AT&T

UNIX System Laboratories, Inc.
A Subsidiary of AT&T

UNIX® SYSTEM V RELEASE 4

Network User's and Administrator's Guide

AT&T

UNIX System Laboratories, Inc.

A Subsidiary of AT&T

UNIX® SYSTEM V RELEASE 4

Network User's and Administrator's Guide

Published by Prentice-Hall, Inc.
A Division of Simon & Schuster
Englewood Cliffs, New Jersey 07632

10 9 8 7 6 5 4 3 2

ISBN 0-13-933813-6

UNIX
PRESS
A Prentice Hall Title

Contents

Preface

TCP/IP Network Administrator's Guide

Distributed File System Administration Guide

Preface to This Volume

About This Volume

The *Network User's and Administrator's Guide* is directed to system administrators who are setting up and maintaining UNIX® System V file sharing capabilities, and to users of remote services provided by System V network software.

File sharing refers to the process of making resources on your local system available to remote systems via a network, and conversely, to accessing resources on remote systems from your local system. System V Release 4.0 offers two file sharing packages, also called "distributed file systems." These are Remote File Sharing (RFS) and Network File System (NFS).

This manual tells you how to set up and administer both RFS and NFS, and how to set up and use Distributed File System Administration (DFS), a software package that provides a common interface to both RFS and NFS.

Also included in this volume is documentation that tells you how to set up TCP/IP, a family of network protocols that determines how data is transferred across network media. TCP/IP supports both RFS and NFS, and is provided in System V Release 4.0 as the TCP/IP Internet package. It is not necessary that you install TCP/IP to run RFS. RFS can run over any network protocol that conforms to the Transport Level Interface (TLI), such as the AT&T STARLAN NETWORK. NFS, however, requires UDP/IP as its transport. UDP/IP are protocols at the transport layer in the TCP/IP protocol family.

Included in TCP/IP at the application layer are a number of commands and programs that allow users to perform remote operations. These, too, are documented in this manual.

NOTE This manual is not intended to be an introduction to networking, nor to all the networking features of UNIX System V Release 4.0. For an introduction to basic networking concepts, see the list of recommended reading at the end of this chapter. For an overview of System V Release 4.0 networking features, see the "Product Overview" in the *Product Overview and Master Index.*

Because you may be setting up network services using a mix and match of applications and protocols, this volume is organized as a collection of stand-alone documents. For example, you may choose to set up file sharing using RFS over a transport provider that is not included in System V (such as the STARLAN NETWORK). In this scenario, only the RFS documentation is of interest to you. Because the *Remote File Sharing Administrator's Guide* is designed as a stand-alone document, you can remove it from the binder and shelve the rest of the documentation.

The stand-alone documents included in this volume are:

- *TCP/IP Network Administrator's Guide*

  Although TCP/IP can be used on a local area network to provide remote services and to support file sharing applications, it was originally developed by the Department of Defense to support the ARPANET, a packet switching wide area network. Today, the ARPANET is part of a wider public network, called the Internet. In addition to telling you how to set up and maintain TCP/IP software on your network machines, the *TCP/IP Network Administrator's Guide* tells you how to join the Internet.

- *Distributed File System Administration Guide*

  The *Distributed File System Administration Guide* describes Distributed File System Administration (DFS), a command interface common to both RFS and NFS. Because both RFS and NFS are provided in System V Release 4.0, one set of commands is provided with which an administrator can administer both packages. For example, the DFS software provides you with the share command, which allows you to share a resource on your system using either RFS or NFS. The *Distributed File System Administration Guide* is directed to administrators who are running both RFS and NFS on their systems. The commands described in the DFS guide are described in the RFS and NFS guides as well, but the RFS and NFS guides describe package-specific options only.

- *Remote File Sharing Administrator's Guide*

  The *Remote File Sharing Administrator's Guide* tells you how to set up and maintain RFS on your system, including how to share resources with remote systems and how to mount remote resources on your machine.

- *Network File System Administrator's Guide*

  The *Network File System Administrator's Guide* tells you how to set up and maintain NFS on your system, including how to share and mount resources, how to mount resources automatically using a feature called the automounter, and how to set up Secure NFS.

- *Remote Services User's Guide*

  The *Remote Services User's Guide* describes the user level services (sometimes known as the DARPA services) provided by TCP/IP. These services include commands that allow users to log in to remote machines, to transfer files to and from remote machines, and to execute commands on remote machines without logging in.

- *Reference Manual*

  The *Reference Manual* includes all the UNIX System V manual pages that are related to the software documented in this volume.

  Also, for convenience, manual pages related to the administration of Remote Procedure Call (RPC) are duplicated here.

NOTE For complete documentation about the administration of RPC, see the RPC manual in the *Programmer's Guide: Networking Interfaces*.

System V File Sharing

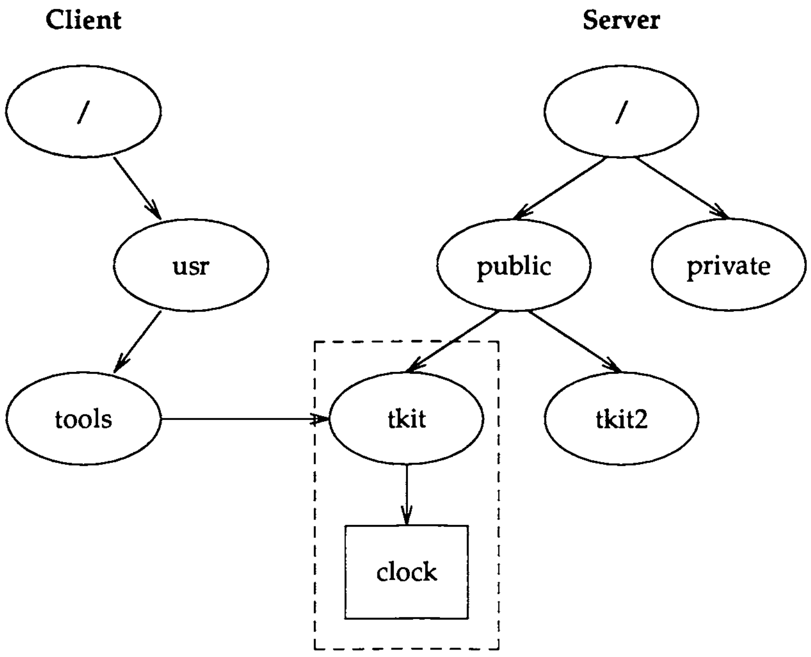
File sharing employs a *client/server* model. A computer that wishes to share file systems with other computers on a network acts as a *server*. Files are physically owned and managed by the server machine. A computer that wishes to access file systems that do not reside on its physical disk acts as a *client* of the server machine. Acting on behalf of its applications, the client makes requests to a server to access data in a file or to perform file manipulations. A single machine may be both a client and a server, if it wishes both to share its local file systems and to access remote file systems.

A server may offer any directory tree for access over the network. Once shared, an authorized client may mount the remote file system on any of its local directories. The mount procedure behaves in a manner similar to mounting local file systems.

Transparency is the key to the usefulness of file sharing. Once mounted, remote file systems look like local file systems from a user or application perspective. Applications, in most cases, run unchanged.

Since remote file systems may be mounted anywhere in the local tree, existing programs can run on several different computers while still having the same files and directory structure available to them. Creating the file environment is now mainly an administrative task, not one requiring program changes.

Servers do not need to make all their files accessible to network clients. In the following illustration, the server is sharing `/public/tkit`. The client mounts `/public/tkit` on its local directory `/usr/tools`. The remote directory tree now appears to be a directory tree under `/usr/tools`, and files in that tree may be accessed as though they were local. Note that the client cannot access `/public/tkit2`.



In a file sharing environment, a large number of users can access a program as though it were on their local machines, when actually the program resides an a single file server. This is a great benefit to small workstations, where disk space is at a premium. A user can have access to a much larger program repertoire than could fit on a private disk.

By having a resource reside physically on a single server, then distributed throughout the network, you can greatly simplify administration. First, you reduce the number of copies of various programs that need to be maintained on the network. Second, you reduce the problems involved in performing backups for a number of machines dispersed over a wide geographical area. By keeping files in a single location, this task becomes comparable to backing up a single machine.

Centralizing files on a few file servers not only simplifies administration, it helps maintain consistency of shared data files. When changes are made to a shared file, they become available to all users immediately.

As an alternative to centralizing files on a few file servers, files may be shared in a peer manner. When a single computer runs out of capacity, more computers can be added to a configuration. Files can be moved to the new computers, while a consistent view of the file system from the user's perspective is maintained.

RFS vs. NFS

In UNIX System V Release 4.0, RFS and NFS have been integrated as file system types under the Virtual File System mechanism. This means that the benefits of both RFS and NFS can be realized on the same computer. RFS and NFS will operate on the same machine and over the same network. Since the same directory trees may be mounted simultaneously over RFS and NFS, applications that require one or the other can co-exist. However, RFS and NFS protocols are different, and do not inter-operate.

NOTE Because RFS and NFS are implemented as file system types under the Virtual File System (VFS) mechanism, applications see a normal file system interface. The same open, read, write, and close operations as used for local files work for remote files. The distributed file system uses the VFS interface to take care of translations between the UNIX file system interface and internal protocols. For more information about Virtual File System, see the *System Administrator's Guide.*

In Release 4.0, the syntax of administrative commands for RFS and NFS has been standardized to provide a uniform interface to distributed file systems. File system-dependent options allow differences to be accommodated, while integrating common features. Older forms of commands are still available to provide compatibility with previous systems.

Both RFS and NFS provide a comparable file sharing capability but have some differences, resulting from their different goals.

The goal of NFS is to provide file sharing in a heterogeneous environment potentially containing many different operating systems. The NFS internal protocol has been designed to be implementable on non-UNIX operating systems in order to provide an open file sharing capability. NFS clients and servers have been implemented on many operating systems ranging from MS-DOS to VMS. A secondary goal of NFS is to provide good recovery characteristics when file servers fail; this has resulted in a design where the NFS servers do not keep any state about clients accessing them. Thus, servers are not sensitive to client crashes, which results in a robust environment.

The major goal of RFS is to provide file sharing in a UNIX System V environment as transparently as possible. A file system shared using RFS closely supports UNIX file system semantics. This makes it relatively easy to migrate programs written to use the local file system.

The following sections highlight some of the similarities and differences between RFS and NFS.

- **Shared Objects**

  An RFS server can share any local directory, and all local files and directories under the shared directory are made accessible to remote users. RFS clients can access all file types transparently, including ordinary files, directories, named pipes, and special devices. Because computers running RFS can access remote devices, expensive peripherals, such as tape drives, can be used for backups by smaller computers that could not justify the cost of a dedicated drive; media incompatibilities (such as diskette formats) can be overcome by using devices supporting required formats on remote computers; and the reliability of the environment can be increased by making remote devices such as printers available if the locally attached device breaks down.

  An NFS server can share a file system or any part of file system, including a single file. All files and directories below the root of the shared resource are made available to clients. A machine cannot share a file hierarchy that overlaps with one that is already shared. For example, it is illegal to share both `/public` and `/public/tkit`.

- **Transparency and Consistency**

  The same system calls and command syntax used to access local files are used to access RFS and NFS resources. Remote data is cached on both RFS and NFS clients. Cache consistency is guaranteed on RFS clients, but not on NFS clients.

- **Location Independence**

  RFS provides a name server with which to register resource names; the administrator of the client does not need to know where a resource resides. If resources (such as on-line manual pages or line printers) are moved, the client does not need to know about the move or modify its actions to access the resources at their new locations.

  The administrator of an NFS client machine must specify the name of the server machine when mounting an NFS file system. If a resource moves, all clients must be aware of the new location when mounting the resource.

- **Heterogeneity**

  RFS can operate over a network of computers running UNIX System V Release 3.0 and later System V releases, regardless of the underlying computer architecture.

  NFS can operate on a variety of operating systems implemented on a variety of computer architectures.

- **Network Protocol Independence**

  RFS can run over any network protocol that conforms to the System V Transport Provider Interface and provides virtual circuit service. In Release 4.0, if multiple network protocols exist on the machine, RFS may run over all of them simultaneously.

  NFS is built on top of the Remote Procedure Call (RPC) facility, which requires the User Datagram Protocol (UDP) transport. UDP is a protocol in the TCP/IP protocol family.

- **Full Semantics**

  RFS supports full UNIX system semantics, including file and record locking, open with append mode, access to remote devices and pipes, and so on.

  NFS takes advantage of a network locking facility called the Lock Manager. The Lock Manager supports the UNIX System V style of advisory and mandatory file and record locking.

- **Application Compatibility**

  Since full file system semantics are supported, RFS allows all applications to run without recompilation.

  Because NFS provides transparent access, within limits, NFS allows existing applications that do not attempt to use unsupported features to run without recompilation.

- **Security**

  RFS allows file access based on machine passwords and user and group IDs. It also provides an administrator with the ability to specify read-only access to resources; to share directories selectively; to restrict which machines can access resources; to unshare a resource to prevent further client mounts; and to force a client to unmount a resource.

  NFS assumes global UID/GID space, and provides an administrator with the ability to restrict which machines can access resources; to specify read-only access to shared directories; and to unshare a directory, causing client access to that directory to fail. NFS also provides an option called Secure NFS, which supports encrypted machine and user identification along with ID mapping.

- **State**

  RFS servers maintain the state of local resources. The server knows which clients hold references to each of its files at any given time.

  NFS servers do not maintain the state of local resources. The server does not know which client has its files open at any given time.

- **Recovery**

  With RFS, if a client crashes, the server removes the client's locks and performs other clean-up actions. If the server crashes, the client acts as if that portion of the file tree has been removed (reads fail, and so on).

  With NFS, if the client crashes, the server is oblivious to it. If the server crashes, clients can either block until the server comes up or return an error after a timeout.

Contents

Figures and Tables

1. INTRODUCTION TO TCP/IP

1. INTRODUCTION TO TCP/IP

1 Introduction to Administering TCP/IP Networks

About This Guide

This document explains how to set up and administer a network built on the TCP/IP Internet protocol family, as provided in UNIX System V Release 4.0. The administrative steps needed are simple and few—once you understand the concepts underlying them. Therefore, this guide concentrates on the concepts that will allow you to install the most appropriate network for your organization.

Audience

This guide is directed to system administrators who are experienced administrators of stand-alone systems. In addition, this guide assumes the reader has a basic understanding of computer networking and data communications.

Organization

This guide is organized as follows:

- "Introducing the Internet Protocol Family," which introduces basic concepts relating to TCP/IP and describes the tasks which an administrator must perform when dealing with a TCP/IP network.
- "Setting Up Network Software," in which various aspects of the network software are discussed. Also covered are the procedures you follow to obtain a network number, register a domain, set up administration files, and boot TCP/IP on network hosts.
- "Maintaining Security in a TCP/IP Environment," which explains the effects of the `hosts.equiv`, and `.rhosts` files, as well as the implications that these files have on security issues.
- "Expanding Your Network," which describes the hardware you need to expand your network, and presents procedures for creating an internetwork, setting up a router, and setting up subnets.
- "Using the Domain Name Service," which tells you how to take advantage of Domain Name Service, a name service at the application layer of TCP/IP.

- "Troubleshooting," in which various commands that help you diagnose problems on the network are discussed.
- "Glossary," which covers most of the new terms used in this guide.
- Appendix A, "Guidelines for Completing the IP Number Registration Form," which contains information you need to register your network with SRI-NIC.
- Appendix B, "Guidelines for Completing the Domain Registration Form," which contains information you need to register your domain with the NIC Domain Registrar.

Introducing the Internet Protocol Suite

A network is a configuration of machines that exchange information among themselves. In order for the network to function properly, the information originating at a sender must be transmitted along a communication line and delivered to the intended recipient in an intelligible form. Because different types of networking software and hardware need to interact to perform this function, network designers developed the concept of the communications protocol family (or suite). A *network protocol* is a set of formal rules explaining how software and hardware should interact within a network in order to transmit information. The Internet Protocol family is one such group of network protocols. It is centered around the Internet Protocol (IP). The other members of the Internet protocol family are Transmission Control Protocol (TCP), User Datagram Protocol (UDP), Address Resolution Protocol (ARP), Reverse Address Resolution Protocol (RARP), and Internet Control Message Protocol (ICMP).

The entire family is popularly referred to as TCP/IP, reflecting the names of the two main protocols. This is also the terminology used in this document.

TCP/IP provides service to many different types of host machines connected to heterogeneous networks. These networks may be wide area networks, such as X.25-based networks, but they may also can be local area networks, such as one you might install in a single building.

TCP/IP was originally developed by the United States Department of Defense to run on the ARPANET, a packet-switching wide area network first demonstrated in 1972. Today the ARPANET is part of a wide area network known as the DoD (Department of Defense) Internet, or, for short, the Internet. Many popular texts use the term *Internet* to describe both the protocol family and the wide area network. This text uses the term *TCP/IP* to refer to the Internet protocol suite and *Internet* when referring to the network itself.

The TCP/IP protocol structure can be conceptualized as being formed of a series of layers as shown in Table 1-1.

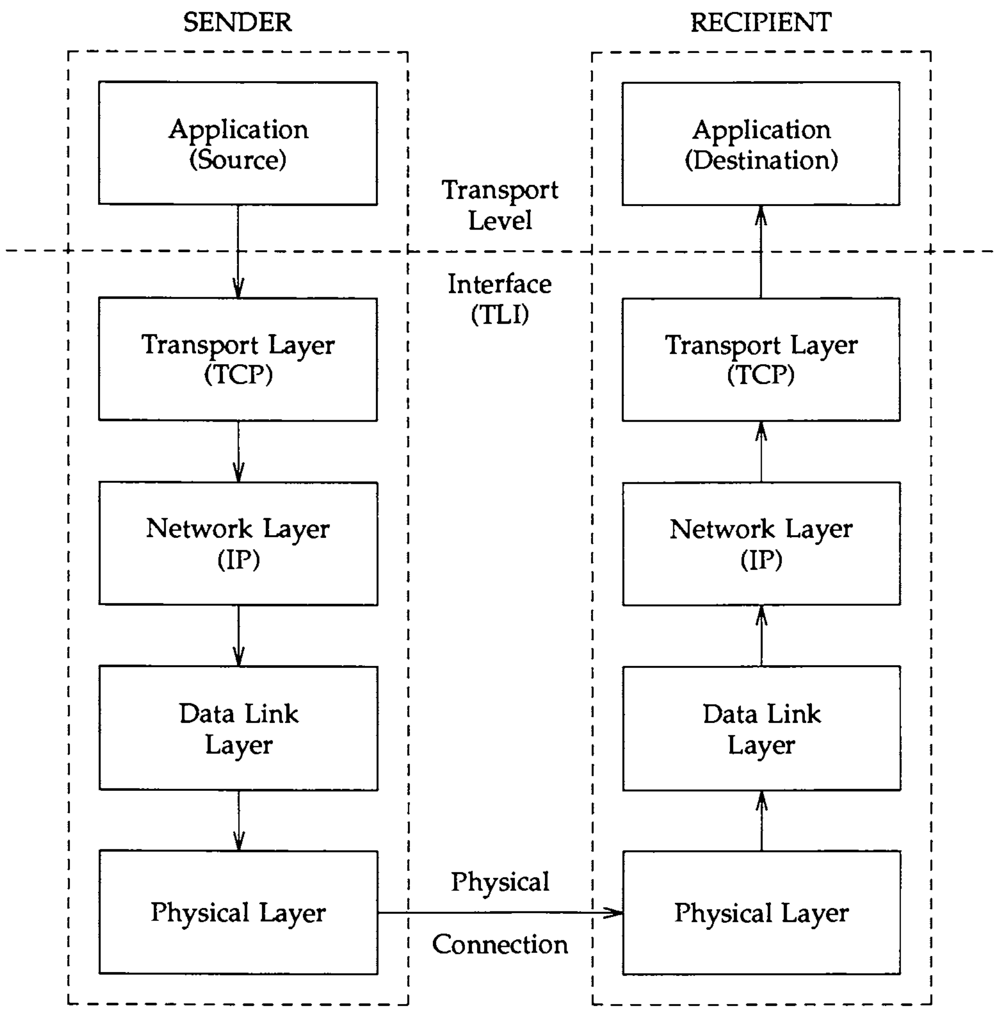
Table 1-1: TCP/IP Protocol Layers

Layer	*Network Services*
Application	Telnet, FTP, TFTP
Transport	TCP, UDP
Network	IP, ICMP
Data Link	ARP, RARP, device driver (such as Ethernet™)
Physical	Cable or other device (such as an Ethernet board)

In TCP/IP jargon, a machine engaged in communication is termed either a sending or receiving host. Every protocol layer on the sending host has its peer protocol layer on the receiving host. Each layer is required by design to handle communications in a pre-determined fashion.

Each protocol formats communicated data and appends or removes information from it. Then the protocol passes the data to a lower layer on the sending host or a higher layer on the receiving host, as illustrated in the following figure:

Figure 1-1: Sender/Receiver Interaction



In the UNIX System V Release 4.0 implementation, the interface between the application and the transport layers is Transport Level Interface (TLI), an interface that eliminates the need for applications to know particulars about the transport layer. Any application written to TLI can run on a TCP/IP network. For more information about TLI, see the *Network Programmer's Guide and Reference Manual*.

The next subsections briefly explain how each protocol layer handles messages. For more detailed information, refer to the manual page for the appropriate protocol.

Physical Layer

The physical layer is the hardware level of the protocol model, which is concerned with electronic signals. Physical layer protocols send and receive data in the form of *packets*. A packet contains a source address, the transmission itself, and a destination address.

TCP/IP supports a number of physical layer protocols, including Ethernet™ and Token Ring™. Ethernet is an example of a *packet switching network*; its communications channels are occupied only for the duration of the transmission of a packet. The telephone network is an example of a *circuit switching network*.

Data Link Layer

The data link layer is concerned with addressing at the physical, machine level. Protocols at this layer are involved with communications controllers, their chips, and their buffers. Ethernet is supported at this layer by TCP/IP.

Two additional TCP/IP protocols, ARP and RARP, can be viewed as existing between the network and data link layers. ARP is the Ethernet address resolution protocol. It maps known IP addresses (32 bits long) to Ethernet addresses (48 bits long).

RARP (or Reverse ARP) is the IP address resolution protocol. It maps known Ethernet addresses (48 bits) to IP addresses (32 bits), the reverse of ARP.

Network Layer

IP (Internet Protocol) and ICMP (Internet Control Message Protocol) are the protocols present at the network layer.

IP provides machine-to-machine communication. It performs transmission routing by determining the path a transmission must take, based on the receiving machine's IP address. (A later section of this guide called "IP Addressing" discusses IP addresses in detail.) IP also provides transmission formatting services; it assembles data for transmission into an *internet datagram*. If the datagram is outgoing (received from the higher layer protocols), IP attaches an *IP header* to it. This header contains a number of parameters, including the IP addresses of the sending and receiving hosts. Refer to the `IP`(7) manual page for more information.

ICMP sends error or control messages to other hosts. It provides communication of Internet software between machines. Refer to the `ICMP`(7) manual page for information about ICMP.

Transport Layer

The TCP/IP transport layer protocols enable communications between processes running on separate machines. Protocols at this level are TCP and UDP.

TCP (Transmission Control Protocol) enables applications to talk to each via virtual circuits, as though they had a physical circuit between them. TCP is a connection-oriented, reliable protocol; any data written to a TCP connection will be received by its peer in sequence, or an error indication will be returned.

UDP (User Datagram Protocol) is the alternative protocol available at the Transport layer. UDP is a connectionless datagram protocol. Datagrams are groups of information transmitted as a unit to and from the upper layer protocols on sending and receiving hosts. UDP datagrams use port numbers to specify sending and receiving processes. However, no attempt is made to recover from failure or loss; packets may be lost with no error indication returned.

Whether TCP or UDP is used depends on the network application invoked by the user. For example, if the user invokes `telnet`, that application passes the user's request to TCP. If the user's request involves the Domain Name Service, that application passes the request to UDP.

Refer to the TCP(7) and UDP(7) manual pages for more information about these protocols.

Application Layer

A variety of TCP/IP protocols exist at the application layer. Here is a description of some of the more widely used:

- `telnet`

  The Telnet protocol enables terminals and terminal-oriented processes to communicate on a network running TCP/IP. It is implemented as the program `telnet` on the local machine and the daemon `telnetd` on the remote machine. Telnet provides a user interface through which two hosts can open communications with each other, then send information on a character-by-character or line-by-line basis. The application includes a series of commands, which are fully documented in the `telnet`(1) manual page.

  The `telnetd` daemon on the remote host handles requests from the `telnet` command. For more information about `telnetd`, see the `telnetd`(1M) manual page.

- `ftp`

  The File Transfer Protocol (FTP) transfers files to and from a remote network. The protocol includes the `ftp` command on the local machine and `ftpd` daemon on the remote machine. `ftp` lets you specify on the command line the host with whom you want to initiate file transfer and options for transferring the file. The `ftpd` daemon on the remote host handles the requests from your `ftp` command.

  The `ftp`(1) manual page describes all options to `ftp`, as well as the commands you invoke through the `ftp` command interpreter.

  The `ftpd`(1M) manual page describes the services provided by this daemon.

- `tftp`

  The Trivial File Transfer Protocol (TFTP) enables users to transfer files to and from a remote machine. Like `ftp`, `tftp` is implemented as a program on the local machine and as a daemon (`tftpd`) on the remote machine. `tftp` invokes a command interpreter for transferring files and maintains a connection between two machines between file transfers.

  The `tftp`(1) manual page describes the commands you can give to the `tftp` command interpreter. The daemon is described on the `tftpd`(1M) manual page.

