the shells (Bourne, C and Korn) that want to have the Emacs commands available. This can be quite a hassle and is hard to maintain.

1 Note that the name of this directory is in singular form, **script** versus **scripts**. After all, you also don't use directory names like **/usr/bins** or **/homes**.

2 In the examples mentioned, only Bourne shell (*sh*(1)) syntax is used. For information on the C and Korn shell, please refer to the corresponding manual pages.



The shell's search path.

Second, the search path gets larger and larger on every such occasion, finally resulting in a twisty little maze containing dead ends as well.

And last but not least, the search path contains architecture dependent parts. This is no real problem, but it is somewhat unaesthetic and it can be avoided as you will see below.

In order to solve the problems mentioned above, you can create a "convenience" bin directory that only contains symbolic links to the actual commands. To resolve the architecture dependency, you can create such a directory for every architecture that you need to support[1]. This directory is also made available as a volume and users only need to incorporate this directory in their search paths.

Suppose you want to support the **emacs** command. Then, what you should have is the following symbolic link:

/vol/bin/emacs → **/vol/emacs/bin.***arch***/emacs**

Here, *arch* should be replaced with **sun3**, **sun4** or **sun386**, whichever is appropriate. The shell's search path must of course include **/vol/bin** in order for the shell to find the command.

Note that both **/vol/bin** and **/vol/emacs** are made available through the automounter.

The specific steps required to create this behavior in this example are described below. But please note that more general procedures are described in "General procedures" on page 3-14.

For example, suppose you want to add the Emacs commands for Sun-3, Sun-4 and Sun386*i* architectures to the environment.

First, you have to build GNU Emacs and it should have the directory structure described in the previous section. You make this instance of Emacs available as a volume on a file server:

```
mkdir -p /partition/vol/emacs-18.55
```

Then export this volume by adding the appropriate line to **/etc/exports** and running *exportfs*(1). Finally, add the following lines to the **auto.vol** map and propagate the changes on the network:

```
emacs           server:/partition/vol/emacs-18.55
emacs-18.55     server:/partition/vol/emacs-18.55
```

Conclude this step with installing the previously built Emacs in this directory on the file server.

Next, what you do is create a volume for the architecture dependent bin directory on a file server if it does not already exist:

```
mkdir -p /partition/vol/arch/bin
```

Note that the architecture dependencies are reversed, or turned inside out, in this occasion. That is, the *arch* directory is located one directory lever higher than the **bin** directory. This concentrates all architecture dependent parts under one single exported directory. If you need to, you can import this single directory later on **/usr/local**. If you also support **lib** and **etc** directories, **/usr/local** becomes what you were used to until now.

As with the Emacs volume, export it and make it available to file clients[2] by modifying and propagating the **auto.vol** map. The trick to

1 It could also be done by "auto-overmounting". However, this technique is more expensive than the one described here and is therefor not discussed.

2 Note that this document prefers to use the term *file client* to be consistent with the term *file server*

resolve architecture dependencies is in this map! You should add the following entry:

```
bin                server:/partition/vol/${ARCH}/bin
```

The variable **ARCH** is set by the automount process at start up (so, during boot time). It's value reflects the application architecture. For example, for a Sun-3 it will equal **sun3** and for a Sun-4 it will equal **sun4**.

This means that the directory imported depends on the architecture of the file *client*.

Finally, create the appropriate symbolic links in the bin directories:

```
cd /partition/vol/arch/bin
ln -s /vol/emacs/bin.arch/* .
```

Repeat these command for all the architectures that you support and you're done!

Please note the final dot at the end of the second command. It is essential that you specify it in order to create all the appropriate symbolic links in one step.

What happens when you invoke a command?

Suppose you invoke the command:

```
emacs
```

The following things will happen. The shell searches through its path and will find **/vol/bin/emacs**. Since **/vol/bin** is handled by the automounter, it will be imported from the appropriate file server on first reference. Remember that **/vol/bin** is architecture dependent. That is, it will come from the location *server*:/*partition*/**vol**/*arch*/**bin**.

Because **/vol/bin/emacs** is a symbolic link to **/vol/emacs/bin.***arch*/**emacs**, **/vol/emacs** will also be imported by the automounter and the shell will finally execute the correct architecture dependent command.

The Emacs process itself will reference its private files also through **/vol/emacs**.