- Domain Name Service

  The Domain Name Service (DNS) is a protocol that provides domain-name-to-address-mapping of forwarding hosts and mail recipients on a network. The manual page `named`(1M) describes the service.

  A later chapter in this manual describes how to implement the Domain Name Service on your network.

Other application layer protocols exist that are also implemented as a program on the local machine and a daemon on the remote one; examples of these are `rlogin` and `rlogind`, which permit a user to log in to a remote machine; `rsh` and `rshd`, which enable the user to spawn a shell on a remote machine, and `finger` and `fingerd`, which permit a user to obtain information about users on remote machines.

To avoid the need to have an excess of daemons running at all times, the daemon `inetd` is initiated at startup time. After consulting the `/etc/inetd.conf` file, `inetd` runs the appropriate daemons as needed. For example, the daemon `rlogind` will be run by `inetd` whenever there is a request for a remote login from another machine, and only at that time and for the duration of the remote login.

An Overview of TCP/IP Administration

The administration of a TCP/IP based network consists of:

- Obtaining an Internet address for a new network.

  In order to connect a network to the Internet it is necessary to obtain a network number from SRI-NIC. Appendix A tells you how to obtain the request form from NIC and complete it.

- Registering the domain name for a new network.

  To register a domain with SRI-NIC, you request a registration form from SRI-NIC and return it with information about your domain. All new domains that connect to the Internet must be registered. Appendix B tells you how to obtain and complete the registration form.

- Establishing the network and (optionally) the sub-networking configurations.

  You need to determine the administrative arrangement of your network; administrative subdivisions may be desirable. See the chapter "Expanding Your Network."

- Maintaining administration files for each host and service.

  Each network has a set of files that lists the hosts and services using that network. These files should be maintained and kept current with each other. The information for doing these tasks is in "Setting Up TCP/IP."

- Maintaining the security of the network.

  The security of the network can be compromised if you do not take proper steps, such as creating and maintaining proper `hosts.equiv` and `.rhosts` files. See the chapter "Security in a TCP/IP Environment."

- Diagnosing and debugging network problems.

  A number of commands are provided that help you to determine the causes of trouble on the network. See "Troubleshooting."

2 Setting Up TCP/IP

Introduction

This chapter explains how to set up a machine to join a TCP/IP network. To set up your software, you must supply information, such as network addresses, to a number of programs and files. Setup tasks include

- Obtaining an Internet network number
- Assigning IP addresses to network hosts
- Selecting a domain name
- Installing network hardware
- Installing the TCP/IP Internet package and additional software on which TCP/IP depends
- Editing the TCP/IP startup script `/etc/inet/rc.inet`
- Setting up administrative files `/etc/hosts` and `/etc/networks`
- Setting up the Listener, a network port monitor
- Setting up TCP/IP as the machine's preferred network (optional)
- Re-booting the system.

If you intend to set up a machine as a router, additional procedures are involved. Once you complete the procedures described in this chapter, and before you re-boot, see the chapter "Expanding Your Network" later in this guide for instructions.

Obtaining an Internet Network Number

Every TCP/IP network should have an Internet network number, even if the network does not join the actual Internet. NIC at SRI International is the organization that registers TCP/IP networks and assigns network numbers.

To obtain an Internet network number, request the proper form from NIC. Instructions for requesting and completing the form appear in Appendix A of this guide.

When you request a network number, you must tell NIC the classification you want for your network. The available network classifications are described below.

- Class A Network

  There are very few Class A networks, but each accommodates a large number of hosts. Typically, Class A networks belong to large organizations or universities. Class A networks are assigned numbers from 0-127.

- Class B Network

  Class B networks provide a median distribution between networks and hosts. For example, the NIC assigns Class B addresses to medium-sized companies with many hosts on their networks. Class B networks are assigned numbers from 129-191.

- Class C Network

  Class C networks are small networks with a maximum of 255 hosts on each. Class C networks are assigned numbers from 192-223.

When you choose a classification, choose the smallest that will accommodate network growth over the next few years. (The chances are slim that your network will need, or that the NIC will grant, Class A classification.) For a fairly large organization that routes together local area networks in several buildings, consider requesting a Class B address. This is a good idea, particularly if you plan to subnet part of your local networks. If your network will not conceivably support more than 255 hosts, ask for a Class C address.

Assigning IP Addresses to Your Network Hosts

Sometimes packets travel only from one host to another on the same local network. At other times, they travel from one local network to another network, going through a router. In the extreme, a packet may traverse many networks, crossing miles of routers, gateways, cabling, and telephone lines to reach its destination. A TCP/IP network routes a packet according to the destination *IP address*—an address provided by the IP protocol on the sending host.

To run TCP/IP properly, every host on the network must have an IP address. A machine's IP address consists of the following:

- the net number
- a subnet number (optional)
- a host number

The net number is the Internet network number you obtain from NIC. The host number is a number that uniquely identifies a machine on your network. You assign the host number.

A subnet number is an optional number representing the subnet to which the host is attached. This is a number that you (as opposed to the NIC) assign to the subnet when you set it up. A machine's IP address includes a subnet number only if the machine is on a subnet. (Refer to "Setting Up Subnets" in "Expanding Your Network" for information about subnet numbering.)

An IP address is 32 bits, divided into four 8-bit fields (octets) separated by periods. Each octet can have a value of 0 to 255. Here is a typical IP address:

    129.144.50.56

NOTE All IP numbers in this document are provided as examples only. Do not use them for your own purposes.

The following table illustrates how addresses are structured:

Figure 2-1: Network Address Structure

	Range	Network Address	Host Address
Class A	0-127	xxx	xxx.xxx.xxx
Class B	128-191	xxx.xxx	xxx.xxx
Class C	192-223	xxx.xxx.xxx	xxx

Once you have received your network number from the NIC, you can create IP addresses for network hosts. You may want to do this on paper, or in an ASCII file for reference, before supplying these addresses to the programs and files that use them.

The first part of the host's IP address must consist of the network number that the NIC assigned you.

You assign the host part of the IP address in the remaining octets after the network number. In the unlikely event that your hosts are on a Class A network, you have three octets for defining the host address. On a Class B network, you have two octets; on a Class C network, one.

You may assign values from 0-255 in each octet allowed for the host address, but the total host address cannot be either 0 or 255. (The addresses 0 and 255 are reserved for broadcasting.) If, for example, you have a Class A network where the net address is 10, a host could have the address 10.30.0.107 or 10.1.1.255, but it could not have the address 10.0.0.0 or 10.255.255.255; in the same vein, if your network is Class C, with address 192.9.90, you should not assign addresses 192.9.90.0 or 192.9.90.255 to any of its hosts.

Suppose you want to install a machine named `dancer` on your Class C network. If the Class C network has the network number 192.9.200, you might assign `dancer` the IP address

    192.9.200.1

Then you might install other hosts on the same network with IP addresses such as 192.9.200.2, 192.9.200.3, and so on.

Establishing a Domain

Once you obtain a network number, assign IP addresses to all the hosts on your network, and install the software, you can establish a domain. A domain is a set of machines that are administered and maintained as a single entity. Generally, all the machines on a local network comprise a domain on the larger network; however, you may choose to break the local network into several administrative entities, called subdomains. For example, you may want all the machines in the Accounting department to comprise one subdomain, and all the machines in Marketing to comprise another.

To join the Internet, your domain must be named according to Internet naming conventions, and you must register your domain name with the NIC. Even if you do not intend to join the Internet at this time, it is recommended that you follow the Internet naming conventions and register with the NIC to avoid problems in the future, should you decide to join the network.

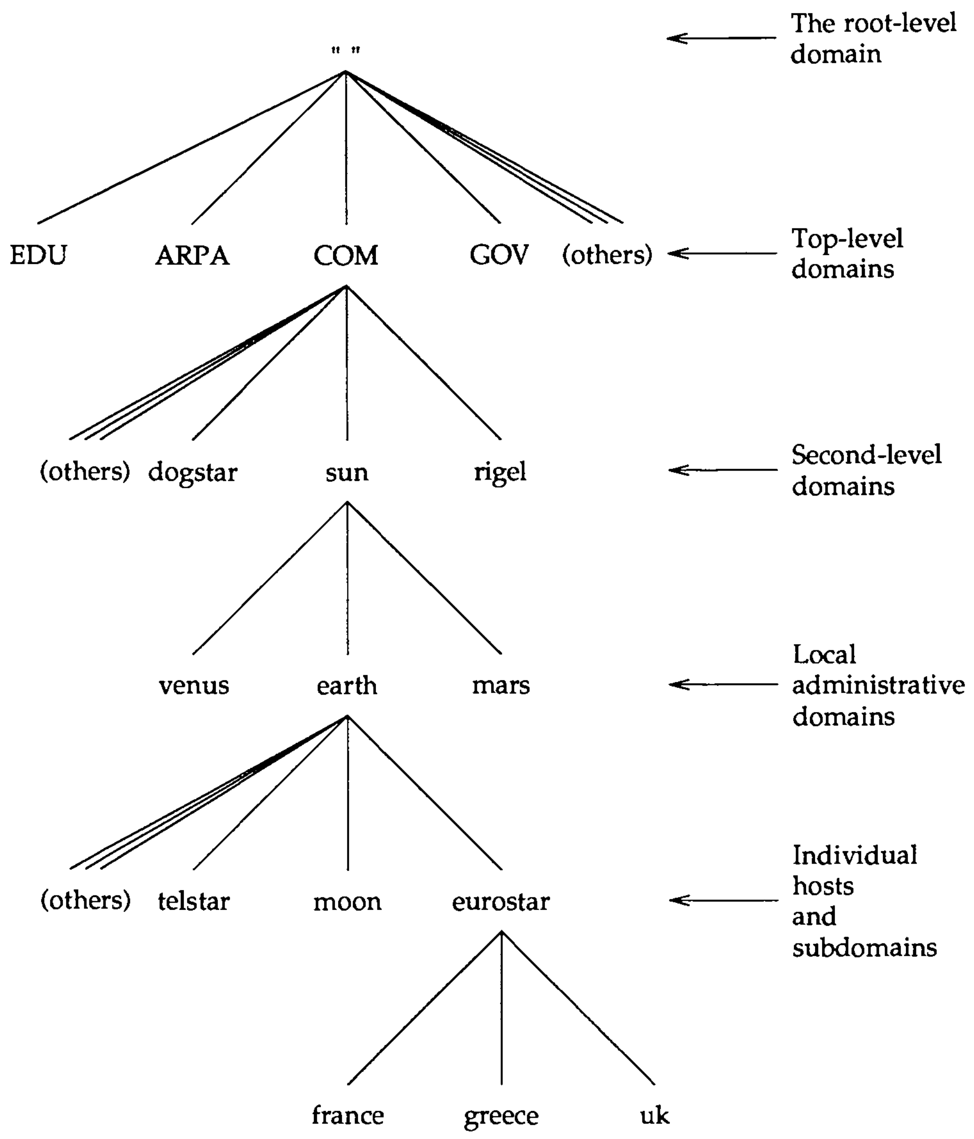
This section explains the Internet naming conventions, and directs you to Appendix B for instructions for obtaining and completing the the Internet domain registration form. If you intend to join a public network other than the Internet, such as BITNET or CSNET, contact the organization that administers the network for information about naming conventions and for a registration form.

The Organization of the Internet

The Internet consists of four levels of domain: the root level, the top level, the second level, and the local level. Each level branches from the root level. Below the local level are hosts and, optionally, subdomains.

The following illustration shows the different domain levels of the Internet and how they relate to each other.

Figure 2-2: A Typical Internet Domain Hierarchy



At each level of the Internet are name servers. A name server is a machine that maintains information about machines at the next lower level and facilitates name-to-address mapping.

The different levels of the Internet are described below:

- The Root-Level Domain

  The root level is the top of the entire Internet; it is maintained by the NIC. (Organizations in charge of other public networks, such as BITNET and CSNET, also administer the roots of their networks.)

  At the root level, the NIC administers *root domain name servers,* which maintain information about name servers at the next lower level.

- Top-Level Domains

  The root level branches into "top-level domains." Currently, some Internet top-level domains are EDU, ARPA, COM, GOV, MIL, NET, ORG, and US.

  The names of the top-level domains are assigned by the NIC. They reflect the types of networks that make up the domain. For example, EDU (an abbreviation of *Education*) is a top-level domain that consists of networks administered by universities; COM (an abbreviation of *Commercial*) consists of networks belonging to business organizations.

  When you register with the NIC, it assigns your network to one of these domains, depending on your organization's function.

- Second-Level Domains

  Each top-level domain branches into second-level domains. Member organizations at the second level assign "domain administrators" to manage their name servers, and the NIC assigns a "technical contact" to coordinate administration across domains. Depending upon your site's size and requirements, you may have domain administrator responsibilities.

- Local Administrative Domains

  Within each second-level domain are local administrative domains. These are the domains you administer for your organization. A local domain can be as small as one host, or large enough to include many hosts and additional name servers. It may also have other administrative domains (called subdomains) nested within it.

For more information about name servers and administrative responsibilities, see "Using the Domain Name Service" later in this guide.

Selecting a Domain Name

The Internet is organized as a hierarchy of domains. The name space is organized as a tree according to organizational or administrative boundaries. Each node of the tree is a domain, which is given a label. The name of the domain is the concatenation of all the labels of the domain from the root to the current domain, listed from right to left and separated by dots. A host address for a host named monet at Podunk University might look like this:

    monet.podunk.EDU

The top-level domain for educational institutions is EDU, Podunk is a subdomain of EDU, and monet is the name of the host.

When you name your domain, you must select a label that is unique within its domain. For example, you could use "podunk" as a label for a domain consisting of all machines at Podunk University, provided that another university in the domain EDU has not used this name first. The label you assign a host within your domain must be unique as well, but only within domain "podunk."

Once you choose a label, you must concatenate it with the labels of all the higher level domains to create your domain name.

Domains are not limited to any fixed number of levels. For example, the Amalgamated Widgit Company might be registered with the NIC as belonging to the domain "Widgit.COM"; it might also have a subdomain called "eng.Japan.Widgit.COM" for the engineering group within their Japan subsidiary, and another called "mktg.Japan.Widgit.COM" for marketing in Japan. However, machines at company headquarters might belong to the subdomain called "HQ.Widgit.COM."

NOTE Upper- and lower-case letters are not significant in domain names. The traditional UNIX convention is to use all lower-case; many other systems use upper-case.

At many sites, users name their individual hosts while the administrators name the servers. The administrator should ensure that there are no duplicate names within a domain by using the `nslookup` program. This is an interactive program that you can use (among other things) either to produce a listing of all the hosts in the domain (by using the `ls` command at the > prompt), or to obtain information about a host. At the prompt, you enter the host's name. If the host is in the database, `nslookup` produces its address; otherwise, it informs you that the host does not exist. (Refer to the manual page `nslookup`(1M) for a detailed explanation of the command.)

Registering Your Domain

After choosing the name of your upper domain (the equivalent of "Widgit.COM" in the example in the preceding section), you need to register it with a higher-level domain. To do this, you should get in touch with the administrator of the higher domain. The top-level domains are administered by the NIC at SRI International.

To register a domain with NIC, request the Domain Registration Form from NIC and complete it according to the instructions in Appendix B.

Once your domain is registered, you can divide it into subdomains as the need arises.

Installing Network Media

Before you install TCP/IP, you need to install your network media. At the present time, the implementation of TCP/IP in Release 4.0 runs over Ethernet.

For information about installing Ethernet on your system, see your Ethernet documentation.

Installing TCP/IP Software

Before you install the TCP/IP Internet package, you must install software on which TCP/IP depends. First, make sure the Network Support Utilities (NSU) package is installed on your system. NSU is provided as part of UNIX System V Release 4.0.

You also need software to drive your network media. Ethernet Media Driver (EMD) Utilities is provided as an optional utilities package with Release 4.0. For NSU and EMD installation instructions, see the Release 4.0 *Release Notes*.

NOTE When you install the Network Support Utilities, you must install pseudo-ttys. For information, see the *Release Notes*.

Once NSU and EMD Utilities are installed, you can install the TCP/IP Internet package. TCP/IP Internet installation instructions also appear in the Release 4.0 *Release Notes*.

When you install TCP/IP, the installation script prompts you for the IP address of your machine and your domain name. If you have not determined your IP address, refer to "Assigning IP Addresses to Your Network Hosts" earlier in this chapter. To determine a domain name, see "Establishing a Domain," also in this chapter.

Modifying the Startup Script

When you install TCP/IP on a host, a shell script is installed in `/etc/inet/rc.inet`. The script configures the system and start the daemons the machine needs to run on a TCP/IP network.

NOTE The names of all TCP/IP Internet daemons include an `in.` prefix; however, names have been abbreviated throughout this guide. For example, the daemon `in.routed` is referred to as `routed`. When you specify a daemon on the command line or in the startup script, use the full name of the daemon.

All daemons are located in `/usr/sbin`.

Before you run the script, you need to edit it, and you need to set up your `/etc/hosts` and `/etc/networks` files (as described later in this chapter). This section tells you how to edit `/etc/inet/rc.inet`.

To edit the script, follow these steps:

1. Locate the line

```
#/usr/sbin/route add default your_nearest_gateway hops_to_gateway
```

   Delete `default` and substitute the name of the router that will be the machine's default router. Delete the comment character (#).

   NOTE If you are unfamiliar with routers and you are unsure at this point if your local network needs a router, see "Expanding Your Network" later in this guide.

   Below the line that specifies the default router, you can specify additional routers that the machine can reach. Each router should be specified on a separate line, in the same form as the line that specifies the default router.

2. If your machine will be communicating with machines that are not connected directly to the local network, locate the line

```
#/usr/sbin/in.routed
```

and delete the comment character (#). (This line starts the route daemon, which facilitates communication with subnets via a router; if your machine operates only on a local network, the route daemon is unnecessary and may degrade performance if allowed to run.)

3. If you are using Domain Name Service (DNS), a name distribution facility available on some larger networks, locate the line

   ```
   #/usr/sbin/named
   ```

   and delete the comment character (#).

   For information about DNS, see "Using Domain Name Service" later in this guide.

Setting Up TCP/IP Files

There are a number of TCP/IP-related files that you, as a TCP/IP network administrator, need to set up and maintain.

`/etc/hosts`	The `hosts` file maps host names to IP addresses. The file is provided by the software, and must be edited and maintained by the system administrator. Every network host must have a `hosts` file.
`/etc/networks`	The `networks` file maps network names to net numbers.
`/etc/ethers`	The `ethers` file maps host names to their Ethernet addresses. You need to create an `ethers` file only if you are running the RARP daemon. device driver.
`/etc/protocols`	The `protocols` file lists the IP protocols installed on your system and their numbers; it is created automatically when TCP/IP is installed and requires no administrative handling.
`/etc/services`	The `services` file lists the names of reserved IP services and their port numbers; it is used by programs that call network services. `services` is created automatically when TCP/IP is installed and requires no administrative handling.
`/etc/netmasks`	The `netmasks` file associates network names with network masks (a is a number used by the system to separate the local subnet address from the rest of a give IP address). You need to maintain a `netmasks` file only if you are setting up subnets. For information about `netmasks`, see "Setting Up Subnets" in the chapter "Expanding Your Network," later in this guide.

Before you run the TCP/IP startup script on a network machine, you need to set up the machine's `hosts`, `networks`, and `ethers` files. The `hosts` and `networks` files on the machine should be consistent and up-to-date with the same files that reside on the other machines on your local network.

Setting Up the hosts File

When you set up your network initially, you must make sure that all the machines know the IP addresses of all other machines on the network. To do this, you specify the IP addresses of all the machines (including the address of the host itself) in the `hosts` file.

The `hosts` file lists all machines in the local domain by IP address and name, in the following format:

IP_number host_name nickname #comment

where *IP number* is the combination net number and host number that the administrator assigns a host; *host name* is the official name of the host machine; *nickname* is another name by which the host can receive information, such as mail or file services, over the network; and *#comment* is any kind of note you want to append to an entry.

The following line should always be in `/etc/hosts`:

```
127.0.0.1        localhost
```

The "localhost" address is 127.0.01 on every machine; the address is used by programs to reach services on the same machine from which they are invoked.

Whenever you add a host to the network, you have to add its name to the `hosts` file; this means that you have to modify the `/etc/hosts` file on each machine to reflect all changes.

Example

Suppose you have a machine named `dancer` with the IP address 192.9.200.1 . Next, assume you need to add three new hosts to your network—`ballet`, `raks`, and `samba`. To update the `hosts` file, you would add the following three (boldface) lines to `dancer`'s database:

```
192.9.200.1        dancer
192.9.200.2    ballet  # This is a comment
192.9.200.3    raks
192.9.200.4    samba
127.0.0.1 localhost
```

Next, edit `/etc/hosts` in each of the new machines, so that they are identical to the database on `dancer`.

At the end of the procedure, the four `/etc/hosts` files in `ballet`, `samba`, `raks`, and `dancer` should be identical (although the order of the entries may vary).

For more information about `/etc/hosts`, refer to the `hosts`(5) manual page.

Setting Up the networks File

The `networks` file contains the names of all TCP/IP-based networks to which your network connects, via routers. (For information about expanding your network with routers, see the chapter "Expanding Your Network" later in this guide.)

The `networks` file associates a network number with a name. It should contain the names and network numbers on every network that the machine can use.

Entries in the file have the following format:

network_name   network_number   nickname(s)   #comments

where *network_name* is the official name by which the network is known; *network_number* is the number assigned by the NIC; *nickname* is any other name by which the network is known; and *#comment* is any kind of note you want to append to an entry in the file.

Once you set up the `networks` file, you need to update it on the following occasions:

- Whenever you install another router, and you want to tell your server about the other network to which the router is attached.
- If you join a wide-area network, such as TELNET or BITNET, which is not listed in the file.

It is particularly important that you maintain the `networks` file, as the `netstat` program described later in the chapter on "Diagnosing Network Problems" uses the information in `/etc/networks` to produce status tables.

Example

Here is a sample /etc/networks file:

```
#
#
# Internet networks
#
arpanet         10              arpa
ucb-ether       46              ucbether
# local networks
loopback        127
eng             193.9.0         #engineering
acc             193.9.1         #accounting
prog            193.9.2         #programming
```

Setting Up the ethers File

The ethers file associates host's names with their Ethernet addresses. RARP uses the file to map Ethernet addresses to IP addresses. If you are running the RARP daemon, you need to set up the ethers file and maintain it on all hosts to reflect changes to the network.

The format of the ethers file is as follows:

Ethernet_address *host_name* *#comment*

where *Ethernet_address* is the address of the device driver on the host; *host_name* is the official name of the host; and *#comment* is any kind of note you want to append to an entry in the file.

The equipment manufacturer provides the Ethernet address. If a machine does not display the Ethernet address when you power up, see your hardware manuals for instructions on locating a host's address.

When adding entries to /etc/ethers, make sure that host names correspond to the primary names in /etc/hosts, not to the nicknames.

Example

Here is a sample /etc/ethers file such as you might find on your system. Entries, as shown in the example, should be in alphabetical order by machine name.

```
8:0:20:1:40:14  ballet
8:0:20:1:40:7   dancer    # This is a comment
8:0:20:1:40:15  raks
8:0:20:1:40:16  samba
```

Refer to the `ethers(4)` manual page for additional information.

Additional TCP/IP-Related Files

In addition to the files you need to create and maintain, TCP/IP installs two files, `protocols` and `services`, that are used by various programs. Although you do not need to handle these files, you may want to display them for information.

The protocols File

`/etc/protocols` contains the names of the TCP/IP protocols installed on the system. Below is an example `/etc/protocols` file:

```
#
# Internet (IP) protocols
#
ip      0       IP      # internet protocol, pseudo protocol number
icmp    1       ICMP    # internet control message protocol
tcp     6       TCP     # transmission control protocol
udp     17      UDP     # user datagram protocol
```

Refer to the `protocols(4)` manual page for more information.

The services File

/etc/services contains entries for reserved TCP/IP network services. Here is an excerpt from a typical /etc/services file:

```
#
# Network services
#
echo            7/udp
echo            7/tcp
discard         9/udp           sink null
discard         9/tcp           sink null
systat          11/tcp
daytime         13/udp
daytime         13/tcp
netstat         15/tcp
ftp-data        20/tcp
ftp             21/tcp
telnet          23/tcp
time            37/tcp          timserver
time            37/udp          timserver
name            42/udp          nameserver
whois           43/tcp          nicname         # usually to sri-nic
```

Notice that various network services like ftp and telnet are listed in the first column. The second column contains the service's port number and the transport layer protocol (either TCP or UDP) that handles it. Refer to the services(4) manual page for more information.

Some of the reserved services may not be available on your machines; however, their port numbers must be reserved nevertheless.

Setting Up the Listener

The listener is a port monitor that listens for communication across the network. Once the listener is properly configured for TCP/IP, your system can provide network services such as Network File System (NFS) and Remote File Sharing (RFS).

To set up the listener on a TCP/IP network, you need to provide it with an address. The address is a hexadecimal representation of a code that includes your IP address. The address has the following format:

\x*FmlyPortInternetReserved*

Fmly,	This is a four digit field that represents the family address (AF_INET). This field is always set to 0002.
Port	This is a four digit field that represents the port number of the listener. The UNIX System V Release 4.0 listener uses the port number 0ACE. The pre-UNIX System V Release 4.0 listener uses 0401. NOTE If a pre-UNIX System V Release 4.0 machine is running a listener, it can continue to be referenced by its address. (See the discussion of the use of heterogeneous address databases, later in this section.)
Internet	This is the host IP address in hexadecimal notation. This field is always 8 digits, with each octet of the host IP address represented by 2 hexadecimal digits.
Reserved	This is a reserved 16 hexadecimal digit field. It field is always set to 16 zeros (0000000000000000).

Before you can determine your listener address, you need to convert your IP address to hexadecimal notation. For instructions, see the following section.

Converting Your IP Address to Hexadecimal Notation

Assume your IP address is 192.9.200.1. To translate the IP address to hexadecimal notation, follow these steps:

1. Enter

   ```
   $ HEX() { for i in $*;
        do echo "$i in Hex: \t\c";
        echo 16o $i p | dc;
        done ;
        }
   ```

2. When the prompt reappears, enter your IP address. Using the sample IP address, you would enter

   ```
   HEX 192 9 200 1
   ```

   The system displays

   ```
   192 in Hex:    C0
   9 in Hex:      9
   200 in Hex:    C8
   1 in Hex:      1
   ```

3. Take the digits from the display and add a zero before each single digit. Using the sample display, 9 becomes 09 and 1 becomes 01.

4. Concatenate the digits together to arrive at your IP address is hexadecimal notation. Again using the sample IP address, your hexadecimal number would be

   c009c801

NOTE Characters may be upper- or lower-case.

5. To arrive at your listener address insert your family address, the port number of the listener, and your hexadecimal IP address into the format described in the preceding section. Then add the 16 zeros needed in the *Reserved* field.

   Given the listener port number 0ACE and the sample IP address, your listener address would be

   ```
   \x00020ACEc009c8010000000000000000
   ```

Creating the Service Access Facility Database

Once you have your hexadecimal listener address, you need to set up a Service Access Facility TCP/IP listener administrative database. To create the database, follow these steps:

1. Type

   ```
   nlsadmin -i tcp
   ```

2. To enter the address into the administrative database, type:

   nlsadmin -l '\x*FmlyPortInternetReserved*' tcp

   where *FmlyPortInternetReserved* is the address you computed for the listener.

   NOTE Be sure to include both the opening and closing single quotation marks, as indicated above.

   The TCP/IP network listener should start automatically every time the system is booted to multi-user mode.

For more information about the Service Access Facility, see sacadm(1M). For more information about the listener, see the Release 4.0 *System Administrator's Guide*.

Setting Up TCP/IP as the Preferred Network

When you install the TCP/IP Internet package, a file named `/etc/netconfig` is updated automatically with TCP/IP entries. `/etc/netconfig` determines what media will be used when you use a network service. The order of values in the `netconfig` file is what determines the order in which media is tried. For example, if you have the option to communicate across a network using either ISO STARLAN NETWORK or TCP/IP, and STARLAN precedes TCP/IP in the `netconfig` file, the system will attempt to use STARLAN whenever you request a network service. If the STARLAN request fails, the system then tries TCP/IP.

If you want TCP/IP to be your preferred network, edit `/etc/netconfig` and make the TCP/IP entries the first in the file.

For more information, see the discussion of Name-to-Address Mapping in the Release 4.0 *System Administrator's Guide*.

Starting TCP/IP

After you install the TCP/IP Internet package, you must reboot your system. To reboot, type

```
shutdown -y -g0 -i6
```

When you boot to multi-user mode, the TCP/IP startup script runs automatically.

Setting Up RFS over TCP/IP

If you intend to run RFS over TCP/IP, you need to add RFS to the listener database, and you need to supply the TCP/IP listener address of your RFS Domain Name Server to the file `/etc/rfs/tcp/rfmaster`.

To enter RFS into the listener database, type

```
nlsadmin -a 105 -c /usr/net/servers/rfs/rfsetup -y "rfsetup" tcp
```

To add your listener address to `rfmaster`, access the file and edit it with any supported text editor. Below is a sample `/etc/rfs/tcp/rfmaster` file:

```
#
# Sample rfmaster
#
# UNIX System V Release 4.0
dancedom P dancedom.dancer
dancedom.dancer A      \x00020ACEc009c8010000000000000000

# UNIX System V Release 3.2 with add-on TCP/IP
otherdom P otherdom.porter
otherdom.porter A    \x00020401c009c8a70000000000000000
```

The machine `dancer` is a UNIX System V Release 4.0 machine; it is the RFS domain name server to `dancedom`. The machine `porter` is a pre-UNIX System V Release 4 machine, whose port number is hexadecimal 0401 rather than 0ACE.

NOTE Rather than type the hexadecimal listener address into `rfmaster`, you may want to display the number on your terminal screen, then redirect it into the file. You can display the number by entering the following command: `nlsadmin -l- tcp`

To notify RFS of the TCP/IP transport provider, use the dname command with the `-N` option. For configurations using only TCP/IP as an RFS provider, enter

```
dname -N tcp
```

For configurations using TCP/IP and STARLAN as an RFS provider, enter

```
dname -N tcp,starlan
```

NOTE

The preceding section, "Setting Up the Listener," tells you how to determine your listener address.

For more information about `/etc/rfs/tcp/rfmaster`, see the *Remote File Sharing Administrator's Guide.*

If you intend to run RFS over TCP/IP and another provider, it is strongly recommended that you read "Multiple Transport Providers" in the *Remote File Sharing Administrator's Guide.*

Stopping and Restarting TCP/IP

If, sometime after initial installation and reboot, you need to restart TCP/IP, follow these steps:

1. To stop TCP/IP, do the following:
   a. If you are running RFS, stop RFS by issuing the `rfstop` command.
   b. Stop other processes using TPC/IP.
   c. Kill the TCP/IP listener by entering the command
      ```
      sacadm -k -p tcp
      ```
   d. Kill `inetd` by entering the command
      ```
      sacadm -k -p inetd
      ```
   e. Disable the TCP/IP STREAMS multiplexors by entering the command
      ```
      sh /etc/init.d/inetinit stop
      ```
2. To restart TCP/IP, do the following:
   a. In extreme cases, you may need to reset the controller board. To do so, type
      ```
      epump -d /dev/emdx
      ```
      where *x* is the slot number of the board.
   b. Enable the TCP/IP STREAMS multiplexors by entering the command
      ```
      sh /etc/init.d/inetinit start
      ```
   c. Restart `inetd` by entering the command
      ```
      sacadm -s -p inetd
      ```

d. Restart and the listener by entering the command

```
sacadm -s -p tcp
```

e. Restart RFS with the `rfstart` command.

3 Maintaining Security in a TCP/IP Environment

Granting Access to Network Machines

UNIX System V Release 4.0 includes various user programs that enable network users to perform operations on remote hosts. For example, Release 4.0 provides the commands `rlogin`, `rsh`, and `rcp`. `rlogin` (remote login) lets a user log in to a remote machine from his or her local machine; `rsh` (remote shell) lets a user create and use a shell on a remote machine; and `rcp` (remote copy) lets a user copy files to and from a remote machine.

Access to a machine on your local network is controlled by each machine's system administrator. Every machine's administrator determines which remote users to grant access and adds entries for those users to the machine's password files. The remote users can then log in to the machine, using the `rlogin` command, by supplying a username and a password.

NOTE This chapter assumes you are familiar with the files `/etc/shadow`, `/etc/passwd`, and `/etc/group`. For information, see the *System Administrator's Guide* or the *System Administrator's Reference Manual*.

You, as a machine's system administrator, may also wish to create a home directory on your machine for each remote user to whom you grant access. If a remote user does not have a home directory on your machine and logs in, the root directory (/) becomes the user's home directory.

Although remote users can log in to your machine once they have entries in your password database, they cannot run a number of remote processes (such as `rsh` and `rcp`) unless their machines are listed in `/etc/hosts.equiv` or the user's `.rhosts` file. In addition to allowing remote users to run certain remote processes on your machine, these files grant users access to your machine without their having to supply passwords.

This chapter explains how your system uses the `hosts.equiv` and `.rhosts` files, then gives you instructions for setting up and maintaining these files.

Administering the host.equiv and .rhosts Files

On a TCP/IP-based network, security is implemented at two levels: first, at the host level, and second, at the user level.

The `/etc/hosts.equiv` file is a general database that controls access at the host level.

The `.rhosts` file controls access to your machine at the user level. It is a file located in the home directory of a specific remote user on your machine, and it is used to allow or deny access to that specific user. There may be multiple instances of `.rhosts` on your machine, one for each remote user with a home directory.

When a remote user attempts to log in to your machine, the security-checking process goes like this:

- A remote user initiates an `rlogin`, and the `rlogin` daemon on your machine checks for the user's username in the password database. If no entry is found, the remote user is denied access.
- If the remote user has an entry in your password database, the daemon next checks for the remote machine's hostname in your `/etc/hosts.equiv` file. If the hostname is found, the remote user gains access.
- If no `/etc/hosts.equiv` entry is found, the system checks for a line with the remote machine's hostname (and, optionally, the remote user's username) in the `.rhosts` file in the user's home directory on your machine. If the entry is found, the remote user gains access.
- If no entry is found for the remote machine in either your `/etc/hosts.equiv` or the remote user's *home_dir/*`.rhosts` file on your machine, the remote user can `rlogin` to the machine after giving the right password. However, the remote user will receive the message `Permission denied` when attempting to run remote processes like `rcp` or `rsh`.

By creating and maintaining a `hosts.equiv` file, you can allow everyone on a particular host to log in to your machine and run remote processes like `rcp` and `rsh`.

If you want certain users on a particular host to access your machine, but not everyone, do not include the host in your `hosts.equiv` file. Instead put the host's name in the `.rhosts` file in each remote user's home directory on your machine.

The following sections tells you how to set up `hosts.equiv` and `.rhosts`.

The /etc/hosts.equiv File

In the simplest case, the `/etc/hosts.equiv` file is a list of machines, or hosts, from which users are permitted to log in (using `rlogin`) without supplying a password. `hosts.equiv` is a local file, pertaining only to the system in which it is found. The machine's system administrator can modify `hosts.equiv` by logging in as the superuser and using a text editor to make changes.

A typical `hosts.equiv` file has the following structure:

```
host1
host2
```

If a user attempting to log in from a remote host listed in `/etc/hosts.equiv` has an entry in the password database, he or she will be granted access without having to enter a password. If the same user attempts to log in from a host that is *not* listed in `hosts.equiv` or in */home/*`.rhosts`, he or she will be granted access only after entering a correct password.

A single + on a line in a machine's `hosts.equiv` file means that all known hosts (that is, all hosts listed in the `hosts` database) are trusted.

Example

Suppose the `hosts.equiv` file on your machine looks like this (note that the file is just a list of hostnames, one per line).

```
raks
dancers
jazz
```

Now you want to allow anyone on host `ballet` to have access to your machine. Edit `/etc/hosts.equiv` and add `ballet` to the list, as follows:

```
raks
dancers
jazz
ballet
```

Now add all the users on `ballet` to your `/etc/passwd`. Once you edit `/etc/passwd`, all users who can log in to `ballet` can also freely `rlogin` to your machine, without having to supply a password. Furthermore, users on `ballet` can remote copy from your machine and use a remote shell on your machine.

Refer to the `hosts.equiv`(4) manual page for more information.

The .rhosts File

The `.rhosts` file is located in a specific remote user's home directory on your machine; it is used to allow or deny access to that specific user.

The format of `.rhosts` is similar to that of `/etc/hosts.equiv`, that is,

```
host1
host2
```

The above entries in a user's `.rhosts` file mean that the user can log in from `host1` or `host2`, without supplying a password.

A user can allow others to log in to the remote machine using his or her login by adding usernames to the `.rhosts` file. For example, user `steve` has an `.rhosts` file on your machine that looks like this:

```
host1
host2
host1    jane
```

This means that `steve` can log in remotely to your machine as himself from `host1` and `host2`, and user `jane` can log in to your machine as `steve` from `host1`.

If your site does not require stringent security measures, the easiest way to administer machine access at the user level is to give each user an account in the password database and a home directory on your machine(s). Then ask the trusted users to create their own `.rhosts` files in their home directories on the machine(s).

Example 1

You want user `chris` to have access to your machine without restrictions. Have Chris create the file `.rhosts` in her home directory on your machine.

If Chris wants other users to have access to her login on your machine, her `.rhosts` file might look like this:

```
adams
monroe
jackson
adams      bob
jackson    jenny
```

Now user `chris` can access your machine remotely from `adams`, `monroe`, and `jackson`; bob can access your machine as `chris` from `adams`; and `jenny` has access as `chris` from `jackson`.

Example 2

Suppose you want only user `chris` to have access to your machine from host `samba`. To set up your machine, you would follow these steps:

1. Make sure `samba` is not in your `/etc/hosts.equiv` file.
2. Create the file `.rhosts` in `chris`'s home directory on your machine. The file should look like this:

```
samba
```

Security Issues

The only way to achieve anything resembling security in a sensitive environment is to exclude users from your password database; once someone knows a password, he or she can access your machine.

Some TCP/IP user services are especially problematic in an environment that requires strict security. For example, the `finger` command allows a user to display users on a remote host. The command starts the finger daemon `fingerd` on the remote machine, which then reports to the user the username and full name of everyone who is logged in to the machine. There is no authentication of the requestor and no auditing of the requests. Similarly, the `rwhod` daemon (on whose information `ruptime` bases its reports) freely passes around information about who is logged in to a machine.

There are no files or databases to restrict access to the information provided by these daemons; therefore, by default, they are not run. If you do not require strict security, you can run `fingerd` by deleting the comment character from the line that starts `fingerd` in `/etc/inetd.conf`. To start `rwhod`, add the line `/usr/sbin/in.rwhod` to the end of the startup script `/etc/inet/rc.inet`.

4. EXPANDING YOUR NETWORK

4 Expanding Your Network

Introduction

In time, you may need to attach more hosts to your network than you have allowable addresses, or you might want two networks in different buildings to share resources. In this chapter we first discuss generally some of the hardware that connects networks. Then we will talk about two ways you can expand your existing network:

- by connecting two or more networks to create an "internetwork."
- by creating a "subnet" on your network.

Hardware Devices for Expanding the Local Network

If your local network fails to meet your needs, you may want to expand it. Below are descriptions of the hardware devices that allow you to connect two or more local networks.

- Repeater: This is a device used at the physical layer of the network; it connects two networks together and copies each bit of a packet from one to the other.

  There are inherent limitations in the use of repeaters, given that they copy *all* data from one network to the other. Implementations like Ethernet, which require that data traverse the network within a specific amount of time, impose a maximum allowable length and number of repeaters.

- Bridge: This is a device used at the data link layer of the network protocol model. It selectively copies packets from one network to another.

  Bridges differ from repeaters in several ways. Because copying done by bridges is selective, this reduces traffic on the destination network. Since bridges copy whole packets, the geographical or timing constraints of the basic physical network can be extended.

  You can split a bridge as you would a repeater. Like repeaters, bridges copy raw packets, a scheme which works for all protocols above the data link layer. Since bridges copy whole packets, in theory, connections like this are "invisible" to the software on all the machines on the network.

- Router (sometimes called a "gateway"): This is a device that forwards packets of a protocol family, in this case TCP/IP, from one logical network to another. A logical network makes sense only to a particular protocol. Usually there is a one-to-one mapping between physical network and logical network, but subnets (explained in a later section) and bridges are exceptions to this rule. A collection of logical networks (all using the same protocol) connected via routers is called an "internetwork."

  The router may forward packets between different physical types of networks, for example, from an Ethernet to a ring network. It can also forward packets between two logical networks of the same type.

  During forwarding, a router looks inside the packet to find the destination address, then consults its routing table, which is normally kept up to date by having routers communicate with each other via some routing protocol. Note that it is possible to have a multilingual router—a single device that forwards packets for several protocol types, such as TCP/IP, ISO, or XNS.

- Application Gateway (sometimes called a "relay" or "forwarder"): This device and associated software enable networks using different protocols to communicate with each other. Because it translates protocols existing at all layers of the protocol model, you can use gateways to connect networks that differ on all layers from each other. You can also use an application gateway to connect parts of the same physical network that are using different protocols.

Creating an Internetwork

The TCP/IP protocol family provides intercommunication among host computers, terminal servers, and other equipment on one or more local-area or wide-area networks. A typical local network serves a limited area—within a building or between neighboring buildings. Hosts attached to the local network may function as file servers, mail servers, print servers, terminals, and workstations.

Some companies expand their networks by linking local networks together via a computer called an "IP router." A network configuration consisting of several local networks linked together by routers is often called an "internetwork."

Be careful not to confuse the term "internetwork" with the Internet. Your company can set up an internetwork if it needs to expand communications services, and you or one of your co-workers may have the responsibility of managing it. By contrast, the Internet is the name of a particular internetwork.

Configuring a Router

A TCP/IP network usually interconnects a number of hosts. Your UNIX System V Release 4.0 host is connected to a TCP/IP network via a hardware network interface. Individual TCP/IP networks are in turn interconnected via IP routers. IP routers forward IP packets from one TCP/IP network to another, and exchange routing information with each other to deliver packets across a number of networks. Other types of routers may forward traffic for protocol families other than TCP/IP.

If all of the hosts at your site are connected to a single TCP/IP network that is *not* interconnected with any other TCP/IP networks, an IP router is unnecessary. If your site comprises many TCP/IP networks, or if you wish to interconnect your IP network with other TCP/IP networks, you must configure the interconnections with IP routers in order for all hosts to communicate.

Many types of machines may serve as IP routers. A number of vendors offer machines dedicated entirely to the function of IP routing. A system may act both as a host (offering network services such as `rlogin`) and a router.

The `routed` (routing daemon) program implements a standard routing protocol in UNIX System V Release 4.0. If a system has only one network interface, `routed` will passively monitor the routing traffic (if that network is a broadcast network). If a system has two or more network interfaces, the `routed` program will actively participate in the exchange of routing information with other routers.

Some simple network applications do not require the router to run `routed`. For more information about `routed`, see "Setting Up the Routing Daemon" later in this section.

Assembling a router first involves setting up its hardware. Refer to the manuals that came with the router controllers for information on physical assembly. Once the router is connected to the networks it joins, you must configure the router's software.

Before actually configuring the software, make the following preparations.

- Assign the router a unique host name and a unique IP address for each network it is on. The Internet Protocol architecture requires each interface to have a unique IP address.
- Make sure you have acquired registered IP network numbers for each network the router is to connect.
- If you have not installed the TCP/IP Internet package, install it following the instructions in the Release 4.0 *Release Notes*.

Once you have TCP/IP installed, do the following:

1. Edit `/etc/hosts`.
    a. Add the host name and IP address of each interface on the machine.
    b. Add the names and IP addresses of all the hosts that the router can reach.
2. Access `/etc/networks` and add the network names and IP addresses of all the networks that the router can reach.
3. Access `/etc/strcf` and add a `cenet` statement for each interface on the machine. The statement should be added below the last `cenet` statement in the file. The `cenet` statement has the form

    ```
    cenet ip /dev/emd emd x
    ```

    where *x* is the device (slot) number.
4. Edit the startup script `/etc/inet/rc.inet`.

a. Locate the following lines at the beginning of the file:

```
#Default (single interface, not a gateway)
EMDMAJOR=`getmajor EMD | cut -d\  -f1`
/usr/sbin/ifconfig emd${EMDMAJOR} `uname -n` up
if [ $? -ne 0 ]
then
  exitcode=1
  echo "/etc/inet/rc.inet: /usr/sbin/ifconfig emd${EMDMAJOR} failed"
fi
```

Comment out the lines by inserting a pound sign (#) before each line.

b. Locate the following lines near the end of the file:

```
#/usr/sbin/ifconfig emdX nameX up
#if [ $? -ne 0 ]
#then
#    exitcode=1
#    echo "/etc/inet/rc.inet: /usr/sbin/ifconfig emdX' failed"
#fi
```

First delete the comment characters (#). Then substitute the device number of your first interface for the X in emdX, and the host name of the interface for nameX. For example, if your device number is 1 and the host name is saturn, the first line above would be:

```
/usr/sbin/ifconfig emd1 saturn up
```

c. Immediately following the block of code described above, locate the following lines:

```
#/usr/sbin/ifconfig emdY nameY up
#if [ $? -ne 0 ]
#then
#   exitcode=1
#   echo "/etc/inet/rc.inet: /usr/sbin/ifconfig emdY' failed"
#fi
```

Delete the comment characters, then substitute the device number of your second interface for the `Y` in **emd`Y`**, and the host name of the interface for **name`Y`**.

d. If you have more than two interfaces, add blocks of code identical to that described above for each remaining interface.

e. Enable the route daemon by deleting the comment character from the line

```
#/usr/sbin/in.routed
```

5. Access the file `/etc/master.d/ip` and change the parameter IPFORWARDING to 1.
6. Type `mkboot /boot/IP`.
7. Built a new `/stand/unix`, either via `cunix` or by entering `shutdown -i6`.

NOTE At this point, you should be able to ping (using the `ping` command) each machine that the router is configured to reach—provided the machine is up. However, the router cannot forward packets until you re-boot.

8. Re-boot.

Once you have your router set up, add the host names and IP addresses of the router's interfaces to the `/etc/hosts` and `/etc/networks` files on each machine on your local network.

Sample Router Files

Consider a router called `jekyll` that connects two networks. Like all routers, `jekyll` must have two hardware network interfaces. For the network software to work properly, the router must have a unique host name in the `/etc/hosts` file for each interface.

Because the internetwork on which `jekyll` resides is a simple one, `jekyll` doesn't need to run the routing daemon. It knows about all the available hosts by virtue of being directly connected to both local networks.

Notice in the sample file that, on the second network, `jekyll` has another name—`jekyll-hyde`. (`jekyll` is the router's primary name.) For ease of administration, similar host names are used for each network. Users on both networks can address the machine by the primary name `jekyll`, while the administrator can tell the difference between the two.

Note also that `jekyll` is added with *two* IP addresses.

```
#
# sample hosts file
#
# 192.9.200 -- eng -- Engineering Network
#
192.9.200.1       jekyll
192.9.200.2       usher
192.9.200.3       lenore
192.9.200.5       raven
# 192.9.201 -- mktg -- Marketing Network
#
192.9.201.11      quasimoto
192.9.201.12      godzilla
192.9.201.13      rodan
192.9.201.4       jekyll-hyde
```

The following screen is an excerpt from `jekyll's /etc/networks` file. The `networks` files contains just the names and IP addresses of the networks to which `jekyll` belongs.

```
#
# sample networks file
#
eng      192.9.200 # Engineering Network
mktg     192.9.201 # Marketing Network
```

Below is an excerpt from jekyll's `/etc/strcf` file. These are the last lines of the file, as it exists when it is installed for the first time.

Note that the sample file contains three lines beginning with `cenet`. The first appears in the file when it is installed and is commented out by default. The other `cenet` statements have been added to the file by `jekyll`'s system administrator. Each of the added lines includes the device number of one of `jekyll`'s two interfaces.

```
#
# excerpt from sample strcf file
#
        #
        # interfaces
        #
        ip = open /dev/ip
#       senetc ip eli /dev/emd0 /dev/emd1 emd0
#       uenet ip /dev/abc emd 0
#       denet ip /dev/def emd 0
#       cenet ip /dev/ghi emd 0
#       senet ip /dev/jkl0 /dev/jkl1 emd0

        cenet ip /dev/emd emd 1
        cenet ip /dev/emd emd 2
        loopback ip
```

The next screen is `jekyll`'s startup script `/etc/inet/rc.inet`.

```
# sample rc.inet startup script

exitcode=0

# Enable software loopback driver.
/usr/sbin/ifconfig lo0 localhost up
if [ $? -ne 0 ]
then
        exitcode=1
        echo "/etc/inet/rc.inet: /usr/sbin/ifconfig lo0 failed"
fi

# Default (single interface, not a gateway)
# EMDMAJOR=`getmajor EMD | cut -d\  -f1`
# /usr/sbin/ifconfig emd${EMDMAJOR} `uname -n` up
# if [ $? -ne 0 ]
# then
#   exitcode=1
#   echo "/etc/inet/rc.inet: /usr/sbin/ifconfig emd${EMDMAJOR} failed"
# fi

# Hosts acting as gateways
# To bring up your system as a gateway, you must:
#   set IPFORWARDING to 1 in /etc/master.d/ip, mkboot /boot/IP,
#               and build a new /stand/unix
#   edit /etc/networks: add network names and number
#   edit /etc/hosts: create unique host name for each interface
#   edit /etc/strcf: call cenet to link internet modules/drivers
#               above each interface
#   edit /etc/inet/rc.inet (this file):
#               comment out the default ifconfig entry above
#               un-comment the ifconfig entries below, changing X, Y to emd
#                 device (slot) numbers and nameX, nameY to your host names
#                 for each interface.
#               un-comment the line that starts in.routed
# See 'Expanding Your Network' in the Internet Administrator's Guide for
# more info
#
/usr/sbin/ifconfig emd0 jekyll up
if [ $? -ne 0 ]
then
        exitcode=1
        echo "/etc/inet/rc.inet: /usr/sbin/ifconfig emd0' failed"
fi
/usr/sbin/ifconfig emd1 jekyll-hyde up
if [ $? -ne 0 ]
```

(continued on next page)

```
47  then
48      exitcode=1
49      echo "/etc/inet/rc.inet: /usr/sbin/ifconfig emd1' failed"
50  fi
51  #/usr/sbin/in.routed
52
53
54  # Add lines here to set up routes to gateways, start other daemons, etc.
55  # For example,
56  # /usr/sbin/route add default your_nearest_gateway hops_to_gateway
57  # /usr/sbin/dname your_domain_name
58  # /usr/sbin/named
59
60  #return status to /sbin/init.d/inetinit
61  exit $exitcode
```

To modify the file so that `jekyll` can act as a router, `jekyll`'s system administrator has done the following:

1. commented out lines 14 through 20.
2. inserted the device number of `jekyll`'s first interface in lines 39 and 43; inserted the name `jekyll` in line 39.
3. inserted the device number of `jekyll`'s second interface in lines 45 and 49; inserted the hostname `jekyll-hyde` in line 45.
4. deleted the comment characters on lines 39 through 50.