Volumes for other purposes

Besides using volumes for applications, you can also use them for other things like an architecture dependent bin directory (as described in the previous section), a network wide tmp directory, network wide user initialization and prototype files, include files, manual pages, etc.

This section discusses some of these topics.

A network wide tmp directory

It is very convenient to have a network wide tmp directory available. It provides an easy way to transfer files between colleagues without having to use each other's home directories. Name this directory **/vol/tmp** and set the access permission to **rwxrwsrwt**. The physical location of this directory could be anywhere on the network and you never have to back it up. Consider putting it somewhere in a partition also containing swap files for diskless clients.

Make sure you remove stale files once in a while to avoid filling up the file system.

User initialization and prototype files

Consider supporting a directory **/vol/default** with the subdirectories **init** and **proto**. You could put **.login**, **.cshrc**, **.profile**, **.openwin-menu** and a bunch of other files in the **init** directory that users source during their login sequence. You could install prototypes of these files in the **proto** directory that are used for new accounts added to your system. Et cetera.

Network wide shell scripts

Consider supporting **/vol/script** for network wide shell, awk, sed, perl and other scrips.

Local manual pages

Consider supporting **/vol/man** with the same structure as **/usr/share/man** (or **/usr/man**). Create symbolic links to the appropriate places from other volumes that you support, e.g.

```
cd /partition/vol/man/man1
ln -s /vol/emacs/man1/* .
```

To let the *man*(1) command find these manual pages, set your manual search path accordingly:

```
MANPATH="/vol/man:/usr/share/man"
export MANPATH
```

Include files, icons and others

Make your favorite local include files and icons available in **/vol/include** and **/vol/icon** respectively. Create support for **/vol/frame** (FrameMaker), **/vol/fmtemplates** (your local FrameMaker document templates), **/vol/wp** (WordPerfect), **/vol/guide** (Devguide), et cetera!

It's up to you. Feel free to exploit **/vol**! Use your imagination.

General procedures

This section describes the generic steps that are required to create an environment defined in the preceding sections. The steps described here can be implemented immediately. Even an ignorant user with the appropriate privileges can execute the procedures described here with success.

√ Root privileges that are required during certain operations are explicitly marked with a square root symbol (√). All other operations normally do not require root privileges, depending on the access permissions and the owner of the files and directories that have to be modified.

In order to get root privileges, invoke the *su*(1) command as follows:

```
/usr/bin/su -
```

The "-" option performs a complete login. It removes all variables from the environment except for **TERM**, sets **USER** to user name, sets **HOME**

and **SHELL** as specified in the password file, sets **PATH** to **:/usr/ucb:/bin:/usr/bin**, changes directory to root's home directory, and tells the shell to read root's **.login** or **.profile** file. This will be more secure and avoids Trojan horse attacks, provided that you have set up root's environment accordingly.

Recall that some examples of data are applications, home directories of end-users, production data like documentation, source code etc. A class instance of a home directory could be **john**. A class instance of an application could be **frame**. Classes are made available through the directory **/***class*. A class instance can then be accessed through **/***class***/***instance*.

For example, the class of home directories could be made available through **/home** and John's home directory would be **/home/john**. A general home directory would be **/home/***user*. As another example, applications could be made available as so called volumes through **/vol/***applic*. E.g. FrameMaker would then be **/vol/frame**.

Adding a data class

In order to add a class, you will have to add an entry to the NIS **auto.master** map that describes the class.

1 Add an entry for the data class

Add the following line to the file **auto.master**:

```
/class   auto.class        [-mount options]
```

√ **2** Propagate the changed map

In order for NIS clients to recognize the new class, you have to propagate the changes. In order to do so, you have to convert it into *dbm*(5) format by using *makedbm*(8). This can be done as follows:

```
/usr/etc/yp/makedbm auto.master\
        /var/yp/domain/auto.master
```

If you have configured slave NIS servers, then you also need to push the changed map to them[1]:

```
/usr/etc/yp/yppush auto.master
```

However, normally these steps are included in a Makefile so that running *make*(1) in the directory containing the source of the maps will perform all necessary steps required to propagate the changes on the network.

Adding a data class instance

This section describes the general procedure that adds data instances, e.g. a user's home directory or an application's installation directory.