Setting Up the Routing Daemon

Routers manage network traffic by maintaining routing tables—tables that contain information as to which networks and hosts can be reached by which routes. A routing table can be either static or dynamic.

If a router is on a simple internetwork—one that consists of two or three local networks, for example—it can manage traffic with static routing tables. It knows how to get to every machine on the internetwork by virtue of being directly connected to the local networks.

On a more complex network—one in which a router connects a local network to other routers and gateways—the router should be configured to use dynamic routing tables. Dynamic routing tables allow the router to route traffic to the most current gateway destinations.

A router builds and maintains dynamic tables by running the routing daemon `routed`. The routing daemon manages its routing table by exchanging routing information with gateways and other routers. When `routed` runs on a router, it broadcasts its routing table and listens for broadcasts from other directly connected routers. It continually updates its routing table based on those broadcasts. A routing daemon that both broadcasts its routing tables and listens for broadcasts from other routers is called an "active" `routed`.

NOTE `routed` can also run on a client; however, when it runs on a client, it simply listens for broadcasts and updates its local routing table; it does not broadcast to other machines. This is called a "passive" `routed`. For information about running `routed` on a client, see "Setting Up Router Clients" later in this guide.

When `routed` is first initialized on a router, it builds its table using the contents of the file `/etc/gateways`, which contains the address and distance to various networks and gateways. Once it starts to run, it immediately begins to update its table based on broadcasts from other routers and gateways.

You need to create `/etc/gateways`, using any supported text editor. The `/etc/gateways` file consists of a series of lines, each in the format

`[net | host] `*filename1* `gateway` *filename2 value* `[passive | active]`

where the operands have the following meanings:

`net | host`	`net` or `host` indicates that the route is to a network or to a specific host, respectively.
filename1	This is the name of the destination network or host. This may be a net name or host name, as specified in `/etc/networks` or `/etc/hosts`, or an IP address.
filename2	This is the name or address of the gateway to which messages should be forwarded.
value	This is a number indicating the number of "hops" to the destination host or network.

`passive | active`
: `passive` or active indicates that the gateway should be treated as either passive or active.

To run `routed` on a router automatically whenever TCP/IP is initialized, do the following:

1. Set up the `/etc/gateways` file.
2. Access the startup script `/etc/inet/rc.inet` and delete the comment character (#) from the following line:

```
#/usr/sbin/in.routed
```

For more information about `/etc/gateways` and `routed`, see the `routed`(1M) manual page.

Setting Up Router Clients

After you configure a router to access new networks, the clients of that router need to be informed of the path to those networks. This is done through the `route` command or by the routing daemon, `routed`.

The client uses the information you supply in the `route` command to build or update its internal routing table, or its routing daemon creates its routing table based on broadcasts from the router. The client then uses the table to find the "next hop" to the destination it is trying to reach.

Either the `route` command or `routed` must be run on the client every time TCP/IP is initialized. To ensure that either the command or the daemon is run automatically after booting, modify the client's startup script, `/etc/inet/rc.inet`. You can edit the script to specify different kinds of routing behavior, as described below:

- If there is only one router on the local network, you can set up the `rc.inet` script so that, by default, your client's attempts to reach outside the local network are directed to this router. Since there is only one router to access external networks, no additional processing is needed on the client.

- If your client has more than one router available to it on the local network, you can direct traffic to specific networks through a specific routers.
- If your client is connected to a router the runs an active routing daemon, the client can take advantage of the router's dynamic routing tables by running a passive `routed`.

The following sections tell you how to specify the kinds of routing described above.

Specifying a Default Router

If there is only one router connected to a local network, clients should direct all external traffic through the router by default.

To specify default routing, do the following:

1. Access `/etc/inet/rc.inet` on the client and locate the line

```
#/usr/sbin/route add default your_nearest_gateway hops_to_gateway
```

2. Enable default routing by editing the line, as follows:
    a. Delete the comment character (#).
    b. Substitute the router's host name on your local network for *your_nearest_gateway*.
    c. Substitute the argument `1` for *hops_to_gateway*.

Using an example from a preceding section, assume that networks named `eng` and `mktg` are connected by a router. The router has two network interfaces, one for each network, and each interface has a different host name. On the network named `eng`, the router's host name is `jekyll`; on `mktg`, its name is `jekyll-hyde`.

If your client is on the network named `eng`, you would edit the `route` command in your `/etc/inet/rc.inet` to look like this:

```
/usr/sbin/route add default jekyll 1
```

If your client is on the network named `mktg`, your `route` command would look like this:

```
/usr/sbin/route add default jekyll-hyde 1
```

In both examples, the argument `default` indicates that all communication directed outside the local network from your client should be routed through the router. The argument `1` indicates that the router is connected to the local network; messages "hop" once, or go through one router, to reach their destination.

Using Specific Routers

If your machine is a client of more than one router that uses static routing tables, you may want to specify which routers should route traffic to which networks.

For example, assume your system has two routers, `barker` and `lovett`, available to it on the local network. Your system needs to communicate regularly with networks named `sales` and `adv`. You know that the router `barker` maintains information about `sales` in its routing table, and `lovett` knows how to get to `adv`. To specify that traffic from your system should follow these routes, you would edit the `route` command in `/etc/inet/rc.inet` as follow:

```
/usr/sbin/route sales barker 1
```

Below this statement in `/etc/inet/rc.inet` you would add a second route command, as follows:

```
/usr/sbin/route adv lovett 1
```

These commands set up the your system so that it sends messages intended for different networks through different routers, rather than sending all traffic through the same router.

A powerful way to set up routing for more complex networks is to provide several specific `route add` commands for frequently used networks, as well as a `route add default` command to handle traffic to all other networks. This is particularly useful when routers, in turn, use other routers.

For more information about the `route`, command, see the `route`(1M) manual page.

Running the Routing Daemon on a Client

If your machine is a client to a router that maintains a dynamic routing table, you can set up your client to update its own tables based on the router's broadcasts.

If the router is running `routed`, `routed` listens for broadcasts from other routers and gateways and continually updates its routing tables based on the information in those broadcasts. It also broadcasts its own routing tables so that other machines can use them for updating their own routing tables. Because `routed` on a router both broadcasts its own routing table and listens for broadcasts from other routers, it is known as an "active" `routed`.

When `routed` is run on a client (a machine with only one network interface), it listens on the network and waits for the local router to broadcast its routing table. It then uses the broadcast information to update its own table. Because `routed` on a client does not broadcast its routing table, it is called a "passive" `routed`.

NOTE

Even if the local router uses dynamic routing tables, you can choose not to run `routed` on the client. In this case, the client's tables remain static, even though the router's are dynamic. You might choose not to run `routed` on the client if the client is on a simple internetwork, and the routes to its usual destinations do not change.

If the client's router uses static routing tables (that is, it does not run `routed`), do not run `routed` on the client.

To enable the routing daemon on a client, following these steps:

1. Access the startup script `/etc/inet/rc.inet` on the client and locate the following line:

   ```
   #/usr/sbin/in.routed
   ```

2. Delete the comment character (#).

Setting Up Subnets

Subnets are logical subsections of a single TCP/IP network. For administrative or technical reasons, many organizations chose to divide one network into several subnets.

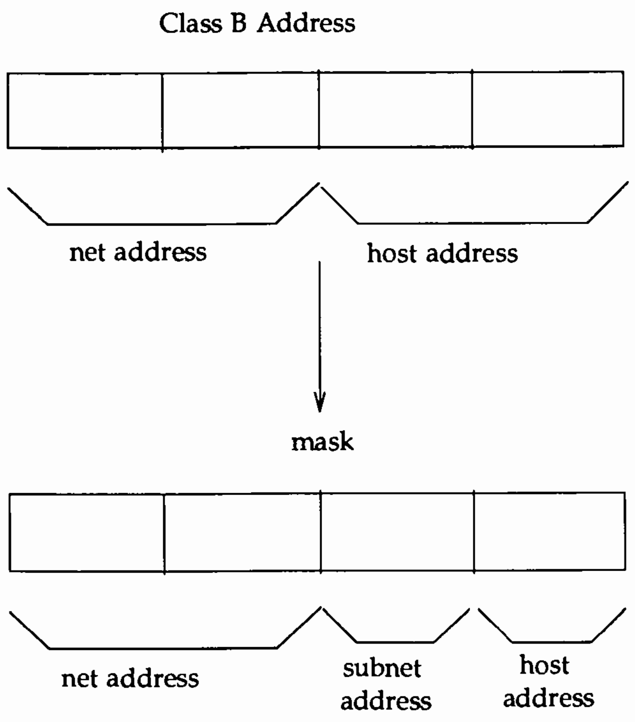
Routing can get very complicated as the number of networks grows. For example, a small organization might give each local network a Class C number. As the organization grows, administering network numbers may get out of hand. A better idea is to allocate a few Class B network numbers for each major division in a company: one for Engineering, one for Operations, and so on. Then, divide each Class B network into physical networks using subnets. In this way, you can isolate hosts from changes you might make to the network in remote parts of the organization.

Subnets allow you more flexibility when assigning network addresses. The Internet Protocol allows 127 Class A networks with 24 bit host fields; 16,383 Class B networks with 16 bit host fields; and over two million Class C networks with eight bit host fields.

Network Masks

Typically, you create subnets by using a subnetting scheme called the "Address Mask." When setting up your network, you should select a network-wide "network mask." A network mask determines which bits in the IP address will represent the subnet number. The remaining bits represent the host within the subnet. For example, you could configure an organization's internetwork as a Class B network. Then you could assign each local subnet a subnet number within that network. The 16 bits could be allocated as eight for subnet and eight for host, or nine for subnet and seven for host, and so on. Your decision would be transparent to everyone outside that organization. The following figure illustrates the effect of the network mask:

Figure 4-1: Network Mask



You can express network masks as a single hexadecimal number, or as four octets of decimal numbers, as described earlier in this guide. The default is a mask of 0xFF000000 (255.0.0.0) for Class A networks, 0xFFFF0000 (255.255.0.0) for Class B networks, and 0xFFFFFF00 (255.255.255.0) for Class C networks. You only have to specify network masks explicitly when they are wider (that is, have more one-bits) than the default values. One common case is a Class C mask on a Class B network. A Class B network provides you with 256 possible subnets, each one of which can accommodate 254 possible hosts (remember, 0 and 255 are not acceptable host addresses). But you may know that none of your subnets will ever have more than, say, 128 hosts, while you may need more than 256 subnets. In that case, you could decide to use nine bits for the subnet number instead of eight, and seven for the host addresses. The appropriate mask for this would be 0xFFFFFF80, or 255.255.255.128 (2 to the power of 7 is 128, and 128 subtracted from the possible 256 is 128).

Given the above scheme, and a network address of, for instance, 131.60, the address for the first host of the first subnet would be 131.60.0.129.

The `netmasks` File

The `/etc/netmasks` file contains the default netmask for your system. To set up the netmask, you need to create this file. Here is a sample `/etc/netmasks`.

```
#
# Network masks
#
# only non-standard subnet masks need to be defined here
#
# Network        netmask
128.32.0.0       255.255.255.0
```

Create an entry with the network number and network mask on a separate line for each network that is subnetted.

You can use `ifconfig` to override the network masks manually. For more information about `ifconfig`, refer to the `ifconfig`(1M) manual page.

For example, consider Class B network 128.32 with an eight bit wide subnet field (and, therefore, an eight-bit wide host field). The `/etc/netmasks` entry for this network would be:

    128.32.0.0 255.255.255.0

You can enter symbolic names for subnet addresses in the `/etc/hosts` file. You can then use these subnet names instead of numbers as parameters to commands.

For more information about `netmasks`, see the `netmasks`(4) manual page.

Changing from a Non-Subnetted to a Subnetted Network

Follow these steps to change from an internetwork that does not use subnets to one that is subnetted.

1. Decide on the new subnet topology, including considerations for subnet routers and locations of hosts on the subnets.
2. Assign all subnet and host addresses.
3. Edit `/etc/netmasks` as mentioned previously.
4. Edit `/etc/hosts` on all hosts to change host address.

Examples of Subnets

The following examples show network installations where subnets are (and are not) in use:

128.32.0.0	Berkeley	Class B network (subnetted)	netmask 255.255.255.0
36.0.0.0	Stanford	Class A network (subnetted)	netmask 255.255.0.0
10.0.0.0	Arpanet	Class A network (non-subnetted)	netmask 255.0.0.0

All of the University of California at Berkeley is assigned the network number 128.32.0.0, so that any external router only needs to know one route to reach Berkeley. Within the campus, a Class C subnet mask is used to give each local network a subnet number, with 254 hosts on each of the 254 possible subnets. (Zero and all ones, that is 255, are reserved.) Stanford University uses a Class A network number with a Class B network mask, for 254 subnets of 65534 hosts each. The Arpanet is a Class A network without subnets; therefore, the default Class A netmask is used.

5. USING DOMAIN NAME SERVICE

5 Using Domain Name Service

An Overview of the Domain Name Service

Domain Name Service (DNS) is an application layer protocol that is part of the standard TCP/IP protocol suite. Specifically, DNS is a *naming* service; it obtains and provides information about hosts on a network.

Domain Name Service performs naming between hosts *within* your local administrative domain and *across* domain boundaries. It is distributed among a set of servers, commonly known as "name servers," each of which implements DNS by running a daemon called named.

NOTE The named daemon is also called the Berkeley Internet Name Domain service, or BIND, because it was developed at University of California at Berkeley.

On the client's side, DNS is implemented through the "resolver." The resolver is neither a daemon nor a particular program; rather, it is a library compiled into applications that need to know machine names. The resolver's function is to resolve users' queries; to do that, it queries a name server, which then returns either the requested information or a referral to another server.

DNS is dependent on the hierarchical structure of the Internet and adds to that structure the concept of domain "zones." This chapter describes the organization of domains as it relates to DNS on the Internet, then tells you how to set up and maintain DNS on your network machines.

The Domain Hierarchy and DNS Administrative Zones

Domain names used by DNS reflect the hierarchical organization of the public networks. Before you set up DNS for your organization, you should understand the organization of the network, your place in the overall structure of the network, and how domain names reflect that structure.

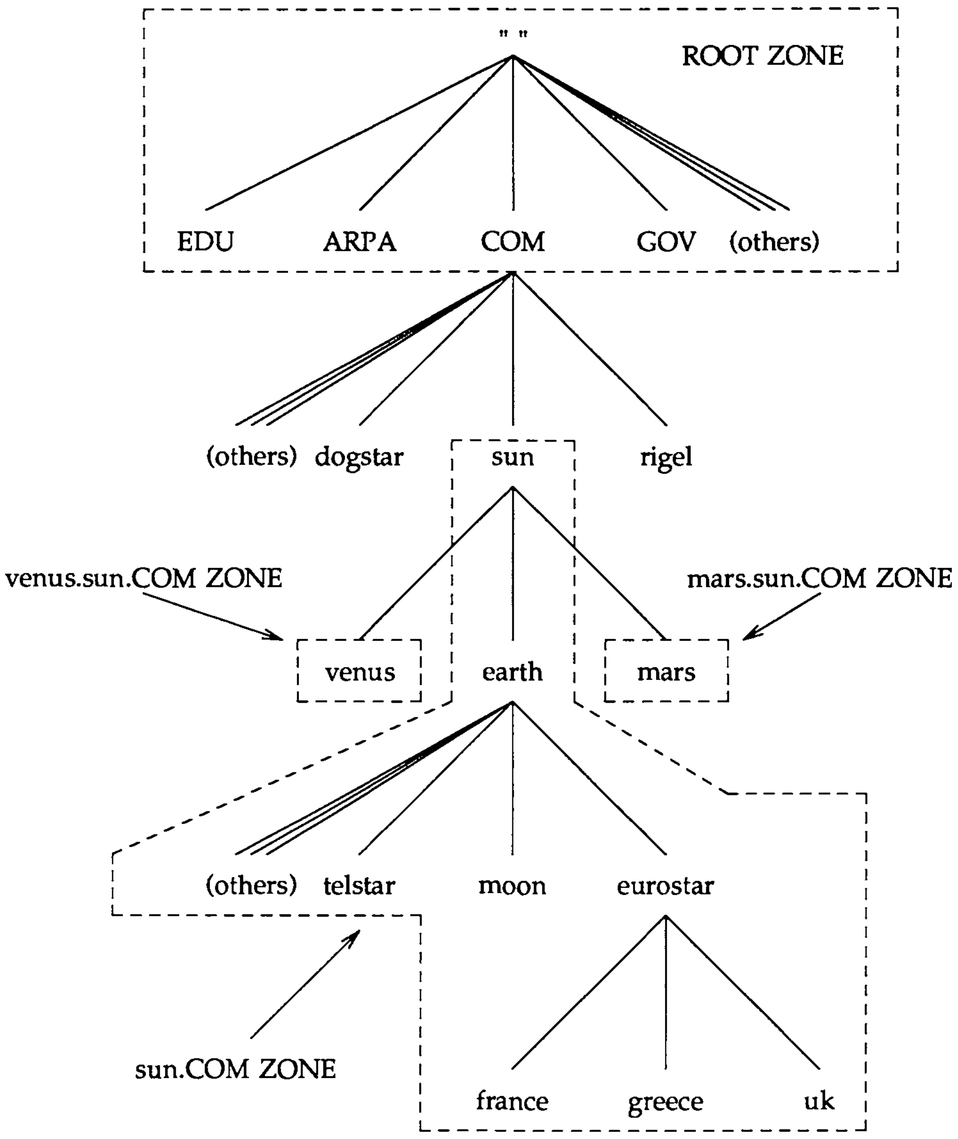
For information about the structure of the Internet and domain naming conventions, see "Establishing a Domain" earlier in this guide.

The Domain Name Service introduces the concept of "zones" to the Internet network model. A zone is a hierarchical community of hosts, administered by a single authority and served by a set of name servers. This community can include individual hosts, plus every name server and its clients (a subzone) nested beneath the authoritative set of name servers for the zone. Zones usually represent administrative boundaries, such as your local administrative domain.

However, a zone is not limited to just one administrative domain. A zone is a domain, plus all or some of the domains under it.

The next figure shows how some fictional zones might be delegated. The illustration shows four zones: the root zone, served by the Internet root domain name servers, and three zones served by name servers at domain sun.COM. The dotted lines point to the areas making up each zone.

Figure 5-1: Administrative Zones



Zones take their names from the label of the domain at the top of the zone hierarchy. In Figure 6-1, the names of the four domains within dotted lines are:

```
sun.COM   venus.sun.COM   mars.sun.COM   . (the root zone)
```

Zone `sun.COM` takes its name from the label of the second level domain `sun.COM`. However, zone `sun.COM` does not have the same administrative authority as the domain of the same name. Zone `sun.COM` consists only of

- domain sun
- local administrative domain `earth`
- subdomain `eurostar`.

The zone `sun.COM` does not include domains venus and `mars`. They have their own separately administered zones: `venus.sun.COM` and `mars.sun.COM`, respectively. However, venus and `mars` are part of the `sun.COM` domain hierarchy.

The domain hierarchy and name space described so far keeps track of information by host name. This enables the named daemon to perform name-to-address mapping.

In addition, there is a special domain recognized by DNS called IN-ADDR.ARPA, which facilitates address-to-name mapping. IN-ADDR.ARPA contains essentially the same information as the hosts name space, but it is expressed in terms of IP addresses.

A name in the IN-ADDR.ARPA domain has four labels preceding it, corresponding to the four octets of an IP address. This host address is listed from right to left. For example, a host whose IP address is 128.32.0.4 has the IN-ADDR.ARPA domain name

4.0.32.128.IN-ADDR.ARPA.

Therefore, if named knows the IP address of a host, it can find out the host's fully qualified domain name by consulting a file representing the IN-ADDR.ARPA domain. (This file, commonly called `hosts.rev`, is explained in a later section.)

The Administrator's Role

Administering DNS involves not only running the appropriate programs on the servers and clients, but also determining domain names, answering complaints, and filling out registration and other forms, should you join a public network. This section contains overall procedures for setting up DNS, based on the responsibilities you might have as a DNS administrator.

Your administrative responsibilities for DNS depend on your domain's position within the overall network hierarchy. For example, managing one set of name servers in a small administrative domain entails less responsibility than managing the authoritative set for a large zone. Responsibilities depend on whether you are the chief authority for a domain or zone, or an administrator reporting to the chief authority.

The NIC divides administrators on the Internet into domain administrators, who have primary responsibility for a domain, and technical contacts, who work with domain administrators to maintain a zone. Descriptions of each position appear below.

The Domain Administrator

The domain administrator (DA) is a coordinator, manager, and technician for a second-level or lower domain. The DA's responsibilities include:

- Registering the domain.

  If your domain is going to be on a public network, you must register it (if you have not done so already). The domain must have a unique name within its level of the network hierarchy. To register your domain with the Internet, obtain the Domain Registration Form from NIC and complete it according to the instructions in Appendix B of this guide. To join other public networks, contact the organization in charge of the network and request the appropriate domain registration form.

NOTE To belong to multiple networks such as the Internet, BITNET, CSNET, you need to register with only one network, as domain names are independent of a network's physical topology.

- Naming hosts and verifying that names within the domain are unique.

  At many sites, users name their individual hosts while the administrators name the servers. The administrator should ensure that there are no duplicate names within a zone.

- Understanding the functioning of the name servers and making sure the data is current at all times.

  This includes either setting up the DNS-related files and programs, or delegating this authority to other technically competent people (such as the technical contact). This chapter should provide you with the basic information you need for understanding DNS. However, as the domain administrator, you should gain more specific technical knowledge of your particular network. Contact the public network to which you belong and ask for technical papers regarding it.

The Technical Contact

The primary responsibility of a technical contact is to maintain DNS programs and files for a zone and to keep the zone's name servers running. Technical contacts interact with their Domain Administrators and with DA's of other domains to solve network problems.

DNS Clients and Servers

As mentioned earlier, there are two sides to DNS: name servers running the named daemon, and clients running the resolver.

Clients

A name server running named can also run the resolver; therefore, there can be two kinds of clients

- client-only
- client/server

A client-only client does not run the named daemon; instead, it consults the resolver, which provides a list of possible name serving machines to which queries should be directed.

A client/server is a machine that uses the domain name service provided by named in order to resolve a user's queries. However, if the daemon dies or hangs, the client/server might be able to solve queries through its resolver.

Servers

You implement DNS for a zone not on a single server, but on a *set* of servers. This set must include two master servers, and may or may not include other servers.

- Master Servers

  The "master" name servers maintain all the data corresponding to the zone, making them the authority for that zone. These are commonly called "authoritative" name servers. The data corresponding to any given zone must be available on at least *two* authoritative servers. You should designate one name server as the primary master server and at least one as a secondary master server, to act as a backup if the primary is unavailable or overloaded.

  The "primary" master server is the name server where you make changes for the zone. This server loads the master copy of its data from disk when it starts up named. The primary server may also delegate authority to other servers in its zone, as well as to servers outside of it.

  The "secondary" master server is a name server that maintains a copy of the data for the zone. The primary server sends its data and delegates its authority to the secondary server. When the secondary server boots named, it requests all the data for the given zone from the primary. The secondary server then periodically checks with the primary to see if it needs to update its data.

A server may function as a master for multiple zones: as a primary for some zones, and as a secondary for others.

A server at the root level of the network is called a "root domain name server." On the Internet, root domain name servers are maintained by the NIC. If a network is not connected to the Internet, primary and secondary name servers must be set up and administered for the root level of the local network.

- Caching and Caching-Only Servers

  All name servers are caching servers. This means that the name server caches received information until the data expires. (The expiration process is regulated by the `time to live` field attached to the data when it is received from another server.)

  Additionally, you can set up a "caching-only server" that is not authoritative for any zone. This server handles queries and asks other name servers who have the authority for the information needed. But the caching-only server does not maintain any authoritative data itself.

Setting Up DNS on a Client

Setting up DNS on a client involves three tasks:

1. creating the file `resolv.conf`
2. modifying the file `netconfig`
3. copying the dynamic sockets library.

NOTE If you are setting up DNS on a name server, you need to complete these steps, in addition to setting up boot and data files and editing the startup script. These additional tasks are described later in this chapter, in the section called "Setting Up DNS on a Name Server."

Creating resolv.conf

The domain name server uses several files to load its database. At the resolver level, it needs a file (called `/etc/resolv.conf`) listing the addresses of the servers where it can obtain the information needed. Whenever the resolver has to find the address of a host (or the name corresponding to an address) it builds a query package and sends it to the name servers it knows of (from `/etc/resolv.conf`). The servers either answer the query locally or use the services of other servers and return the answer to the resolver.

The `resolv.conf` file is read by the resolver to find out the name of the local domain and the location of name servers. It sets the local domain name and instructs the resolver routines to query the listed names servers for information. Every DNS client-only system on your network must have a `resolv.conf` file in its `/etc` directory.

NOTE On a system where a name server is running, an `/etc/resolv.conf` is unnecessary.

The first line of the file lists the domain name in the form

```
domain domain_name
```

where *domain_name* is the name registered with the NIC. Succeeding lines list the IP addresses that the resolver should consult to resolve queries. IP address entries have the form:

`nameserver` *IP_address*

Below is a sample `resolv.conf` file:

```
; Sample resolv.conf file
domain Podunk.Edu
; try local name server
nameserver 127.0.0.1
; if local name server down, try these servers
nameserver 128.32.0.4
nameserver 128.32.0.10
```

NOTE

It is not advised that you have an `/etc/resolv.conf` file on a system that is running a name server, as `/etc/resolv.conf` causes the system to be less efficient; however, when the name server is not running, `/etc/resolv.conf` consults other name servers to answer queries. Therefore, if it is important that your system have little or no down-time, create `/etc/resolv.conf` on your system and include the loopback address, followed by the addresses of other name servers.

Modifying netconfig

Once you have created the appropriate `/etc/resolv.conf` file on each machine that is to run the resolver, you need to edit the `netconfig` file. Look at the file and make sure that all transport providers listed in the `/etc/netconfig` file that provide IP connections have the name-to-address resolving routing that uses DNS (possibly `resolv.so`) listed in the name `lookups` field. (See the `netconfig`(4) manual page for more information about `netconfig`.)

For example, if your `/etc/netconfig` file lists the following entries:

```
#netid semantics    flags family proto device      nametoaddr_libs
tcp    tpi_cots_ord v     inet   tcp   /dev/tcp    /usr/lib/tcpip.so
udp    tpi_clts     v     inet   udp   /dev/udp    /usr/lib/tcpip.so
icmp   tpi_raw      -     inet   icmp  /dev/icmp   /usr/lib/tcpip.so
rawip  tpi_raw      -     inet   -     /dev/rawip  /usr/lib/tcpip.so
```

you will have to modify the `nametoaddr_libs` field so it says:

```
#netid semantics    flags family proto device      nametoaddr_libs
tcp    tpi_cots_ord v     inet   tcp   /dev/tcp    /usr/lib/tcpip.so,/usr/lib/resolv.so
udp    tpi_clts     v     inet   udp   /dev/udp    /usr/lib/tcpip.so,/usr/lib/resolv.so
icmp   tpi_raw      -     inet   icmp  /dev/icmp   /usr/lib/tcpip.so,/usr/lib/resolv.so
rawip  tpi_raw      -     inet   -     /dev/rawip  /usr/lib/tcpip.so,/usr/lib/resolv.so
```

NOTE In single-user mode, **named** is not available, so the `/etc/netconfig` file *must* include a lookup routine that uses local text files (`tcpip.so` in the example above is the default). You should therefore put the local machines' interface addresses and at least a couple of machine names and addresses in `/etc/hosts`.

Copying the Dynamic Sockets Library

The TCP/IP Internet package provides two versions of the sockets library: `/usr/lib/libsockhost.so` and `/usr/lib/libsockdns.so`. The former uses the `/etc/hosts` file to resolve host addresses; the latter goes to the domain name server to resolve addresses. By default, `/usr/lib/libsocket.so` is a copy of `/usr/lib/libsockhost.so`, which means that all TCP/IP commands use `/etc/hosts`.

If you are running DNS and you want TCP/IP commands to use the name server, you must copy `/usr/lib/libsockdns.so` into `/usr/lib/libsocket.so`. Copy the library while in single-user mode, with no TCP/IP commands or daemons active.

NOTE

Remember, machines acting as name servers are clients of other name servers (including itself), so name servers must copy the sockets library, even if they are not running the resolver; that is, a `telnet a.b.com` performed on a name server requires that the `/usr/lib/libsockdns.so` be copied into `/usr/lib/libsocket.so`.

Setting Up DNS on a Name Server

Because every name server is a client of other name servers, you must complete the steps involved in setting up DNS on a client when you set up a machine to be a name server. These steps are

1. creating the file `resolv.conf`
2. modifying the file `netconfig`
3. copying the dynamic sockets library.

Instructions for completing these steps appear in the preceding section, "Setting Up DNS on a Client."

Once you complete these steps, you need to do some additional tasks to set up DNS on a machine that is to be a name server. These steps are

1. creating the boot and data files that `named` needs.
2. editing the server's startup script `/etc/inet/rc.inet` and removing the comment character (#) from the `in.named` line.

Instructions for completing these steps appear in this section.

If your local network is not on the Internet, you must set up primary and secondary servers in the root-level domain on the local network. Instructions for setting up a root domain name server appear at the end of this section.

Creating Boot and Data Files

In addition to the daemon `named`, DNS on a name server consists of a boot file and local data files. The default location of the boot file is `/etc/named.boot`. Common names for the local data files are `named.ca`, `named.local`, `hosts`, and `hosts.rev`. In the descriptions of these files that follow, these names are used. However, you can name these files whatever you wish. In fact, it is suggested that you use names other than the ones used in this guide, to avoid confusion with files with similar names.

- `named.boot`

  The boot file `named.boot` establishes the server as a primary, a secondary, or a caching-only name server. It also specifies the zones over which the server has authority, and which data files it should read to get its initial data.

The boot file is read by `named` when the daemon is started by the server's startup script `/etc/inet/rc.inet`. The boot file directs `named` either to other servers or to local data files for a specified domain.

- `named.ca`

  `named.ca` establishes the names of root servers and lists their addresses. If you are connected to the Internet, `named.ca` lists the Internet name servers; otherwise, it lists the root domain name servers for your local network.

  `named` cycles through the list of servers until it contacts one of them. It then obtains from that server the current list of root servers, which it uses to update `named.ca`.

  To avoid confusion with the caching process, with which this has very little to do, you may want to call your file something like `named.root` or `boot.root`.

- `hosts`

  The `hosts` file contains all the data about the machines in the local zone. The name of this file is specified in the boot file. To avoid confusion with `/etc/hosts`, name the file something other than `hosts`. You may want to give the file name the suffix `zone`; in the following illustration, the file is called called `A.puzone`.

- `hosts.rev`

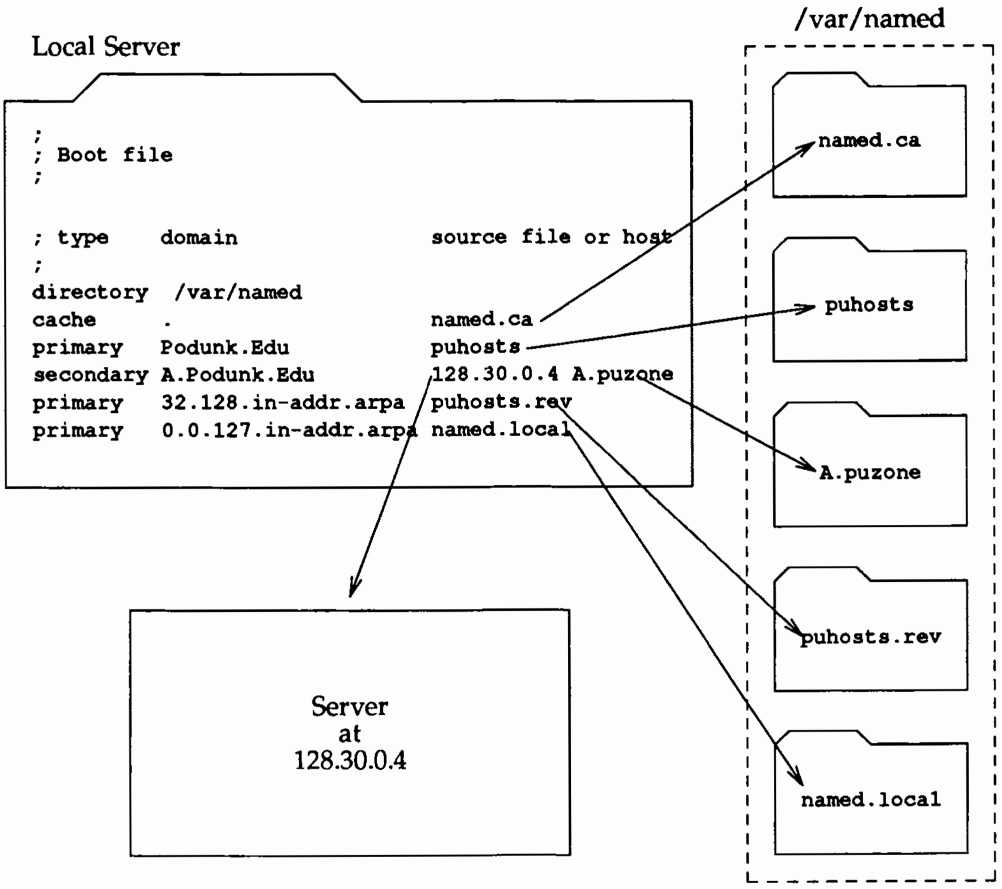
  The `hosts.rev` file specifies a zone in the IN-ADDR.ARPA domain (the special domain that allows inverse mapping). The name of this file is specified in the boot file. In the illustration, the file is called `puhosts.rev`.

- `named.local`

  The `named.local` file specifies the address for the local loopback interface, or `localhost`, with the network address 127.0.0.1. The name of this file is specified in the boot file. As with all other files, it can be called something other than the name used in this guide.

The relationship between the boot file and the data files (or, alternatively, other servers) is illustrated by the following figure:

Figure 5-2: Boot File and Data Files



```
;
; Boot file
;

; type     domain                     source file or host
;
directory  /var/named
cache      .                          named.ca
primary    Podunk.Edu                 puhosts
secondary  A.Podunk.Edu               128.30.0.4 A.puzone
primary    32.128.in-addr.arpa        puhosts.rev
primary    0.0.127.in-addr.arpa       named.local
```

Setting Up the Boot File

The contents of the boot file varies, depending on the type of server. This section describes boot files for primary and secondary master servers and caching-only servers.

Boot File for a Primary Master Server

The following is a sample boot file for a primary server:

```
;
; Sample named.boot file for Primary Master Name Server
;

; type  domain                    source file or host
;
directory /var/named
cache     .                       named.ca
primary   Podunk.Edu               puhosts
primary   32.128.in-addr.arpa      puhosts.rev
primary   0.0.127.in-addr.arpa          named.local
```

The entries in the file are explained below.

`directory` The following line in the boot file designates the directory in which you want the name server to run:

```
directory /var/named
```

This allows the use of relative path names for the files mentioned in the boot file or, later, with the `$INCLUDE` directive. It is especially useful if you have many files to be maintained and you want to locate them all in one directory dedicated to that purpose.

If there is no `directory` line in the boot file, all file names listed in it must be full pathnames.

`cache` A name server needs to know which servers are the authoritative name servers for the root zone. To do this you have to list the addresses of these higher authorities.

All servers should have the following line in the boot file to find the root name servers:

```
cache .      named.ca
```

The first field indicates that the server will obtain root servers hints from the indicated file, in this case `named.ca` (located in the directory `/var/named`).

`primary` To set up a primary server, you need to create a file that contains all the authoritative data for the zone. Then you create a boot file that designates the server as a primary server and tells it where to find the authoritative data.

The following line in the boot file names the server and the data file.

```
primary      Podunk.Edu   puhosts
```

The first field designates the server as primary for the zone stated in the second field. The third field is the name of the file from which authoritative data is read.

The lines

```
primary      32.128.in-addr.arpa      puhosts.rev
primary      0.0.127.in-addr.arpa     named.local
```

indicate that the server is also a primary server for the domains `32.128.in-addr.arpa` (that is, the reverse address domain for Podunk.Edu) and `0.0.127.in-addr.arpa` (that is, the local host loopback), and that the data for them is to be found, respectively, in the files `puhosts.rev` and `named.local`.

Boot File for a Secondary Master Server

The following is a sample boot file for a secondary server in the same domain as the above primary server:

```
;
; Sample named.boot file for Secondary Master Name Server
;

; type      domain                  source file or host
;
directory /var/named
cache       .                       named.ca
secondary   Podunk.Edu              128.32.0.4 128.32.0.10 128.32.136.22 puhosts.zone
secondary   32.128.in-addr.arpa     128.32.0.4 128.32.0.10 128.32.136.22 purev.zone
primary     0.0.127.in-addr.arpa    named.local
```

In appearance, this file is very similar to the boot file for the primary server; the main difference is to be found in the lines

```
secondary   Podunk.Edu              128.32.0.4 128.32.0.10 128.32.136.22 puhosts.zone
secondary   32.128.in-addr.arpa     128.32.0.4 128.32.0.10 128.32.136.22 purev.zone
```

The word `secondary` establishes that this is a secondary server for the zone listed in the second field, and that it is to get its data from the listed servers (usually the primary server followed by one or more secondary ones); attempts are made in the order in which the servers are listed. If there is a filename after the list of servers (as in the example above), data for the zone will be put into that file as a backup. When the server is started, data are loaded from the backup file, if it exists, and then one of the servers is consulted to check whether the data is still up to date.

This ability to specify multiple secondary addresses allows for great flexibility in backing up a zone.

The interpretation of the other "secondary line" is similar to the above. Note also that although this is a secondary server for the domain Podunk.Edu and 32.128.in-addr.arpa, this is a primary server for 0.0.127.in-addr.arpa. (the local host).

NOTE

A server may act as the primary server for one or more zones, and as the secondary server for one or more zones; it is the mixture of entries in the boot file that determines it.

Boot File for Caching-Only Server

The following is a sample boot file for a caching only server:

```
;
; Sample named.boot file for Caching Only Name Server
;
; type   domain                  source file or host
;
cache    .                       named.ca
primary  0.0.127.in-addr.arpa    named.local
```

You do not need a special line to designate a server as a caching only server. What denotes a caching-only server is the absence of authority lines, such as `secondary` or `primary` in the boot file. As explained above, a caching-only server does not maintain any authoritative data; it simply handles queries and asks the hosts listed in the `named` file for the information needed.

Setting Up the Data Files

All the data files used by the DNS daemon `named` are written in Standard Resource Record Format. In Standard Resource Record Format, each line of a file is a record, called a Resource Record (RR). Each DNS data file must contain certain Resource Records. This section describes the DNS data files, and the Resource Records each file should contain. Following this section is a discussion of the Standard Resource Record Format, including an explanation of each Resource Record relevant to the DNS data files.

The `hosts` File

The `hosts` file contains all the data about all the machines in your zone, including server names, addresses, host information (hardware and operating system information), canonical names and aliases, the services supported by a particular protocol at a specific address, and group and user information related to mail services. This information is represented in the the records NS, A, HINFO, CNAME, WKS, MX, MB, MR, MG. The file also include the SOA record, which indicates the start of a zone and includes the name of the host on which the hosts data file resides.

The following is a sample hosts file:

```
;    sample hosts file
@               IN   SOA     ourfox.Sample.Edu. kjd.monet.Sample.Edu. (
                             1.1         ; Serial
                             10800       ; Refresh
                             1800        ; Retry
                             3600000     ; Expire
                             86400 )     ; Minimum
                IN   NS      ourarpa.Sample.Edu.
                IN   NS      ourfox.Sample.Edu.
ourarpa         IN   A       128.32.0.4
                IN   A       10.0.0.78
                IN   HINFO   3B2 UNIX
arpa            IN   CNAME   ourarpa
ernie           IN   A       128.32.0.6
                IN   HINFO   3B2 UNIX
ourernie        IN   CNAME   ernie
monet           IN   A       128.32.7
                IN   A       128.32.130.6
                IN   HINFO   Sun-4/110 UNIX
ourmonet        IN   CNAME   monet
ourfox          IN   A       10.2.0.78
                IN   A       128.32.0.10
                IN   HINFO   Sun-4/110 UNIX
                IN   WKS     128.32.0.10 UDP syslog route timed domain
                IN   WKS     128.32.0.10 TCP ( echo telnet
                             discard rpc sftp
                             uucp-path systat daytime
                             netstat qotd nntp
                             link chargen ftp
                             auth time whois mtp
                             pop rje finger smtp
                             supdup hostnames
                             domain
                             nameserver )
fox             IN   CNAME   ourfox
toybox          IN   A       128.32.131.119
                IN   HINFO   3B2 UNIX
toybox          IN   MX      0  monet.Sample.Edu
```

(continued on next page)

```
miriam          IN  MB     vineyd.Neighbor.COM.
postmistress    IN  MR     miriam
bind            IN  MINFO  bind-request kjd.Sample.Edu.
                IN  MG     ralph.Sample.Edu.
                IN  MG     zhou.Sample.Edu.
                IN  MG     painter.Sample.Edu.
                IN  MG     riggle.Sample.Edu.
                IN  MG     terry.pa.Xerox.Com.
```

The named.local File

The `named.local` file sets up the local loopback interface for your name server. It needs to contain the host name of the machine, plus a pointer to the host name `localhost`, which represents the loopback mechanism. The server name is indicated in the NS resource record, and the pointer to `localhost` by the PTR record. The file also needs to include an SOA record, which indicates the start of a zone and includes the name of the host on which the `named.local` data file reside.

Here is a sample `named.local` file:

```
;    sample named.local file

@   IN  SOA   ourhost.Podunk.Edu. kjd.monet.Podunk.Edu. (
              1          ; Serial
              3600       ; Refresh
              300        ; Retry
              3600000    ; Expire
              3600 )     ; Minimum
    IN  NS    ourhost.Podunk.Edu.
1   IN  PTR   localhost.
```

The hosts.rev File

`hosts.rev` is the file that sets up inverse mapping. It needs to contain the names of the primary and master name servers in your local domain, plus pointers to those servers and to other, non-authoritative name servers. The names of the primary and secondary master servers are indicated by NS records, and the pointers by PTR records. The file also needs an SOA record to indicate the start of a zone and the name of the host on which `hosts.rev` resides.

Here is a sample `hosts.rev` file:

```
;     sample hosts.rev file
@        IN   SOA   ourhost.Podunk.Edu. kjd.monet.Podunk.Edu. (
                    1.1        ; Serial
                    3600       ; Refresh
                    300        ; Retry
                    3600000    ; Expire
                    3600 )     ; Minimum
         IN   NS    ourarpa.Podunk.Edu.
         IN   NS    ourhost.Podunk.Edu.
4.0      IN   PTR   ourarpa.Podunk.Edu.
6.0      IN   PTR   ernie.Podunk.Edu.
7.0      IN   PTR   monet.Podunk.Edu.
10.0     IN   PTR   ourhost.Podunk.Edu.
6.130    IN   PTR   monet.Podunk.Edu.
```

The named.ca File

The `named.ca` file contains the names and addresses of the root servers. Server names are indicated in the record NS and addresses in the record A. You need to add an NS record and an A record for each root server you want to include in the file.

The following is a sample named.ca file:

```
;
;Initial cache data for root domain servers.
;
; list of servers...
.                   99999999    IN      NS    NS.NASA.GOV.
                    99999999    IN      NS    NIC.DDN.MIL.
                    99999999    IN      NS     A.ISI.EDU.
                    99999999    IN      NS    TERP.UMD.EDU.
                    99999999    IN      NS    C.NYSER.NET.
                    99999999    IN      NS    BRL-AOS.ARPA.
                    99999999    IN      NS    GUNTER-ADAM.ARPA.

; ...and their addresses
NIC.DDN.MIL.         99999999    IN      A    10.0.0.51
NIC.DDN.MIL.        99999999    IN      A    26.0.0.73
C.NYSER.NET.         99999999    IN      A    192.33.4.12
BRL-AOS.ARPA.        99999999    IN      A    128.20.1.2
BRL-AOS.ARPA.        99999999    IN      A    192.5.22.82
NS.NASA.GOV.         99999999    IN      A    128.102.16.10
TERP.UMD.EDU.        99999999    IN      A    10.1.0.17
A.ISI.EDU.           99999999    IN      A    26.3.0.103
GUNTER-ADAM.ARPA.    99999999    IN      A    26.1.0.13
```

Standard Resource Record Format

In the Standard Resource Record Format, each line of a data file is a record called a Resource Record (RR), containing the following fields separated by white space:

{name}   {ttl}   class   Record Type   Record Specific data

The order of the fields is always the same; however, the first two are optional (as indicated by the braces), and the contents of the last vary according to the Record Type field.

name	The first field is the name of the domain that applies to the record. If this field is left blank in a given RR, it defaults to the name of the previous RR.

A domain name in a zone file can be either a fully qualified name, terminated with a dot, or a relative one, in which case the current domain is appended to it.

`ttl`	The second field is an optional time-to-live field. This specifies how long (in seconds) this data will be cached in the database before it is disregarded and new information is requested from a server. By leaving this field blank, the ttl defaults to the minimum time specified in the Start Of Authority resource record discussed below. If the ttl value is set too low, the server will incur in a lot of repeat requests for data refreshment; if, on the other hand, the ttl value is set too high, changes in the information will not be timely distributed. Most ttl's should be initially set to between a day (86400) and a week (604800); then, depending on the frequency of actual change of the information, change the appropriate ttl's to reflect that frequency. Also, if you have some ttl's that have very high values because you know they relate to data that rarely changes, and you know that the data is now about to change, reset the ttl to a low value (3600 to 86400) until the change takes place, and then change it back to the original high value. All RR's with the same name, class and type should have the same ttl.
`class`	The third field is the record class. Only one class is currently in use: IN for the TCP/IP Internet protocol family.
`type`	The fourth field states the type of the resource record. There are many types of RR's; the most commonly used types are discussed below, in the section "Resource Record Types."
RR data	The contents of the data field depend on the type of the particular Resource Record.

Although case is preserved in names and data fields when loaded into the name server, all comparisons and lookups in the name server database are case insensitive. However, this situation may change in the future, and you are advised to be consistent in your use of lower and upper case.

Special Characters

The following characters have special meanings:

.	A free standing dot in the name field refers to the current domain.
@	A free standing @ in the name field denotes the current origin.
..	Two free standing dots represent the null domain name of the root when used in the name field.
\X	Where X is any character other than a digit (0-9), quotes that character so that its special meaning does not apply. For example, you can use \. to place a dot character in a label.
\DDD	Where each D is a digit, this is the octet corresponding to the decimal number described by DDD. The resulting octet is assumed to be text and is not checked for special meaning.
( )	Use parentheses to group data that crosses a line. In effect, line terminations are not recognized within parentheses.
;	Semicolon starts a comment; the remainder of the line is ignored.
*	An asterisk signifies wildcarding.

Most resource records have the current origin appended to names if they are not terminated by a "." This is useful for appending the current domain name to the data, such as machine names, but may cause problems where you do not want this to happen. A good rule of thumb is to use a fully qualified name ending in a period if the name is not in the domain for which you are creating the data file.

Control Entries

The only lines that do not conform to the standard RR format in a data file are control entry lines. There are two kinds of control entries:

`$INCLUDE`	An include line begins with `$INCLUDE` in column 1, and is followed by a file name. This feature is particularly useful for separating different types of data into multiple files, for example:

```
$INCLUDE /etc/named/data/mailboxes
```

The line is interpreted as a request to load the file `/etc/named/data/mailboxes` at that point. The `$INCLUDE` command does not cause data to be loaded into a different zone or tree. This is simply a way to allow data for a given zone to be organized in separate files. For example, mailbox data might be kept separately from host data using this mechanism.

`$ORIGIN` The origin command is a way of changing the origin in a data file. The line starts in column 1, and is followed by a domain name. It resets the current origin for relative domain names (i.e. not fully qualified names) to the stated name. This is useful for putting more than one domain in a data file.

Resource Record Types

The following are some of the most commonly used types of RR's:

Type	Description
SOA	Start of Authority
NS	Name Server
A	Internet Address
CNAME	Canonical Name (nickname)
HINFO	Host Information
WKS	Well Known Services
PTR	Pointer
MX	Mail Exchanger

The following is an example of a hosts file. It is presented here for illustration purposes only. We will shortly analize it in full.

```
;    sample hosts file
@              IN   SOA    ourfox.Sample.Edu. kjd.monet.Sample.Edu. (
                           1.1        ; Serial
                           10800      ; Refresh
                           1800       ; Retry
                           3600000    ; Expire
                           86400 )    ; Minimum
               IN   NS     ourarpa.Sample.Edu.
               IN   NS     ourfox.Sample.Edu.
ourarpa        IN   A      128.32.0.4
               IN   A      10.0.0.78
               IN   HINFO  3B2 UNIX
arpa           IN   CNAME  ourarpa
ernie          IN   A      128.32.0.6
               IN   HINFO  3B2 UNIX
ourernie       IN   CNAME  ernie
monet          IN   A      128.32.7
               IN   A      128.32.130.6
               IN   HINFO  Sun-4/110 UNIX
ourmonet       IN   CNAME  monet
ourfox         IN   A      10.2.0.78
               IN   A      128.32.0.10
               IN   HINFO  Sun-4/110 UNIX
               IN   WKS    128.32.0.10 UDP syslog route timed domain
               IN   WKS    128.32.0.10 TCP ( echo telnet
                           discard rpc sftp
                           uucp-path systat daytime
                           netstat qotd nntp
                           link chargen ftp
                           auth time whois mtp
                           pop rje finger smtp
                           supdup hostnames
                           domain
                           nameserver )
fox            IN   CNAME  ourfox
toybox         IN   A      128.32.131.119
               IN   HINFO  3B2 UNIX
toybox         IN   MX     0  monet.Sample.Edu
miriam         IN   MB     vineyd.Neighbor.COM.
postmistress   IN   MR     miriam
bind           IN   MINFO  bind-request kjd.Sample.Edu.
               IN   MG     ralph.Sample.Edu.
               IN   MG     zhou.Sample.Edu.
               IN   MG     painter.Sample.Edu.
               IN   MG     riggle.Sample.Edu.
               IN   MG     terry.pa.Xerox.Com.
```

SOA - Start Of Authority The following is the format of a Start of Authority resource record:

```
name   (ttl)   (class)  SOA        origin   person in charge (
                        serial
                        refresh
                        retry
                        expire
                        minimum )
```

The Start of Authority (SOA) record designates the start of a zone. The zone ends at the next SOA record.