1 Note that the NIS master server does not actually *push* the map to the slave server. In fact, the master server sends a multicast message to the slave servers that tells them to issue a subsequent *ypxfr*(8) command that *fetches* a copy of the map.

Create the required physical directories on the file server

1 Login on the file server

In order to be able to create the directory with the appropriate properties, you must be logged in on the appropriate file server.
The subsequent steps will refer to *server* in order to denote this NFS file server.

2 Create the physical directory for the class if it does not exist

Create the directory that will be used to store all the instances for this class (remember that */partition* below is the mount point of a disk partition containing a 4.2 file system. *Partition* should be replaced with an element name from the periodic system).
Make sure that you give it the appropriate properties like owner and group of the directory and access permissions. Refer to the specific class descriptions for details on these properties. The commands below are only an example and will leave the directory accessible for everyone, but only writable for the owner and the group of the directory.

Perform the following commands in order to create the class' directory if it does not already exist:

```
mkdir -p /partition/class
chown class-user.class-group /partition/class
chmod ug=rwx,o=rx /partition/class
```

You have to repeat this step once for each partition that is going to support data instances of this class.

3 Create the physical directory for the class instance

Next, create the directory for the class instance. As with the class' directory, make sure all properties are conform the conventions that you have set up.

Perform the following commands in order to create the instance's directory:

```
mkdir -p /partition/class/instance
chown instance-user.instance-group\
        /partition/class/instance
chmod ug=rwx,o=rx /partition/class/instance
```

This completes the steps required to create a directory for the instance. All data must be stored in this physical directory.

Export the directory

In order to make the directory available to file clients, you must first export it. This is done by adding an entry to **/etc/exports** and exporting the directory.

√ **1** Add the directory to **/etc/exports**

In order to be able to actually export the directory, you must add it to the file **/etc/exports**. You can do this by adding the following line to that file:

```
/partition/class/instance [-export options]
```

You could keep the entries sorted in this file for convenience. Furthermore, you can specify a number of export access options on the same line, like root access, read-only access, netgroup access etc. For more information on these options refer to *exports*(5).

You could consider storing the original version of **/etc/exports** on another partition than the root partition. If you maintain it there, you can simply copy the modified version to **/etc**.

Export the directory by executing the following command:

```
exportfs -v /partition/class/instance
```

The "-v" option makes the command verbose, so you can actually see the export take place.

At this point in time, file clients can actually mount the directory hierarchy from the file server.

Change the NIS map auto.*class*

In order to make the directory available through */class/instance*, you will have to change the appropriate NIS map in order for the automounter to recognize the new instance. Remember that changing NIS maps can only be done on the NIS master server.

1 Login on the NIS master server

2 Add an entry for the class instance

Add the following line to the NIS map **auto**.*class*:

```
instance [/mountpoint] [-mount options]\
         server:/partition/class/instance
```

The optional *mountpoint* is taken as a path name relative to the destination of the instance. If the mountpoint is omitted, a mountpoint of **/** is implied.

You can also specify mount options that take precedence over the default mount options specified in the **auto.master** map. This way you create exceptions to the default mount options specified in the master map.

√ **3** Propagate the changed map

The commands for this step are equal to those described in step 2 in "Propagate the changed map" on page 3-15. Please refer to that description.

This completes the steps required to add a data instance to the network.

The instance is now available on all file clients in the current NIS domain through */class/instance*.

Summary: it is easy

Looking back at the previous sections that describe the procedures needed to install new data classes and their instances, it appears that it really *is* simple to implement and maintain. It only requires a handful of steps to do so. From these steps, only three of them essentially require super user privileges.

Given this framework, you should be able to define the steps required to move data instances to another partition or server or to remove a data instance from the network. Good luck!

A complete example: GNU Emacs

This section contains an overview of the steps required to build and install GNU Emacs according to the rules laid out in the previous sections.

The description is kept terse so you can concentrate on the commands. It uses the following environment: the NIS domain is **amersfoort.Holland.Sun.COM**. The main file server that is used for applications is a Sun-4 machine called **bach**. The partition that is used to store sources of free software is mounted on **/helium**. The partition that is used to store locally developed, built and maintained applications, tools and utilities is mounted on **/argon**. Furthermore, **mozart** is the master NIS server and there are no NIS slave servers. **mozart** is also used as the file server for home and project directories. Your own SPARCstation 2 is called **liszt**. This is the first time that you do this, so you really start from scratch.