`name`	indicates the name of the zone. In the example below, `@` indicates the current zone or origin.
`IN`	is the address class
`SOA`	is the type of this Resource Record.
`Origin`	is the name of the host where this data file resides.
`Person in charge`	is the mailing address for the person responsible for the name server.
`Serial`	is the version number of this data file. You *must* increment this number whenever you make a change to the data: secondary servers use the Serial field to detect whether the data file has been changed since the last time they copied the file from the master server. Note that the name server cannot handle numbers over 9999 after the decimal point.
`Refresh`	indicates how often, in seconds, a secondary name server should check with the primary name server to see if an update is needed.

`Retry`	indicates how long, in seconds, a secondary server is to retry after a failure to check for a refresh.
`Expire`	is the upper limit, in seconds, that a secondary name server is to use the data before it expires for lack of getting a refresh.
`Minimum`	is the default number of seconds to be used for the time to live field on resource records that don't have a ttl specified.

There should only be one SOA record per zone.

The following is a sample SOA resource record:

```
;name   (ttl)   class   SOA      origin                  person in charge
@               IN      SOA      ourfox.Sample.Edu.      kjd.monet.Sample.Edu. (
                        1.1      ;Serial
                        10800    ;Refresh
                        3600     ;Retry
                        432000   ;Expire
                        86400)   ;Minimum
```

NS - Name Server

The following is the format of an NS resource record:

```
(name)   (ttl)   class   NS   Name-server name
```

The Name Server record (NS) lists by name a server responsible for a given domain. The name field lists the domain that is serviced by the listed name server. If no name field is listed, then it defaults to the last name listed. One NS record should exist for each primary and secondary master server for the domain. The following is a sample NS resource record:

```
;{name}   {ttl}   class   NS    Name-server name
                  IN      NS    ourarpa.Sample.Edu.
```

A - Address

The following is the format of an A resource record:

```
{name}    {ttl}   class   A    address
```

The Address record (A) lists the address for a given machine. The name field is the machine name, and the address is the IP address. One A record should exist for each address of the machine (in other words, gateways should be listed twice, once for each address).

```
;{name}   {ttl}   class   A    address
ourarpa           IN      A    128.32.0.4
                  IN      A    10.0.0.78
```

HINFO - Host Information The following is the format of a HINFO resource record:

```
{name}    {ttl}   class   HINFO   Hardware   OS
```

The Host Information resource record (HINFO) contains host specific data. It lists the hardware and operating system that are running at the listed host. If you want to include a space in the machine name or in the entry in the `Hardware` field, you must surround the entry with quotes. The name field specifies the name of the host. If no name is specified it defaults to the last named host. One HINFO record should exist for each host. The following is a sample HINFO resource record:

```
;{name}   {ttl}   class   HINFO   Hardware    OS
                  IN      HINFO   Sun-3/280   UNIX
```

WKS - Well Known Services

The following is the format of a WKS resource record:

```
{name}   {ttl}   class   WKS   address   protocol   list of services
```

The Well Known Services record (WKS) describes the well known services supported by a particular protocol at a specified address. The list of services and port numbers come from the list of services specified in the `services` database. Only one WKS record should exist per protocol per address. The following is an example of a WKS resource record:

```
;{name}   {ttl}   class   WKS   address       protocol   list of services
                  IN      WKS   128.32.0.10   UDP        who route timed domain
                  IN      WKS   128.32.0.10   TCP        (echo telnet
                                                         discard rpc sftp
                                                         uucp-path systat daytime
                                                         netstat qotd nntp
                                                         link chargen ftp
                                                         auth time whots mtp
                                                         pop rje finger smtp
                                                         supdup hostnames
                                                         domain
                                                         nameserver)
```

CNAME - Canonical Name

The format of a CNAME resource record is:

```
nickname   {ttl}   class   CNAME   Canonical name
```

The Canonical Name resource record (CNAME) specifies a nickname for a canonical name. A nickname should be unique. All other resource records should be associated with the canonical name and not with the nickname. Do not create a nickname and then use it in other resource records. Nicknames are particularly useful during a transition period, when a machine's name has changed but you want to permit people using the old name to reach the machine. The following is a sample CNAME resource record:

```
;nickname   {ttl}   class   CNAME   Canonical name
ourmonet            IN      CNAME   monet
```

PTR - Domain Name Pointer

The following is the format for a PTR resource record:

```
special name   {ttl}   class   PTR   real name
```

A Pointer record (PTR) allows special names to point to some other location in the domain. PTR's are used mainly in the IN-ADDR.ARPA records for the translation of an address (the special name) to a real name. PTR names should be unique to the zone. The PTR records below set up reverse pointers for the special IN-ADDR.ARPA domain.

```
;special name            (ttl)  class  PTR  real name
7.0                             IN     PTR  monet.Podunk.Edu.
2.2.18.128.in-addr.arpa         IN     PTR  blah.junk.COM.
```

MX - Mail Exchanger

The following is the format for an MX resource record:

```
name  (ttl)  class  MX  preference value  mailer exchanger
```

The Mail Exchanger (MX) resource records are used to specify a machine that knows how to deliver mail to a domain or machines in a domain. There may be more than one MX resource record for a given name. In the example below, Seismo.CSS.GOV. (note the fully qualified domain name) is a mail gateway that knows how to deliver mail to Munnari.OZ.AU. Other machines on the network cannot deliver mail directly to Munnari. Seismo and Munnari may have a private connection or use a different transport medium. The `preference value` field indicates the order a mailer should follow when there is more than one way to deliver mail to a single machine. The value 0 (zero) indicates the highest preference. If there is more than one MX resource record for the same name, they may or may not have the same preference value.

You can use names with the wildcard asterisk (*) for mail routing with MX records. There are likely to be servers on the network that simply state that any mail to a domain is to be routed through a relay. In the example below, all mail to hosts in domain foo.COM is routed through RELAY.CS.NET. You do this by creating a wildcard resource record, which states that the mail exchanger for *.foo.COM is RELAY.CS.NET. Note that the asterisk will match any host or subdomain of foo.COM, but it will not match foo.COM itself.

```
;name           {ttl}  class  MX  preference value  mailer exchanger
Munnari.OZ.AU.         IN     MX  0                 Seismo.CSS.GOV.
foo.COM.               IN     MX  10                RELAY.CS.NET.
*.foo.COM.             IN     MX  20                RELAY.CS.NET.
```

Modifying the Data Files

When you add or delete a host in one of the data files in the master DNS server, or otherwise modify the data files, you must also change the Serial number in the SOA resource record so the secondary servers modify their data accordingly; you should then inform `named` in the master server that it should re-read the data files and update its internal database, as explained below.

When `named` successfully starts up, it writes its process ID to the file `/etc/named.pid`. To have `named` re-read `named.boot` and reload the database, enter

```
# kill -HUP 'cat /etc/named.pid'
```

Note that all previously cached data is lost, and the caching process starts over again.

Modifying the Startup Script

Once you create the boot and data files that `named` needs, you have to edit the startup script `/etc/inet/rc.inet`, following these steps:

1. Set the host name in the startup script to the full domain name by adding the line

   `hostname=`*hostname*

   For example, on server `widget` you would enter the following line:

   ```
   hostname=widget.junk.COM.
   ```

CAUTION Do not attempt to run named from inetd. This will continuously restart the name server and defeat the purpose of having a cache.

2. Make sure the following line is in the startup script:

```
/usr/sbin/in.named
```

The above line assumes that your boot file is called /etc/named.boot. If you are using a different name, substitute the following lines for the one above and include the name of the boot file. For example, if your boot file is called /etc/named.init, this is what should appear in the startup script:

```
if [ -f -a /etc/named.init]; then
   /usr/sbin/in.named [options] -b /etc/named.init & echo \c ' named' >/dev/console
fi
```

Setting Up a Root Server for a Local Network

If you are not connecting your local network to the Internet, you must set up primary and secondary name servers in the root-level domain on the local network. This is so all domains in the network have a consistent authoritative server to which to refer; otherwise, machines may not be able to resolve queries.

Since a single machine can be the primary domain name server for more than one machine, the easiest way to create a root domain name server is to have a server be the name server for all the domains that make up its own domain name. For example, if a server is named x.sub.dom., then it should be designated the primary name server for ".", dom., and sub.dom.

Since the root name server provides an authoritative name server at the root level of the network, all top-level domains should have their name server records (IN NS) defined in the root domain.

NOTE It is strongly recommended that the root domain server name be the primary name server for all top-level domains in the network.

Troubleshooting

From time to time, you may need to debug named. You can do this by sending it signals through the `kill(1)` utility. Depending on the signal received named will change its behavior.

To get a good idea of what named thinks the database is, enter

```
kill -INT `cat /etc/named.pid`
```

Upon receiving this signal, named dumps the current data base and cache to `/var/tmp/named_dump.db`. This should give you an indication of whether the database was loaded correctly.

When named is running incorrectly, you can also look in `/var/adm/messages` and check for any messages logged by `syslog`. For instance, if there is a data file that lists a hostname as a nickname and also has other data under that name instead of the machine's canonical name, you may see a message similar to:

```
May  4 02:35:26 hostname named[4804]: hazy.widget.junk.COM
has CNAME and other data (illegal)
```

Or, if there is a problem with the database, you may see

```
May  1 11:02:33 hostname named[17808]: /etc/named/junk.zone:
line 759: database format error ()
```

To turn on debugging, you can start named with the `-d` option, or you can enter, if named is already running,

```
kill -USR1 `cat /etc/named.pid`
```

Each following USR1 increments the debug level. The output goes to `/var/tmp/named.run`.

To turn off debugging completely, enter

```
    kill -USR2 `cat /etc/named.pid`
```

A Practical Example

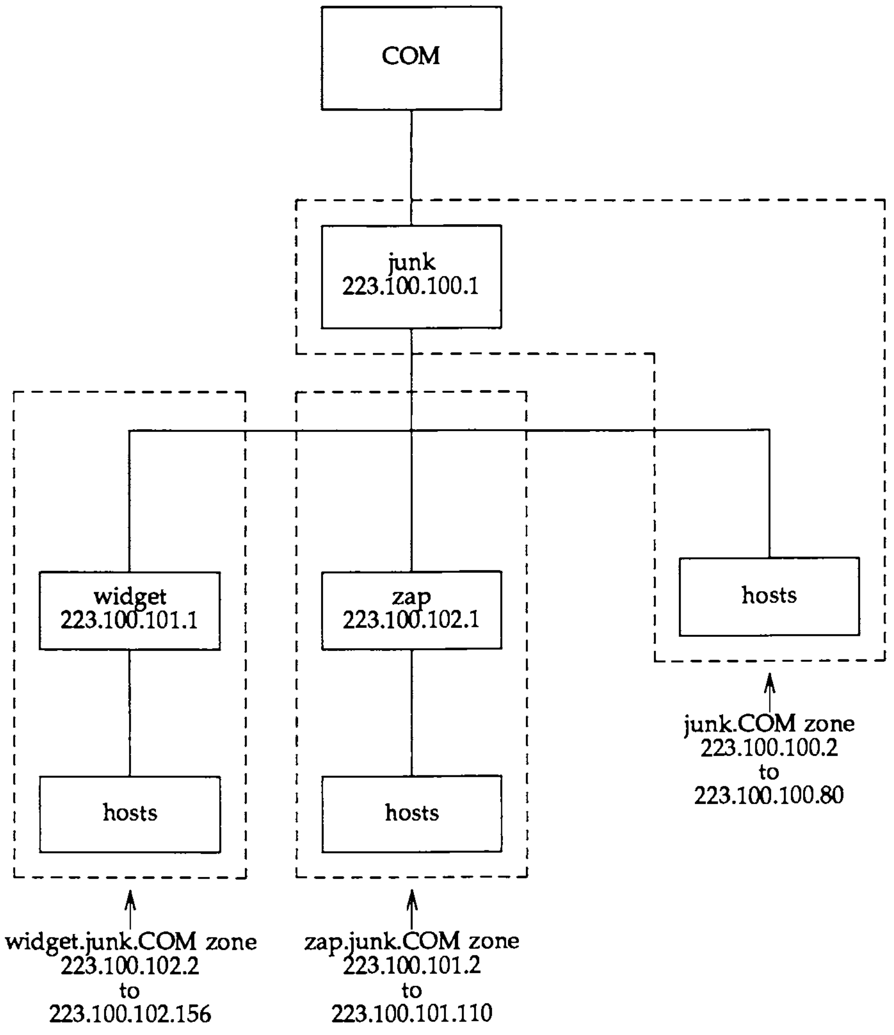
We can now start building up the files that an *imaginary* network would need. Let's assume that our network is composed of three networks, all with access to the Internet. Each network has a Class C Network Number:

```
name  number
junk  223.100.100
widget      223.100.101
zap   223.100.102
```

The names of the zones are also the names of the hosts that we are designating as the master servers.

Our imaginary network can therefore be represented by the following figure:

Figure 5-3: An Imaginary Network



Let's further assume that after careful consideration we decide that we want to set up the Domain Name Service in our network so that each master server is the primary server for its zone and a secondary server for the other zones. All this can be summed up by the following tables:

junk zone		
hostname	function	address
junk	primary	223.100.100.1
widget	secondary	223.100.101.1
zap	secondary	223.100.102.1
	hosts	223.100.100.2-80

widget zone		
hostname	function	address
widget	primary	223.100.101.1
junk	secondary	223.100.100.1
zap	secondary	223.100.102.1
	hosts	223.100.101.2-110

zap zone		
hostname	function	address
zap	primary	223.100.102.1
junk	secondary	223.100.100.1
widget	secondary	223.100.101.1
	hosts	223.100.102.2-156

The following are the boot files for the three servers in the network:

```
;
;          Boot file for server junk
directory  /var/named
cache      .                          named.root
primary    junk.COM                   junk.zone
primary    100.100.223.in-addr.arpa   junk.revzone
primary    0.0.127.in-addr.arpa       named.local

secondary  widget.junk.COM            223.100.101.1 223.100.102.1 widget.zone
secondary  zap.junk.COM               223.100.101.1 223.100.102.1 zap.zone
secondary  101.100.223.in-addr.arpa   223.100.101.1 widget.rev
secondary  102.100.223.in-addr.arpa   223.100.102.1 zap.rev
```

```
;
;          Boot file for server widget
directory  /var/named
cache      .                          named.root
primary    widget.junk.COM            widget.zone
primary    101.100.223.in-addr.arpa   widget.revzone
primary    0.0.127.in-addr.arpa       named.local

secondary  junk.COM                   223.100.100.1 223.100.102.1 junk.zone
secondary  zap.junk.COM               223.100.100.1 223.100.102.1 zap.zone
secondary  100.100.223.in-addr.arpa   223.100.100.1 junk.rev
secondary  102.100.223.in-addr.arpa   223.100.102.1 zap.rev
```

```
;
;       Boot file for server zap
directory   /var/named
cache       .                          named.root
primary     zap.junk.COM               zap.zone
primary     102.100.223.in-addr.arpa   zap.revzone
primary     0.0.127.in-addr.arpa       named.local

secondary   junk.COM                   223.100.100.1 223.100.102.1 junk.zone
secondary   widget.junk.COM            223.100.100.1 223.100.101.1 widget.zone
secondary   100.100.223.in-addr.arpa   223.100.100.1 junk.rev
secondary   101.100.223.in-addr.arpa   223.100.101.1 widget.rev
```

The following are some sample `resolv.conf` files. Note that if the host in question is not running named the local host address should not be used as a nameserver.

```
;
; resolv.conf file for server junk
;
domain          junk.COM
nameserver      127.0.0.1
nameserver      223.100.101.1
nameserver      223.100.102.1
```

```
;
; resolv.conf file for a host in zone junk not running named
;
domain          junk.COM
nameserver      223.100.100.1
nameserver      223.100.101.1
nameserver      223.100.102.1
```

```
;
; resolv.conf file for a host in zone widget.junk not running named
;
domain              widget.junk.COM
nameserver          223.100.100.1
nameserver          223.100.101.1
nameserver          223.100.102.1
```

```
;
; resolv.conf file for a host in zone zap.junk not running named
;
domain              zap.junk.COM
nameserver          223.100.100.1
nameserver          223.100.101.1
nameserver          223.100.102.1
```

The following are sample `named.local` files:

```
;
; named.local for server junk
;
@      IN   SOA      junk.COM.    ralph.sysad.zap.junk.COM. (
            1.1      ;Serial
            10800    ;Refresh
            3600     ;Retry
            432000   ;Expire
            86400)   ;Minimum

       IN   NS       junk.COM.
1      IN   PTR      localhost.
```

```
;
; named.local for server widget
;
@      IN   SOA      widget.junk.COM.   ralph.sysad.zap.junk.COM. (
            1.1      ;Serial
            10800    ;Refresh
            3600     ;Retry
            432000   ;Expire
            86400)   ;Minimum

       IN   NS       widget.junk.COM.
1      IN   PTR      localhost.
```

```
;
; named.local for server zap
;
@      IN   SOA      zap.junk.COM.   ralph.sysad.zap.junk.COM. (
            1.1      ;Serial
            10800    ;Refresh
            3600     ;Retry
            432000   ;Expire
            86400)   ;Minimum

       IN   NS       zap.junk.COM.
1      IN   PTR      localhost.
```

The following is the `hosts` file for server `junk`, followed by its `$INCLUDE`'d file:

```
;
; junk zone hosts file for server junk
;
@                   IN  SOA   junk.COM.         ralph.sysad.zap.junk.COM. (
                        1.1       ;Serial
                        10800     ;Refresh
                        3600      ;Retry
                        432000    ;Expire
                        86400)    ;Minimum

                    IN  NS    junk.COM.
                    IN  NS    widget.junk.COM.
                    IN  NS    zap.junk.COM.
widget.junk.COM.    IN  NS    widget.junk.COM.
                    IN  NS    junk.COM.
                    IN  NS    zap.junk.COM.
zap.junk.COM.       IN  NS    zap.junk.COM.
                    IN  NS    junk.COM.
                    IN  NS    widget.junk.COM.
junk.COM.           IN  MX    10 junk.COM.
*.junk.COM.         IN  MX    10 junk.COM.
; junk.COM hosts
$INCLUDE /var/named/hosts/junk
```

```
; hosts in junk zone as listed in /var/named/hosts/junk
junk            A       223.100.100.1
                A       10.1.0.56
                MX      10 junk.COM.
widget          A       223.100.101.1
                HINFO   "Sun 3/180"     Unix
                MX      10 junk.COM.
                WKS     223.100.101.1   UDP syslog timed domain
                WKS     223.100.101.1   TCP (echo telnet
                                        discard rpc sftp
                                        uucp-path systat daytime netstat
                                        qotd nntp link chargen ftp auth
                                        time whots mtp pop rje finger
                                        smtp supdup hostnames
                                        domain nameserver)
zap             A       223.100.102.1
                HINFO   Sun-4/110       Unix
                MX      10 junk.COM.
                WKS     223.100.102.1   UDP syslog timed domain
                WKS     223.100.102.1   TCP (echo telnet
                                        discard sftp
                                        uucp-path systat daytime netstat
                                        qotd nntp link chargen ftp auth
                                        time whots mtp pop rje finger
                                        smtp supdup hostnames
                                        domain nameserver)
lazy            A       223.100.100.2
                HINFO   "Sun 3/50"      Unix
                MX      10 junk.COM.
crazy           A       223.100.100.3
                HINFO   3B2             Unix
                MX      10 junk.COM.
hazy            A       223.100.100.4
                HINFO   3B2             Unix
                MX      10 junk.COM.
; all other hosts follow, up to 223.100.100.80
```

The following is the `hosts` file for server `widget`, followed by its `$INCLUDE`'d file:

```
;
; widget zone hosts file for server widget
;
@       IN   SOA      widget.junk.COM.   ralph.sysad.zap.junk.COM. (
             1.1      ;Serial
             10800    ;Refresh
             3600     ;Retry
             432000   ;Expire
             86400)   ;Minimum

        IN   NS       widget.junk.COM.
        IN   NS       junk.COM.
        IN   NS       zap.junk.COM.
; junk.COM hosts
$INCLUDE /var/named/hosts/widget
```

```
; hosts in widget zone as listed in /var/named/hosts/widget
widget          A       223.100.101.1
                HINFO   "Sun 3/180"       Unix
                MX      10 junk.COM.
whatsit         CNAME   widget.junk.COM
smelly          A       223.100.101.2
                HINFO   3B2               Unix
                MX      10 junk.COM.
stinky          A       223.100.101.3
                HINFO   3B2               Unix
                MX      10 junk.COM.
dinky           A       223.100.101.4
                HINFO   "Sun 3/160"       Unix
                WKS     223.100.101.4     UDP who route
                WKS     223.100.101.4     TCP (echo telnet
                                          discard sftp
                                          uucp-path systat daytime netstat
                                          ftp finger domain nameserver)
                MX      10 junk.COM.
; all other hosts follow, up to 223.100.101.110
```

The following is the sample file of reverse addresses for hosts in the zone junk. Note that the name of the domain is fully qualified, so that the addresses of the hosts (without the network address) is sufficient in this case:

```
; reverse address file for server junk, in /var/named/junk.revzone

100.100.223.in-addr.arpa.    IN   SOA    junk.COM.         ralph.sysad.zap.junk.COM. (
                                  1.1        ;Serial
                                  10800      ;Refresh
                                  3600       ;Retry
                                  432000     ;Expire
                                  86400)     ;Minimum

                             IN   NS     junk.COM.
1                                 PTR    junk.COM.
2                                 PTR    lazy.junk.COM.
3                                 PTR    crazy.junk.COM.
4                                 PTR    hazy.junk.COM.

; all other hosts follow, up to 223.100.100.80
```

The reverse address files for servers widget and zap should be written in a manner similar to the above.

6. TROUBLESHOOTING

6 Troubleshooting

Troubleshooting Commands

TCP/IP includes several commands that help you troubleshoot, should you have problems with the network. These commands are:

```
ping
ifconfig
netstat
```

This chapter describes these commands and gives suggestions for using them.

The ping Command

The `ping` command offers the simplest way to find out if a host on your network is up or down. Its basic syntax is:

```
/usr/sbin/ping host [ timeout ]
```

where *host* is the hostname of the machine in question. The optional `timeout` argument indicates the time in seconds for `ping` to keep trying to reach the machine—20 seconds by default. The `ping(1M)` manual page describes additional syntaxes and options.

When you run `ping`, the ICMP protocol sends a datagram to the host you specify, asking for a response. (Recall that ICMP is the protocol responsible for error handling on a TCP/IP network.)

Suppose you typed:

```
$ ping elvis
```

If host `elvis` is up, you receive the following message:

```
elvis is alive
```

indicating that `elvis` responded to the ICMP request. However, if host elvis is down or cannot receive the ICMP packets, you receive the following response from `ping`:

```
no answer from elvis
```

If you suspect that a machine may be loosing packets even though it is up, you can use the `-s` option of `ping` to try and detect the problem. For example, suppose you type:

```
ping -s elvis
```

ping then continually sends packets to host elvis until you press the INTERRUPT key or a timeout occurs. The responses on your screen will resemble the following:

```
PING elvis: 56 data bytes
64 bytes from 129.144.50.21: icmp_seq=0. time=80. ms
64 bytes from 129.144.50.21: icmp_seq=1. time=0. ms
64 bytes from 129.144.50.21: icmp_seq=2. time=0. ms
64 bytes from 129.144.50.21: icmp_seq=3. time=0. ms
       .
       .
       .
----elvis PING Statistics----
4 packets transmitted, 4 packets received, 0% packet loss
round-trip (ms)  min/avg/max = 0/20/80
```

The statistical message appears after you type the interrupt, or timeout is reached. The packet loss statistic indicates whether the host has dropped packets.

If ping fails, check the status of the network reported by ifconfig, and netstat (described in the following sections).

The ifconfig Command

The ifconfig command displays information about the configuration of an interface that you specify. (Refer to the manual entry for ifconfig(1M) for complete information regarding this command.)

The syntax of ifconfig is:

ifconfig *interface* [*protocol_family*]

where *interface* indicates the interface that you want information about (such as the Ethernet interface).

For example, if you type the following

```
$ ifconfig emd1
```

you may receive the following output:

```
emd1: flags=63<UP,BROADCAST,NOTRAILERS,RUNNING>
        inet 129.144.50.28 netmask ffffff00 broadcast 129.144.50.0
```

NOTE This example shows the output for `ifconfig` when used on an Ethernet interface. When using `ifconfig` on a different type of interface, the characteristics of that interface are displayed, and therefore different output is seen.

This tells you several things about the Ethernet interface. First, the flags section shows that the interface is up, broadcasting to the network, not using "trailer" link level encapsulation, and that the interface is running with no problems. Information included on the second line are the internet address of the host you are using, the netmask being currently used, and the address of the network that broadcasts are sent over.

If you get output that indicates an interface is not running, it might mean a problem with that interface. In this case, refer to the `ifconfig`(1M) manual page.

The netstat Command

The `netstat` command generates displays that show network status. You can run `netstat` to display the status of network traffic in table format, including routing table information (available routes and their status), and interface information. The `netstat`(1M) manual page provides full details.

`netstat` can display, according to the command line options chosen, one of the various network data structures available. These displays are the most useful for system administration. The syntax for this form is:

`netstat` [`-n`] [`-s`] [`-i` | `-r`] [`-f` *address_family*]

The options you might use most frequently to determine network status are `-s`, `-r`, `-i`, and `-rs` used together.

Displaying Per Protocol Statistics

The -s option displays per-protocol statistics for the UDP, TCP, ICMP, and IP protocols. When you run `netstat -s`, the result is a display resembling the following:

```
ip:
        227201 total packets received
        0 bad header checksums
        0 with size smaller than minimum
        0 with data size < data length
        0 with header length < data size
        0 with data length < header length
        12 fragments received
        0 fragments dropped (dup or out of space)
        0 fragments dropped after timeout
        3086 packets forwarded
        0 packets not forwardable
        0 redirects sent
icmp:
        0 calls to icmp_error
        0 errors not generated 'cuz old message was icmp
        Output histogram:
                address mask reply: 3
        .
        .
        .
        Input histogram:
                address mask request: 3
        3 message responses generated
tcp:
        connections initiated: 7
        connections accepted: 59
        connections established: 66
        connections dropped: 5
        embryonic connections dropped: 9
                   .
        .
        .
        packets rcvd after "close": 1
        rcvd window probe packets: 0
        rcvd duplicate acks: 90
        rcvd acks for unsent data: 0
        rcvd ack packets: 11230
        bytes acked by rcvd acks: 901832
        rcvd window update packets: 95
```

(continued on next page)

```
udp:
        0 incomplete headers
        0 bad data length fields
        0 bad checksums
```

The statistical information can indicate areas where a protocol is having problems. For example, statistical information from ICMP can indicate where this protocol has found errors.

Displaying Communications Controller Status

The `-i` option of `netstat` shows the state of the communications controllers that are configured with the machine where you ran the command. Here is a sample display produced by `netstat -i`.

```
Name  Mtu   Network      Address       Ipkts    Ierrs Opkts     Oerrs Collis
emd1   1500  mtnview-en  speed          14093893 8492  10174659 1119  2314178
emd2   1500  mtnview-en  speed-ebb      9299762 54423 12451748   0     775125
```

Using this display, you can find out how many packets a machine thinks it is transmitting and receiving on each network. For example, the input packet count (Ipkts) displayed for a server may increase each time a client tries to boot, while the output packet count (Opkts) remains steady. This suggests that the server is seeing the boot request packets from the client, but does not realize it is supposed to respond to them. This might be caused by an incorrect address in the `hosts` or `ethers` database.

On the other hand, if the input packet count is steady over time, it means that the machine does not see the packets at all. This suggests a different type of failure, possibly a hardware problem.

Displaying Routing Table Status

The `-r` option of `netstat` displays the IP routing table. Here is a sample display produced by `netstat -r` run on machine `ballet`.

```
Routing tables
Destination    Gateway    Flags  Refcnt Use    Interface
temp8milptp    elvis      UGH     0      0       xxn
irmcpeb1-ptp0  elvis      UGH     0      0       xxn
route93-ptp0   speed      UGH     0      0       xxn
mtvb9-ptp0     speed      UGH     0      0       xxn
                 .
                 .
                 .
mtnside        speed      UG      1    567       xxn
ray-net        speed      UG      0      0       xxn
mtnside-eng    speed      UG      0     36       xxn
mtnside-eng    speed      UG      0    558       xxn
mtnside-eng    ballet     U      33 190248       xxn
```

The first column shows the destination network, the second the router through which packets are forwarded. The U flag indicates that the route is up; the G flag indicates that the route is to a gateway. The H flag indicates that the destination is a fully qualified host address, rather than a network.

The Refcnt column shows the number of active uses per route, and the Use column shows the number of packets set per route. Finally, the Interface column shows the network interface that the route uses.

Displaying Routing Statistics

Combining the options `-rs` with `netstat` produces routing statistics. You can use the resulting display to determine if your network is having routing problems. Here is sample output from `netstat -rs`

```
routing:
        0 bad routing redirects
        0 dynamically created routes
        0 new gateways due to redirects
        2 destinations found unreachable
        2330 uses of a wildcard route
```

If the output indicates that bad routing redirects occurred or that a number of destinations were found unreachable, this is indicative of problems on your network.

Running Software Checks

If there is trouble on the network, here are some actions you can take to diagnose and fix software-related problems.

1. Use the `netstat` command to determine network status as described previously.
2. Check the `hosts` database to make sure that the entries are correct and up-to-date.
3. If you are running RARP, check the Ethernet addresses in the `ethers` database to make sure that the entries are correct and up to date.
4. Check the accuracy of the `ifconfig` line(s) in the startup scripts for your machine.

   An `ifconfig` line should look like the following:

   `ifconfig` emd*x* *hostname* `up`

   If your machine has more than one network interface, your startup scripts will have more than one `ifconfig` line.
5. Try to `rlogin` to yourself.
6. Make sure the network daemon `inetd` is running. Log in as superuser and type the following:

   ```
   ps -ef | grep inetd
   ```

   The resulting display should resemble the following if the `inetd` daemon is running.

   ```
   root    57     1  0  Apr 04 ?       3:19 inetd
   root  4218  4198  0 17:57:23 pts/3  0:00 grep inetd
   ```

Restarting TCP/IP

If you need to stop and restart TCP/IP, use the following procedure.

1. To stop TCP/IP, do the following:
   a. If you are running RFS, stop RFS by issuing the `rfstop` command.
   b. Stop other processes using TPC/IP.
   c. Kill `inetd`and the TCP/IP listener by entering the commands
      ```
      sacadm -k -p inetd
      sacadm -k -p tcp
      ```
   d. Kill `slink` by entering
      ```
      sh -x /etc/init.d/inetinit stop
      ```
2. To restart TCP/IP, do the following:
   a. In extreme cases, you may need to reset the controller board. To do so, type
      ```
      epump -d /dev/emdx
      ```
      where *x* is the slot number of the board.
   b. Start `slink` by entering the command
      ```
      sh -x /etc/init.d/inetinit start
      ```
   c. Restart `inetd` and the listener by entering the commands
      ```
      sacadm -s -p inetd
      sacadm -s -p tcp
      ```
   d. Restart RFS with the `rfstart` command.

Improving System Performance

If you are having performance problems, you may be running the daemons `routed` and `rwhod`. The `rwho` daemon (`rwhod`) is not run by default because it has a severe impact on performance. You are strongly advised not to run it.

The routing daemon `routed` runs on every machine by default. If you have limited memory and there are no routers on your network, then you do not need to run `routed`. You can disable the daemon by commenting out the line

```
/usr/sbin/in.routed
```

in the startup script `/etc/inet/rc.inet`.

If your machine has limited memory *and* only one router in your network, then you can run `routed` only on the router, and disable it on all other machines.

For more information about configuring a router, see "Expanding Your Network" earlier in this guide.

Logging Network Problems

If you suspect a routing daemon malfunction, you may log its actions—and even all the packet transfers. To create a log file of routing daemon actions, just supply a file name when you start up the `routed` daemon, for example:

```
# /usr/sbin/in.routed /var/routerlog
```

(Refer to the `routed`(1M) manual page for more information about `routed`.)

CAUTION On a busy network this generates almost constant output.

Whenever a route is added, deleted, or modified, a log of the action and a history of the previous packets sent and received will be printed in the log file. To force full packet tracing, specify the `-t` option when the daemon is started up.

APPENDIX A: THE IP NUMBER REGISTRATION FORM

APPENDIX A: THE IP NUMBER REGISTRATION FORM

A Guidelines for Completing the IP Number Registration Form

Guidelines for Completing the IP Number Registration Form

The following questions appear on the IP Number Registration Form provided by the Hostmaster at SRI-NIC. You can request the form via electronic mail from HOSTMASTER@SRI-NIC.ARPA, or, if electronic mail is not available to you, write to:

> DDN Network Information Center
> SRI International
> Room EJ217
> 333 Ravenswood Avenue
> Menlo Park, CA 94025

Answer the questions according to the guidelines presented here, and return the form to the same address.

NOTE If the network will NOT be connected to either the DARPA Internet or the DDN Internet, then you do not need to provide this information.

1. If the network will be connected to the DARPA Internet or the DDN Internet, you must provide the name of the sponsoring organization, and the name, title, mailing address, phone number, net mailbox, and NIC Handle (if any) of the contact person (POC) at that organization who has authorized the network connection. This person will serve as the POC for administrative and policy questions about authorization to be a part of the DARPA Internet or the DDN Internet. Examples of such sponsoring organizations are: Defense Communications Agency (DCA), Defense Advanced Research Projects Agency (DARPA), the National Science Foundation (NSF), or similar military or government sponsors.

Example:

Sponsor

Organization	DARPA
Name	Lastname, Firstname
Title	Program Manager
Mail Address	DARPA/ISTO Office 1400 Wilson Boulevard Arlington, VA 22209
Phone Number	(XXX) XXX-XXX
Net Mailbox	progmgr@VAX.DARPA.MIL
NIC Handle	AA12

2. Provide the name, title, mailing address, phone number, and organization of the administrative POC for the network requesting the number. This is the POC for administrative and policy questions about the network itself. If the network is associated with a research project this POC should be the Principal Investigator of the project. The online mailbox and NIC Handle (if any) of this person should also be included.

Example:

Administrator

Organization	SRI International Network Information Center
Name	Lastname, Firstname
Title	Principal Investigator
Mail Address	SRI International 333 Ravenswood Avenue Menlo Park, CA 94025
Phone Number	(XXX) XXX-XXXX
Net Mailbox	pi@SRI-NIC.ARPA
NIC Handle	BB34

3. Provide the name, title, mailing address, phone number, and organization of the technical POC. The online mailbox and NIC Handle (if any) of the technical POC should also be included. This is the POC for resolving technical problems associated with the network and for updating information about the network. The technical POC may also be responsible for hosts attached to this network.

   Example:

   Technical POC

Organization	SRI International Network Information Center
Name	Lastname, Firstname
Title	Computer Scientist
Mail Address	SRI International 333 Ravenswood Avenue Menlo Park, CA 94025
Phone Number	(XXX)XXX-XXXX
Net Mailbox	cs@SRI-NIC.ARPA
NIC Handle	CC56

4. Supply the short mnemonic name for the network (up to 12 characters). This is the name that will be used as an identifier in internet name and address tables.

   Example:

   ALPHA-BETA

5. Supply the descriptive name of the network (up to 20 characters). This name might be used to clarify the ownership, location, or purpose of the network.

   Example:

   Greek Alphabet Net

6. Identify the network geographic location.

   Example:

   SRI International
   Network Information Center
   333 Ravenswood Avenue
   Menlo Park, CA 94025

7. Provide a citation to a document that describes the technical aspects of the network. If the document is online, give a pathname suitable for online retrieval.

   Example:

   "The Ethernet, a Local Area Network: Data Link Layer and Physical Layer Specification," X3T51/80-50 Xerox, Stamford Connecticut, October 1980.

   NOTE For networks connected to the DARPA Internet or the DDN Internet the gateway must be either a core gateway supplied and operated by BBN, or a gateway of another Autonomous System. If this gateway is not a core gateway, then an identifiable gateway in this gateway's Autonomous System must exchange routing information with a known core gateway via EGP.

8. Gateway information required:

   If the network is to be connected to the DARPA Internet or the DDN Internet, answer questions 8a and 8b.

   a. Describe the Gateway that connects the new network to the DARPA Internet or the DDN Internet, and the date it will be operational. The gateway must be either a core gateway supplied and operated by BBN, or a gateway of another Autonomous System. If this gateway is not a core gateway, then an identifiable gateway in this gateway's Autonomous System must exchange routing information with a known core gateway via EGP.

      A good way to answer this question is to say "Our gateway is supplied by person or company X and does whatever their standard issue gateway does".

Example:

Our gateway is the standard issue supplied and operated by BBN, and will be installed and made operational on 1-April-83.

b. Describe the gateway machine, including:

(a) Hardware (LSI-11/23, VAX-11/750, etc. interfaces)

(b) Addresses (what host on what net for each connected net)

(c) Software (operating system and programming language)

Example:

(a) Hardware

PDP-11/40, ARPANET Interface by ACC, Ethernet Interfaces by 3COM.

(b) Address

10.9.0.193 on ARPANET

(c) Software

Berkeley Unix 4.2 BSD and C

9. Estimate the number of hosts that will be on the network:

(a) Initially,

(b) Within one year,

(c) Within two years

(d) Within five years.

Example:

(a) initially = 5

(b) one year = 25

(c) two years = 50

(d) five years = 200

10. Unless a strong and convincing reason is presented, the network (if it qualifies at all) will be assigned a class C network number. If a class C network number is not acceptable for your purposes state why. (Note: If there are plans for more than a few local networks, and more than 100 hosts, you are strongly urged to consider subnetting. [See RFC 950])

    Example:

    Class C is fine.

11. Networks are characterized as being either Research, Defense, Government - Non Defense, or Commercial, and the network address space is shared between these three areas. Which type is this network?

    Example:

    Research

12. What is the purpose of the network?

    Example:

    To economically connect computers used in DARPA sponsored research project FROB-BRAF to the DARPA Internet or the DDN Internet to provide communication capability with other similar projects at UNIV-X and CORP-Y.

For further information contact the DDN/ARPANET Network Information Center (NIC):

Via electronic mail:	HOSTMASTER@SRI-NIC.ARPA
Via telephone:	(800) 235-3155
Via postal mail:	SRI International DDN Network Information Center 333 Ravenswood Avenue EJ217 Menlo Park, CA 94025

APPENDIX B: THE DOMAIN REGISTRATION FORM

APPENDIX B: THE DOMAIN REGISTRATION FORM

B Guidelines for Completing the Domain Registration Form

Guidelines for Completing the Domain Registration Form

To establish a domain, the following information must be sent to the NIC Domain Registrar (`HOSTMASTER@SRI-NIC.ARPA`). Questions may be addressed to the NIC Hostmaster by electronic mail at the above address, or by phone at (415) 859-3695 or (800) 235-3155.

NOTE The key people must have electronic mailboxes and NIC "handles," unique NIC database identifiers. If you have access to "WHOIS," please check to see if you are registered and if so, make sure the information is current. Include only your handle and any changes (if any) that need to be made in your entry. If you do not have access to WHOIS, please provide all the information indicated and a NIC handle will be assigned.

1. The name of the top-level domain to join.

   Example:

   COM

2. The NIC handle of the administrative head of the organization. Alternately, the person's name, title, mailing address, phone number, organization, and network mailbox. This is the contact point for administrative and policy questions about the domain. In the case of a research project, this should be the principal investigator.

   Example:

   Administrator

Organization	The NetWorthy Corporation
Name	Penelope Q. Sassafrass
Title	President
Mail Address	The NetWorthy Corporation 4676 Andrews Way, Suite 100 Santa Clara, CA 94302-1212
Phone Number	(415) 123-4567
Net Mailbox	Sassafrass@ECHO.TNC.COM
NIC Handle	PQS

3. The NIC handle of the technical contact for the domain. Alternately, the person's name, title, mailing address, phone number, organization, and network mailbox. This is the contact point for problems concerning the domain or zone, as well as for updating information about the domain or zone.

   Example:

   Technical and Zone Contact

Organization	The NetWorthy Corporation
Name	Ansel A. Aardvark
Title	Executive Director
Mail Address	The NetWorthy Corporation 4676 Andrews Way, Suite 100 Santa Clara, CA. 94302-1212
Phone Number	(415) 123-6789
Net Mailbox	Aardvark@ECHO.TNC.COM
NIC Handle	AAA2

4. The name of the domain (up to 12 characters). This is the name that will be used in tables and lists associating the domain with the domain server addresses. [While, from a technical standpoint, domain names can be quite long (programmers beware), shorter names are easier for people to cope with.]

   Example:

   TNC

5. A description of the servers that provide the domain service for translating names to addresses for hosts in this domain, and the date they will be operational.

   Example:

   Our server is a copy of the one operated by the NIC; it will be installed and made operational on 1 November 1987.

6. Domains must provide at least two independent servers for the domain. Establishing the servers in physically separate locations and on different PSNs is strongly recommended. A description of the primary and secondary server machines, including
   - Host domain name and network addresses
   - Any domain-style nicknames (please limit your domain-style nickname request, if any, to one)
   - Hardware and software, using keywords from the Assigned Numbers RFC.

   The preferred format for this information is:

   Primary Server: HOST-DOMAIN-NAME, NETADDRESS, HARDWARE, SOFTWARE

   Secondary Server: HOST-DOMAIN-NAME, NETADDRESS, HARDWARE, SOFTWARE

   Example:

   Primary Server: BAR.FOO.COM, 10.9.0.13, VAX-11/750, UNIX

   Secondary Server: XYZ.ABC.COM, 128.4.2.1, IBM-PC, MS-DOS

7. Planned mapping of names of any other network hosts (including any ARPANET or MILNET hosts), other than the server machines, into the new domain's naming space.

   Example:

   ```
   BAR-FOO2.ARPA (10.8.0.193) -> FOO2.BAR.COM
   BAR-FOO3.ARPA (10.7.0.193) -> FOO3.BAR.COM
   BAR-FOO4.ARPA (10.6.0.193) -> FOO4.BAR.COM
   ```

8. An estimate of the number of hosts that will be in the domain.

   (a) Initially

   (b) Within one year

(c) Two years

(d) Five years.

Example:

(a) Initially = 50

(b) One year = 100

(c) Two years = 200

(d) Five years = 500

NOTE Registration of a domain does not imply an automatic name change to previously registered ARPANET or MILNET hosts that will be included in the domain. Please list below the official host names and network addresses of any ARPANET or MILNET hosts that now appear in HOSTS.TXT whose names will change as a result of this domain registration. (Also be sure to answer question # 7, above.)

9. The date you expect the fully qualified domain name to become the official host name in HOSTS.TXT, if applicable.
10. Please describe your organization briefly.

    Example:

    The NetWorthy Corporation is a consulting organization of people working with UNIX and the C language in an electronic networking environment. It sponsors two technical conferences annually and distributes a bimonthly newsletter.

TCP/IP GLOSSARY

Glossary

address	A unique number that identifies a machine on a network (see *IP address*).
ARP	Address Resolution Protocol, a protocol that maps an IP address to its corresponding Ethernet address.
ARPANET	The Advanced Research Projects Agency funded network, for which TCP/IP was originally developed (see *DoD Internet*).
bridge	A device used at the Data Link layer that selectively copies packets between networks of the same type.
caching-only server	A domain name server that is not authoritative for any domain. This server queries servers who have authority for the information neeeded and caches that data.
DARPA Internet	The Defense Advanced Research Projects Agency Internet (see *Internet*).
datagram	Transmission unit at the IP level.
DDN Internet	The Defense Data Network Internet (see *Internet*).
DoD Internet	The Department of Defense Internet, a wide area network to which the ARPANET belongs (see *Internet*).
domain	An administrative entity that provides management for a network.
Domain Name Service	The name service of the Internet Protocol family.
EGP	Exterior Gateway Protocol, a specialized protocol that allows exchange of information with a backbone under a separate administration.
gateway	An IP router.
header	Information attached to the beginning of data. Headers usually contain information about the following data to aid in processing it.

host	An individual machine on a network.
ICMP	Internet Control Message Protocol, which is responsible for handling errors and printing error messages.
Internet	A wide area network originally funded by the Department of Defense, which utilizes TCP/IP for data interchange. The term *Internet* is used to refer to any and all of ARPANET, DARPANET, DDN, or DoD Internets.
internetwork	A group of networks connected by routers.
IP	Internet Protocol, which allows host-to-host datagram delivery.
IP **address**	A unique number that identifies each host in a network.
IP **network number**	A unique number that identifies each IP network (See *net number*).
net number	A number that NIC assigns to your network. The net number forms the first part of a host's IP address.
network mask	A number used by software to separate the local subnet address from the rest of a given IP address.
network protocols	Sets of rules that explain how software and hardware should interact within a network to transmit information.
NIC	Network Information Center, a service run by SRI that administers IP network numbers and domain names.
NIC **handle**	A unique NIC database identifier assigned to a network's administrator and technical contact.
packet switching	A concept which states that a network transmits packets over connections that last only for the duration of the transmission.
packets	A piece of data smaller than a datagram. A datagram is made up of one or more packets.

port numbers	Numbers used by UDP and TCP to identify the end points of communication.
primary master server	The primary domain name server for a zone, which maintains all the data corresponding to its domain.
RARP	Reverse Address Resolution Protocol, a protocol that is the reverse of ARP. RARP maps Ethernet addresses to its corresponding IP addresses.
remote server	A machine, which is not the local one, acting as a server for information.
repeater	A machine that indiscriminately transmits data from one segment of a network to another, used at the physical layer (see *bridge*).
router	A device that forwards packets of a protocol family, from one network to another (also called a *gateway*).
secondary master server	A backup server that takes over for the primary master server, should it become overloaded or inoperable.
SRI	SRI International, a not-for-profit organization that runs the NIC.
SRI-NIC	See *NIC*.
subnet	An administrative division of a network into smaller networks.
subnet number	The part of an IP address that refers to the specific subnet desired.
TCP	Transmission Control Protocol.
TCP/IP	A term used to refer to the entire Internet family of protocols, consisting of the names of the two most important protocols.

UDP	User Datagram Protocol, a protocol at the same layer as TCP, but without acknowledgment of transmission and therefore unreliable.
virtual circuits	An apparent connection between processes which is facilitated by TCP. A virtual circuit allows applications to talk to each other as if they had a physical circuit.
zones	Administrative boundaries within a domain, often made up of one or more sub-domains (see *domain*).

TCP/IP INDEX

Index

DISTRIBUTED FILE SYSTEM ADMINISTRATION GUIDE

Contents

7. INTRODUCTION TO DFS

7 Introduction to Distributed File System Administration

About the DFS Administration Guide

UNIX System V Release 4.0 provides you with two distributed file system packages, both of which have been implemented as file system types—Remote File Sharing (RFS) and Network File System (NFS). Both packages allow you to access resources residing on remote machines on a network, and to make resources on your machine available to remote machines over a network. You may choose to run one or the other package, depending on the specific needs of your users, or you may want to run both packages, so that you can take advantage of the different services each package provides. (For a comparison of RFS and NFS, see the introduction to the *Network User's and Administrator's Guide.*)

This guide describes the tasks you can perform using Distributed File System Administration—a set of commands and files that allows you to administer RFS and NFS (as well as other distributed file system types that may be available in the future) in a consistent way. The files and commands that support common administration of RFS and NFS are provided in the Distributed File System (DFS) Administration Utilities package, which is delivered with System V Release 4.0 software.

This guide is not intended to be a complete reference for administering either RFS or NFS. Although there are a number of tasks common to the administration of both packages, there are many differences as well. Therefore, you must see *The Remote File Sharing Administrator's Guide* for a complete discussion of RFS administration, and *The Network File System Administrator's Guide* for a discussion of NFS administration. Both guides are packaged in this volume.

Audience

The DFS Administration Guide is intended only for administrators who are administering both RFS and NFS. If you are running only one distributed file system package, all the information you need to administer that package is in the package-specific administration guide.

This guide is directed to administrators who are experienced administrators of stand-alone systems. It is assumed that you understand such basic concepts as

- changing run levels (init states)
- assigning user ID (uid) numbers

- mounting and unmounting local resources
- assigning access rights to local resources

It is assumed, too, that you are familiar with the basic concepts of RFS and NFS, including

- the server/client model
- domains

and that you know enough about the benefits of RFS and NFS to make the decision to install both packages.

For information about basic system administration, see the *System Administrator's Guide*. For information about RFS and NFS, see the *Remote File Sharing Administrator's Guide* and the *Network File System Administrator's Guide*.

Organization

This guide is organized by the tasks you can perform using DFS Administration commands and files.

In addition to the command line interface, a menu interface is provided through which you can enter DFS commands. The menus are included with System Administration Menus (sysadm), a menu-based administration interface that is standard with System V Release 4.0. Once you access a sysadm menu, help screens provide you with all the information you need to complete a task. Therefore, this guide does not lead you step-by-step through the menu-based procedures. Instead, it documents DFS commands as you would enter them at the command line, then, in an appendix, describes the DFS menus and directs you to the *System Administrator's Guide* for instruction in using the menu interface.

The organization of the guide is as follows:

- "Introduction to Distributed File System Administration" describes this guide, presents an overview of DFS Administration, and describes all the commands and files that the Distributed File System Administration Utilities package either installs or utilizes.

- "Setting Up DFS Administration," describes the software that must be installed before you can use DFS Administration and directs you to installation instructions.
- "Using DFS Administration" tells you how to share and unshare RFS and NFS resources using DFS commands and files; how to mount and unmount remote resources; and to display information about resources that are shared and mounted on your local system and on network clients.
- Appendix, "Using Distributed File System Management Menus," describes the menu interface to DFS Administration.

Basic Terminology

This guide uses terminology that should be known to administrators familiar with basic system administration and basic RFS and NFS concepts, including:

client	A system that uses the resources of another system; a system can be both a *client*, utilizing resources that reside on other systems, and a *server*, making local resources available to other systems.
file system type	An implementation of a file system to support a particular file type. (The characteristics of a file are determined by its file type.) RFS and NFS are implemented as file systems that support files of a type whose characteristics make them suitable for sharing across a network.
mount point	A location within a directory tree through which your computer accesses mounted file systems. You need to create and/or specify a mount point when you mount a remote resource. Any empty directory can serve as a mount point. (A resource can be mounted on a directory that is not empty, but the mount then obscures the directory's contents.)
resource	An object, such as a directory or a file system, that is made available to users of a computer system. An RFS resource can be a directory and everything below the directory in the directory tree, including ordinary files, subdirectories, named pipes, and special files. An NFS resource can be a

	whole or partial file system and includes all the ordinary files and directories below the root of the shared resource that are part of the same file system.
server	A system that provides resources to another system; a system can be both a *server*, making local resources available to other systems, and a *client*, utilizing resources that reside on other systems.
sharing	Making a local resource available to remote systems.
unsharing	Making a shared local resource unavailable to remote systems.