The example below combines multiple steps into one, as opposed to doing it phase by phase as described in "General procedures" on page 3-14. The steps are only executed for the main application server (**mozart**). In order to mirror the changes on **chopin**, you should make duplicates of the directory hierarchies created in these steps. This simplification is done for convenience, clarity and efficiency.

It assumes that you want to be able to create source distributions for other friends using **/vol/distrib**, that you will compile and build it in **/source/emacs**, that you will support the manual pages in **/vol/man** and that the Emacs command suite will be made available through **/vol/bin**

Set up the directory structure and export it accordingly

1. Login on the **bach** (the file server for applications)
2. Create the necessary physical directories to build, install and use Emacs

```
mkdir -p /argon/distrib/emacs-18.55\
         /helium/source/emacs-18.55\
         /argon/vol/emacs-18.55\
         /argon/vol/sun4/bin\
         /argon/vol/man\
         /argon/vol/man/man1\
         /argon/vol/man/man3\
         /argon/vol/man/man4\
         /argon/vol/man/man5\
         /argon/vol/man/man6\
         /argon/vol/man/man7\
         /argon/vol/man/man8\
         /argon/vol/man/cat1\
         /argon/vol/man/cat3\
         /argon/vol/man/cat4\
         /argon/vol/man/cat5\
         /argon/vol/man/cat6\
         /argon/vol/man/cat7\
         /argon/vol/man/cat8

chmod a=rwx /argon/vol/man/cat[13-8]
```

3 Create the appropriate lines in **/etc/exports**

Add the following lines to **/etc/exports**:

```
/argon/distrib/emacs-18.55
/helium/source/emacs-18.55
/argon/vol/emacs-18.55
/argon/vol/man
/argon/vol/sun4/bin
```

4 Export the directory hierarchies

```
/usr/etc/exportfs -va
```

Read the Emacs distribution tape

Still on **bach**, read in the tape with the Emacs distribution you got from your friend.

1 Read in the tape

```
cd /argon/distrib/emacs-18.55
tar xvf /dev/rst0
```

This results in a file called **emacs-18.55.tar.Z**. This file will later be available anywhere in your domain through **/distrib/emacs/emacs-18.55.tar.Z** as well as through **/distrib/emacs-18-55/emacs-18.55.tar.Z**.

Configure the automounter

1 Login on the NIS master server (**mozart**)

2 Create the **auto.master** file

Create the file **/etc/auto.master** with the following contents:

```
/distrib auto.distrib      -ro,nosuid,hard,intr
/source  auto.source       -rw,nosuid,hard,intr
/vol     auto.vol          -ro,nosuid,hard,intr
```

3 Create the **auto.distrib** file

Create the file **/etc/auto.distrib** with the following contents:

```
emacs-18.55       bach:/argon/distrib/emacs-18.55
emacs             bach:/argon/distrib/emacs-18.55
```

4 Create the **auto.source** map

Create the file **/etc/auto.source** with the following contents:

```
emacs-18.55       bach:/helium/source/emacs-18.55
emacs             bach:/helium/source/emacs-18.55
```

5 Create the **auto.vol** map

Create the file **/etc/auto.vol** with the following contents:

```
emacs-18.55       bach:/argon/vol/emacs-18.55
emacs             bach:/argon/vol/emacs-18.55
bin               bach:/argon/vol/${ARCH}/bin
man      -rw,nosuid,hard,intr\
                  bach:/argon/vol/man
```

6 Propagate the new maps to the NIS clients

```
cd /var/yp/amersfoort.Holland.Sun.COM
/usr/etc/yp/makedbm /etc/auto.master auto.master
/usr/etc/yp/makedbm /etc/auto.distrib auto.distrib
```

```
/usr/etc/yp/makedbm /etc/auto.source auto.source
/usr/etc/yp/makedbm /etc/auto.vol auto.vol
```

As of this moment, you have the data classes and instances just created available on all the systems in the current NIS domain.[1]

Build it

The building of the targets that comprise the Emacs volume can be done on your personal workstations, so:

1 Login on **liszt** (your workstation)

2 Switch to the directory were you will compile and link Emacs

```
cd /source/emacs
```

3 Unpack the source distribution

```
uncompress < /distrib/emacs/emacs-18.55.tar.Z |
tar xvBf -
```

4 Configure and build the targets

Follow the steps described in **/source/emacs/INSTALL**. At step 3) you are told to modify **src/paths.h** and **lisp/paths.el**. In the first, set the C macros as follows:

```
#define  PATH_LOADSEARCH   "/vol/emacs/lisp"
#define  PATH_EXEC         "/vol/emacs/etc.sun4"
#define  PATH_LOCK         "/vol/emacs/lock/"
#define  PATH_SUPERLOCK\
                  "/vol/emacs/lock/!!!SuperLock!!!"
```

This causes Emacs to look in the correct directories at run time.

Install it

Since you're on your own workstation and since **/vol/emacs** is read-only, you will have to login to the file server in order to install Emacs in the appropriate directories.

1 Login on **bach** (the file server for applications)

2 Copy the targets to the installation directory

```
cd /source/emacs
find info lisp man -depth -print |
         cpio -pdvm /argon/vol/emacs-18.55
cd src
cp ctags emacs emacsclient emacstool ctags\
         /argon/vol/emacs-18.55/bin.sun4
cd ../etc
cp DOC TUTORIAL emacs.icon yow.lines\
         /argon/vol/emacs-18.55/etc
cp cvtmail env fakemail loadst movemail server yow\
         /argon/vol/emacs-18.55/etc.sun4
```

1 The very first time you do this, you have to reboot the systems that need these resources in order for the automounter to start successfully.

3 Create the appropriate links

In order for Emacs to find its own files, you have to set up some links to the shared information that it needs.

```
cd ../etc.sun4
ln ../etc/* .
```

Make the Emacs commands available

Still on **bach**, execute the following:

1 Switch to the architecture dependent bin directory

```
cd /argon/vol/sun4/bin
```

2 Create the symbolic links to the executables

```
ln -s /vol/emacs/bin.sun4/* .
```

Make the manual pages available

Emacs supports manual pages in section 1 and 6.

1 Create the appropriate symbolic links for section 1 and 6

```
cd /argon/vol/man/man1
ln -s /vol/emacs/man/man1/* .
cd ../man6
ln -s /vol/emacs/man/man6/* .
```

Use it!

The proof of the pudding is in the eating...

1 Login on **liszt** (your workstation)

2 Modify your search paths

```
PATH="/vol/bin:${PATH}"             export PATH
MANPATH="/vol/man:/usr/share/man"   export MANPATH
```

3 Try it out

```
man emacs
emacs
```

That's it. Any questions?

Summary

To summarize, the following table shows you the essence of this document.

Location	Path
logical	**/***class***/***instance*
physical	**/***partition***/***class***/***instance*

"For instances"

For applications you need something extra:

Applications	Path
logical	**/vol/***applic***-***version* and **/vol/***applic* for the default version
physical	**/***partition***/vol/***applic***-***version*

Commands and architecture dependencies are handled as follows:

Command	Path
logical	**/vol/bin/***command*
physical	**/***partition***/vol/***arch***/bin/***command*

And the corresponding maps for the automounter look like:

Map name	Contents		
auto.master	**/***class*	**auto.***class*	[*options*]
	/-	**auto.direct**	[*options*]
	/net	**-hosts**	[*options*]
auto.*class*	*instance*	[*options*]	*server***:/***partition***/***class***/***instance*

And to resolve architecture dependencies, versions and variants use:

Map name	Contents	Location
auto.vol	**bin**	*server***:/***partition***/vol/${ARCH}/bin**
	lib	*server***:/***partition***/vol/${ARCH}/lib**
	etc	*server***:/***partition***/vol/${ARCH}/etc**
	app	*server***:/***partition***/vol/***application***-***version*
	*app***-***vers*	*server***:/***partition***/vol/***application***-***version*
auto.direct	**/usr/local**	*server***:/***partition***/vol/${ARCH}**

Easy, consistent and efficient.

Conclusion

With the concepts laid out in this document, plug and play is the way to go. Especially with SunOS 4.1.1 Revision B, life becomes easy. The only thing users have to do, is to get the Ethernet address of their new hot box—which is available from the customer information sheet in the plastic bag attached to the system unit carton—and give this to the system administrator. The administrator then uses this information to set up the NIS hosts and ethers maps accordingly. That's all. The user can then turn on the power switch and play! All the relevant files are automagically available from the first moment the system is switched on.

How's that for a change?