Conventions

This guide follows the standard conventions of the System V Release 4.0 document set. These standard conventions include the following:

- Command and directory names appear in `constant width` type.
- Text that you enter from your terminal is shown in `constant width` type.
- Text that is displayed on your terminal by the computer is shown in `constant width` type.
- The syntax of a command is presented in the following format:

  share [**-F** *fstype*] [**-o** *specific_options*]

  - An option in italics indicates that you must supply a specific option of the general type; for example, in place of *fstype* you enter either `rfs` or `nfs`.
  - Bold type indicates that the option must be typed exactly as it appears.
  - Square brackets indicate that the option is not required to execute the command.

- Braces indicate that one or the other of two options must be entered to execute the command; for example, the syntax of the umount command is

  ```
  umount {resource | mountpoint}
  ```

  where the braces indicate that you must enter either the name of the resource you want to unmount, or the name of the mountpoint where the resource is mounted.

An Overview of DFS Administration

Some tasks involved in the administration of a distributed file system package are the same, whether you are using RFS or NFS. The Virtual File System architecture introduced in System V Release 4.0 provides a mechanism for administering file system types in a general, or generic, way. What this means for administrators of RFS and NFS is that a set of commands is provided with which you can administer both RFS and NFS (as well as any distributed file system type that may be supported in System V in the future).

For example, to share a resource on your computer with remote systems, you must make the resource available and communicate its availability. To make a resource available using earlier versions of RFS, you would "advertise" it, using the `adv` command. To make a local resource available running earlier versions of NFS, you would "export" it, using the `exportfs` command. If you were to run both distributed file system packages, keeping the corresponding package-specific commands straight might be difficult. Multiply this by the number of corresponding package-specific commands in both packages, and the job becomes even more difficult.

DFS Administration provides generic commands that call package-specific commands, freeing you of the need to learn two sets of corresponding commands. For example, DFS Administration provides you with the `share` command, which lets you both advertise a resource over RFS and export a resource over NFS.

DFS Administration commands and files allow you to share and unshare local resources, mount and unmount remote resources, and display information about shared and mounted resources. Resources can be shared or mounted *automatically*, by adding commands to a file so that the sharing or mounting is done whenever you enter run level 3; and resources can be shared and mounted *explicitly*, by typing commands at the command line.

DFS Commands and Files

The DFS Administration Utilities package installs six commands: `share`, `unshare`, `shareall`, `unshareall`, `dfshares`, and `dfmounts`. In addition to the commands it installs, DFS Administration utilizes a number of files and commands installed by the Release 4.0 Essential Utilities or created by scripts or commands.

All the files and commands relevant to DFS administration are described in this section. Files and commands that operate on all file system types are described in this section only as they relate to administering distributed file systems.

The DFS Administration files are:

- `/etc/dfs/fstypes`, which registers the distributed file system packages you have installed on your system and sets one as the default. The `fstypes` file is created by the distributed file system package you install first.
- `/etc/dfs/dfstab`, which allows you to share a resource or a set of resources automatically when you enter run level 3.
- `/etc/vfstab`, which allows you to mount resources automatically when you enter run level 3.
- `/etc/dfs/sharetab`, which logs the resources currently shared on your system. The `sharetab` file is created by the `share` command and does not require handling by an administrator.
- `/etc/mnttab`, which logs the file systems currently mounted by your system, including remote file systems and directories. The `mnttab` file is created by the `mount` command and does not require handling by an administrator.

DFS Administration commands are:

- `share`(1M), which allows you to make a resource available for mounting by clients, or to display a list of the resources on your system that are currently shared.
- `unshare`(1M), which allows you to make a previously available resource unavailable for mounting by clients.
- `shareall`(1M), which executes a script that shares a pre-determined set of resources.
- `unshareall`(1M), which executes a script that unshares all currently shared resources.
- `mount`(1M), which allows you to mount a remote resource on your system, or to display a list of resources, both local and remote, that are currently mounted on your system.

- `umount`(1M), which allows you to remove a remote resource you previously mounted.
- `mountall`(1M), which executes a script that mounts a pre-determined set of resources.
- `umountall`(1M), which executes a script that unmounts all currently mounted resources.
- `dfshares`(1M), which displays a list of remote resources that are available to you, as well as a list of local resources that are currently shared.
- `dfmounts`(1M), which shows you which local resources are mounted by which clients.

Command Syntax

All DFS Administration commands and the generic commands used by DFS Administration have the following syntax:

command [`-F` *fstype*] [`-o` *specific_options*] [*additional_options*] [*operands*]

The options and operands are described below:

`-F` *fstype*	which indicates the specific file system type on which the command is to operate.
`-o` *specific options*	which indicates options specific to the file system type on which the command is to operate.
operands	which indicates non-option arguments to the command.

8. SETTING UP DFS

8 Setting Up DFS Administration

Introduction

This chapter describes the tasks you must complete before you execute DFS Administration commands. These tasks are installing the software, possibly editing a file to set up a default file sharing package, and starting distributed file system operation.

Installing the Software

DFS Administration Utilities are packaged on floppy diskettes and distributed with UNIX System V Release 4.0 software. If you installed all the software when you received Release 4.0, DFS Administration is already installed on your system. If you did not install all the add-on utilities with Release 4.0, you must install DFS in addition to RFS and NFS Utilities, and all the hardware and software required to run RFS and NFS. DFS Utilities must be installed *before* you install RFS and NFS.

Installation instructions for all UNIX System V Release 4.0 software—including DFS Administration Utilities, RFS Utilities, and NFS Utilities—appear in the Release 4.0 *Release Notes*. Hardware and software prerequisites for RFS and NFS appear in the *Remote File Sharing Administrator's Guide* and the *Network File System Administrator's Guide*, respectively.

Once all your network hardware and software are installed, you must set up your file sharing packages before you can enter DFS commands. The setup procedures for RFS and NFS differ; for RFS setup procedures, see the *Remote File Sharing Administrator's Guide*; for NFS setup procedures, see the *Network File System Administrator's Guide.*

Changing the Package Default—the /etc/dfs/fstypes File

So that the system can distinguish between local and remote file system types, the distributed file system packages you have installed must be "registered." When you install either RFS or NFS, the installation script creates the `fstypes` file and populates it. If the file already exists when you install either package, the installation script adds a line to the file. RFS adds

```
rfs Remote File Sharing Utilities: Version m
```

and NFS adds

```
nfs Network File System Utilities: Version m
```

where `m` indicates the version number of the package installed.

The package indicated in the first line of the file becomes your default package. If you enter DFS Administration commands without specifying a package (or file system type), the system assumes the first entry in the `/etc/dfs/fstypes` file. Therefore, if you know that you will administer one package more frequently than the other, make that the first entry in the file. This saves you from specifying the package every time you enter a command.

To change the default package, edit the file using any supported text editor.

Starting Distributed File System Operation

Both RFS and NFS normally become operational automatically whenever you take your system to run level 3; however, both can be started in other run levels by entering commands at the command line.

You use different procedures and commands to start and stop RFS and NFS operations manually. For information, see the RFS and NFS administration guides packaged in this volume.

To start RFS and NFS automatically, use the `init` command to take your system to run level 3. If you choose, you can have your system enter run level 3 automatically when you boot by changing the `initdefault` line in the `/sbin/inittab` file to read: `is:3:initdefault:`

As explained earlier, DFS Administration allows you to share and mount resources *explicitly*, by entering commands at the command line, and *automatically*, by editing files that share and mount resources when you take the system to run level 3. Commands can be entered explicitly in any run level once you start distributed file system operation. Automatic sharing and mounting is done *only* when your system enters run level 3. When you exit run level 3, any resource you shared is automatically unshared, and any remote resource you mounted is automatically unmounted.

9. USING DFS COMMANDS AND FILES

9 Using DFS Administration Commands and Files

Sharing and Unsharing Resources

This section tells you how to share and unshare RFS and NFS resources in a consistent way, using the same commands and files. The commands and files described in this chapter are the `share`, `unshare`, `shareall`, and `unshareall` commands, and the `/etc/dfs/dfstab` file.

Before you unshare a resource, using either the `unshare` or `unshareall` command, be sure you understand the effect of unsharing on existing mounts. When you unshare an RFS resource, existing mounts are unaffected; clients with the resource mounted can continue to use the resource, but no new mounts are permitted. When you unshare an NFS resource, access to existing mounts is inhibited; for more information, see the *Network File System Administrator's Guide*.

Sharing a Resource Explicitly—the share Command

To make a resource on your computer available to clients, you use the `share` command, which the following syntax:

```
share [-F fstype] [-o specific_options] [-d description]
      [pathname [resourcename]]
```

Options and operands are described below:

`-F` *fstype*	which indicates the distributed file system type to be administered—either `rfs` or `nfs`.
`-o` *specific_options*	which is a list of file-system-type-specific options that can follow the -o flag; if no option is specified, then the system assumes the `rw` option, and sharing is done read/write to all clients (see below for more information).
`-d` *description*	which is a description enclosed in quotation marks that you supply to describe an RFS or NFS resource to clients; descriptions cannot include special characters and cannot exceed 32 characters.

NOTE: If you enter the `-d` option when you share an NFS resource, the description is stored in your `sharetab` file. However, clients will not see the description displayed when they use the `dfshares` command to list the resources shared on your system.

pathname	which indicates the pathname of the resource to be shared; pathnames cannot exceed 64 characters.
resourcename	which is a name you give an RFS resource when you share it; clients then use the name to access the resource once it is shared; resource names cannot exceed 14 characters. NFS does not support this operand.

When sharing a resource over RFS or NFS, the following suboptions can follow the -o flag:

`rw`	which indicates that sharing is to be done read/write to all clients.
`ro`	which indicates that sharing is to be done read-only to all clients.
`rw=`*client*[:*client*]...	which indicates that sharing shall be done read/write to the listed clients; when sharing over NFS, this suboption can be used in conjunction with the `ro` or `ro=` suboption to share the resource read/write to some clients and read-only to others.
`ro=`*client*[:*client*]...	which indicates that sharing shall be done read-only to the listed clients; when sharing over NFS, this suboption can be used in conjunction with the `rw` or `rw=` suboption to share the resource read/write to some clients and read-only to others.

NOTE: When sharing over RFS, only one of the suboptions `rw`, `ro`, `rw=`, and `ro=` can be specified.

The following suboptions can follow the -o flag only when sharing over NFS.

anon=*uid*	which indicates that *uid* will be the effective user ID of root on an unknown host. If this suboption is not given, unknown users are given the effective ID –2.
root=*host*[*:host*]...	which indicates that root users from the specified hosts only shall have root access. If this suboption is not given, then no host is granted root access.
secure	which indicates that clients must use the AUTH_DES authentication of RPC. If this suboption is not given, then AUTH_U authentication must be used.

NOTE The command fails if the same client is given ambiguous access privileges, such as when the same client name is included in both an `rw=` list and an `ro=` list, or when both the `rw` option and the `ro` option are given without arguments.

(For more information about NFS-specific options, see the *Network File System Administrator's Guide.*)

If no argument is specified when the share command is entered, then the command displays all resources on your system that are currently shared. If only a file system type is specified, then the share command displays all resources of the specified type that are currently shared. The share command can be used in this capacity by users as well as administrators.

Example 1

You want to share over RFS a directory called mtgnotes, which is a subdirectory to your /usr/reports directory. RFS is your default file sharing package. You want to share the directory by the name FORUM. You want to share the directory read-only to all clients. At the command line, type

```
share -o ro -d "oct 9 materials" /usr/reports/mtgnotes FORUM
```

Example 2

You want to share a partial file system on your computer. The root directory of the branch of the file system is graphics, which is a subdirectory to your /export directory. The directory is an NFS resource. You want to share the directory read-only to all clients except a client named "art.dept," with which you want to share the directory read/write. At the command line, type

```
        share -F nfs -o ro,rw=art.dept /export/graphics
```

Sharing a Resource Automatically—the /etc/dfs/dfstab File

The `etc/dfs/dfstab` file allows you to share a set of resources automatically whenever your system enters run level 3. For example, if you want a directory to be available to clients on a regular basis, and you can anticipate few occasions when you would need to make it unavailable, you can enter a `share` command for that directory into the `dfstab` file. Then, whenever you take the system to run level 3, the directory becomes available to clients automatically.

Each line of the file consists of the `share` command line needed to `share` a particular resource; the `share` command you enter in the file has the same syntax as the `share` command you enter at the command line. (See the description of the `share` command in the preceding section.)

If you want to add or delete a resource from a list of shareable resources, or to modify the way the sharing is done, edit the file with your text editor. The next time you enter run level 3 from another run level, the changes you made to the file will take effect.

Example

You want the directory `editors` to be available to all clients at all times. The directory is an NFS resource, located in your `/export` directory. Use your text editor to add the `share` command to the `/etc/dfs/dfstab` file. The file entry should look like this:

```
        share -F nfs /export/editors
```

Unsharing a Resource—the unshare Command

If you wish to reclassify a resource you have shared so that it is no longer available for mounting by remote systems, you "unshare" it. To unshare a resource, you use the `unshare` command.

The `unshare` command can be used to unshare any resource—whether the resource was shared explicitly with the `share` command or automatically through the `dfstab` file. If you use the `unshare` command to unshare a resource that you shared through the `dfstab` file, remember that it will be shared again when you exit and re-enter run level 3. The syntax for the `unshare` command is:

```
unshare [-F fstype] [-o specific_options] {pathname | resourcename}
```

Options and operands are described below.

`-F` *fstype*	which indicates the distributed file system type to be administered—either `rfs` or `nfs`.
`-o` *specific_options*	which indicates a file-system-type-specific option. (There are no RFS- or NFS-specific options at this time; see the "note" below.)
pathname	which indicates the pathname of the resource to be unshared.
resourcename	which is a name you give an RFS resource when you share it.

When unsharing an RFS resource, you supply either a pathname or a resource name, not both. When unsharing an NFS resource, you specify a pathname only.

NOTE The `unshare` command supports file-system-type-specific options so that file system types developed in the future can offer package-specific functionality. At this time, RFS and NFS do not supply specific options to the `unshare` command.

Example 1

You want to unshare a directory called invdata.88. The directory is an RFS resource, which you shared by the name OLDSTATS. RFS is your default file sharing package. Type

```
        unshare OLDSTATS
```

Example 2

Although the directory /export/templates on your machine is shared continually through the dfstab file, you need to unshare the directory temporarily. The directory is an NFS resource. Type

```
        unshare -F nfs /export/templates
```

Now you can share the directory immediately using the share command, or the directory will be shared again automatically when you exit and re-enter run level 3.

Sharing a Set of Resources—the shareall Command

DFS Administration lets you share a set of resources, including a combination of RFS and NFS resources, by entering a single command—the shareall command. To use the command, you first create a file that lists the resources you want to share. The syntax of the entries in your file is the same syntax as the share command and the entries in the dfstab file, as follows:

```
        share [-F fstype] [-o specific_options] [-d description]
              [pathname [resourcename]]
```

Once you create the file, you specify it as the input file when you enter the shareall command.

If you do not specify an input file, the /etc/dfs/dfstab file is used by default. If you enter a dash (–) in place of the name of an input file, the system accepts standard input, which means you can enter a number of share commands in succession, then execute the commands all at once by pressing <CTRL-d>. This saves you from entering one share command, waiting for the system to execute the command and return your prompt, entering another command, and so on.

The syntax of the `shareall` command is as follows:

```
shareall [-F fstype[,fstype...]] [- | file]
```

Options and operands are described below.

`-F` *fstype[,fstype...]*	which indicates that the resources of the specified file system type with entries in the input file are to be shared. If no `-F` option is given, then all resources with entries in the input file are shared.
-	which indicates that the command should accept standard input.
file	which is the name of the file you created to be your input file.

Example 1

You create an input file called `misc` that contains commands to share three separate resources. The file looks like this:

```
#cat misc
share -F rfs -o ro /usr/reports/mtg.notes FORUM
share -F nfs -o ro,rw=art.dept /export/graphics
share -F nfs /usr/man
```

To share all the resources listed in the file you created, type

```
shareall misc
```

To share only the NFS resources listed in the `misc` file, type

```
shareall -F nfs misc
```

Example 2

You want to share three separate resources. Because you will not be sharing the same resources as a set on a regular basis, you do not want to create an input file. Type

```
shareall -
```

Your cursor moves to a new line. Enter the following commands, pressing <Return> after you enter each command.

```
share -F rfs -o ro /usr/reports/mtg.notes FORUM
share -F nfs -o ro,rw=art.dept /export/graphics
share -F nfs /usr/man
```

To execute the commands once they are entered, press <CTRL-d>.

Unsharing a Set of Resources—the unshareall Command

DFS Administration provides you with the `unshareall` command, which lets you unshare all the shared resources on your system, or all the shared resources of a specified file system type. The syntax of the `unshareall` command is

`unshareall [-F` *fstype*`[,`*fstype...*`]]`

where the `-F` option indicates the file system type of the resources to be unshared (either `rfs` or `nfs`).

When you specify a file system type, the command unshares all local resources of the distributed file system type you specified. If no `-F` option is specified, then the command unshares all local resources currently shared.

Example 1

You want to unshare all the currently shared RFS resources on your computer. Type

```
unshareall -F rfs
```

Example 2

You want to unshare all currently shared local resources, regardless of file system type. Type

```
unshareall
```

Mounting and Unmounting Remote Resources

This section tells you how to mount and unmount remote RFS and NFS resources explicitly, using the generic `mount`, `umount`, `mountall`, and `umountall` commands. It also tells you how to mount remote resources automatically, using the `/etc/vfstab` file. The commands and files described in this section are described only as they relate to the administration of remote resources.

Mounting a Remote Resource Explicitly—the mount Command

The `mount` command allows you to mount both local and remote file systems. To mount a remote resource using the `mount` command, you must be able to reach, via a network, the server that is sharing the resource you want to mount.

The syntax of the `mount` command as it relates to mounting distributed file systems is:

```
mount [-F fstype] [-o specific-options] special   [mountpoint]
```

Options and operands are described below.

`-F` *fstype*	which is the file system type of the resource you want to mount, either `rfs` or `nfs`. If the `-F` option is not specified, and *special* and *mountpoint* are, then `mount` looks in `/etc/vfstab` for a corresponding entry and mounts the resource according to the file system type specified there.
`-o` *specific options*	which is a list of file-system-type-specific options that can be specified after the `-o` flag. There are different options for RFS and NFS (described later in this section).
special	where, if the resource is an RFS resource, *special* is the resource name (the name the resource was assigned when it was shared), and, if the resource is an NFS resource, *special* is the name of the server sharing the resource the client wants to mount, followed by a colon, then the pathname of the resource to be mounted.

mountpoint	which indicates that the resource should be mounted on the specified mountpoint; if no mount point is specified, but the resource is listed in `/etc/vfstab`, the command takes the mount point from the `vfstab` file.

The options that can follow the `-o` flag when mounting either an RFS or an NFS resource are:

`rw` | `ro`	where `rw` indicates that the resource is to be mounted read/write and `ro` indicates it is to be mounted read-only. (If no option is specified, the resource is mounted read/write by default—as long as the resource was shared read/write. If no option is specified and the resource was share read-only, the command fails.)
`suid` | `nosuid`	where `suid` indicates that set-uid bits are to be obeyed on execution and `nosuid` indicates that they are to be ignored. (If no option is specified, set-uid bits are obeyed by default.) Every directory mounted `rw` is a good candidate for `nosuid`.

RFS also supplies the option *nocaching* to the `-o` flag; *nocaching* indicates that client caching is be to disabled.

NOTE

The options `-c`, `-d`, and `-r` are valid options when mounting an RFS resource. The `mount` command continues to support the `-c`, `-d`, and `-r` options for compatibility with previous releases of RFS.

Some NFS-specific options that can follow the `-o` flag are:

[bg | fg]	where `bg` indicates that a retry should be initiated in the background when the server does not respond, and `fg` indicates it should be initiated in the foreground. If no option is specified, the retry is initiated in the foreground by default. (This option does not apply to the Automounter.)
`intr`	which indicates that keyboard interrupts should be allowed while NFS requests are being issued. If interrupts are not

	allowed on a hard mount the terminal will hang until the request succeeds. This suboption affects all subsequent NFS requests. (For more information, see the *Network File System Administrator's Guide.*)
`retry=`*n*	which indicates the number of times to retry the mount operation. The default for *n* is 10,000 times.
`timeo=`*n*	which sets the timeout to *n* tenths of a second. If no option is specified, the timeout is set to 11 seconds by default.

For more information about NFS-specific options, see the *Network File System Administrator's Guide.*

Resources mounted explicitly with the `mount` command stay mounted during a work session unless you unmount them, using the `umount` command. If you exit run level 3 and re-enter it, the resource is no longer mounted (unless you edited the `vfstab` file to include the mount automatically).

Example 1

You want to mount an RFS resource called `BLUEPRINTS`. You have created a mount point called `/usr/old.blues`. To mount the remote directory, type

```
mount -F rfs BLUEPRINTS /usr/old.blues
```

Because no access rights are specified, the directory is mounted read/write.

Example 2

You want to mount an NFS resource residing on a server called "tools.srv." The resource is a directory called `graphics` in the server's `/usr` directory. You created a mount point called `graphics` in your `/usr/tools` directory. You want to mount the resource read/write, and you want set-uid bits to be ignored. Type

```
mount -F nfs -o tools.srv:/usr/graphics /usr/tools/graphics
```

Because no access rights are specified, the directory is mounted read/write.

Example 3

You want to hard mount a directory containing manuals pages that resides on a server called "docgroup." You want the directory mounted read-only, with interrupt enabled. The directory is an NFS resource named `/usr/man`. You want to mount the directory to a mount point of the same name on your machine. Type

```
mount -F nfs -o ro,intr docgroup:/usr/man /usr/man
```

Mounting a Remote Resource Automatically—the /etc/vfstab File

The `/etc/vfstab` file allows you to mount a local file system automatically when you boot the system. If you edit the file to include a remote file system or directory, the remote resource is mounted automatically when you take the system to run level 3.

To mount a remote resource automatically, create a mount point for the resource using the `mkdir` command, then edit the `vfstab` file. Entries in the `vfstab` file require the following syntax:

special fsckdev mountp fstype fsckpass automnt mntopts

The fields are described below:

special	where, if the resource is an RFS resource, *special* is the resource name (the name the resource was assigned when it was shared), and, if the resource is an NFS resource, *special* is the name of the server sharing the resource the client wants to mount, followed by a colon, then the name of the resource to be mounted.
fsckdev	which is the name of the raw device to `fsck`; for a remote mount, the parameter is not applicable, and the entry should be - .
mountp	which is the mount point on which the resource is to be mounted.

fstype	which is the file system type of the resource you want to mount.
fsckpass	which is the pass number to use for multiple `fsck`'s. For a remote mount, the parameter is not applicable, and the entry should be - .
automnt	which indicates whether the entry should be automounted by `/sbin/mountall` (`yes`) or not (`no`) when the client is booted or enters the appropriate run level.
mountopts	which is a list of comma-separated `-o` sub-options identical to the options passed to the `mount` command; any of the general or RFS-specific and NFS-specific `-o` options supported by the `mount` command can be entered into the `vfstab` file. (See the preceding section for a description of the options that can follow the `-o` flag.)

Once you create the mount point and edit the `vfstab` file, the file system or directory will be mounted automatically when you enter run level 3. It will be mounted automatically every time you enter run level 3 until you modify the `vfstab` file (unless, of course, the server sharing the directory unshares it). The contents of `/etc/vfstab` remain the same until you edit the file.

Example

You want to mount the resource `/usr/share` from a server named "frontoffice." You want the resource to be mounted automatically every time you take your system to run level 3. The directory is an NFS resource. You want to mount the directory on the mount point `/usr/local/tmp` with read-only access. First create the mount point by typing

```
mkdir /usr/local/tmp
```

Make sure the mode and permissions of the new mount point match those of the resource you want to mount on it. Then use the `cd` command to move to your `/etc` directory and edit the `vfstab` file, using any supported text editor. The file entry should look like this:

```
frontoffice:/usr/share - /usr/local/tmp nfs - yes ro
```

Unmounting a Remote Resource—the umount Command

The `umount` command allows you to unmount both local and remote file systems. You can unmount a resource with `umount` whether is was mounted explicitly using the `mount` command or automatically through the `vfstab` file.

The syntax of the `umount` command, as it relates to unmounting distributed file systems, is:

```
umount {resource | mountpoint}
```

The options are described below:

resource	where, if the resource is an RFS resource, *resource* is the resource name (the name the resource was assigned when it was shared), and, if the resource is an NFS resource, *resource* is the name of the server sharing the resource, followed by a colon, then the pathname of the resource.
mountpoint	which is the pathname of the mount point on the client where the resource is mounted.

Example 1

You want to unmount a remote resource called `BLUEPRINTS`, which you mounted on a mountpoint called `old.blues` in your `/etc` directory. The resource is an RFS resource. Type

```
umount /etc/old.blues
```

Example 2

You want to unmount a remote NFS resource by specifying the server sharing the resource and the pathname of the resource on the server. Type

```
umount docgroup:/usr/man
```

Mounting a Set of Resources—the mountall Command

DFS Administration lets you mount a set of resources, including a combination of remote RFS and NFS resources, by entering a single command—the `mountall` command. To use the command, you first create a file that lists the resources you want to mount. The syntax of the entries in your file is the same syntax as the entries in the `/etc/vfstab` file, as follows:

special fsckdev mountp fstype fsckpass automnt mntopts

(For an explanation of the fields, see the section "Mounting a Remote Resource Automatically," which describes the `vfstab` file.)

Once you create a file, you specify it as the input file when you enter the `mountall` command.

If you do not specify an input file, the `/etc/vfstab` file is used by default. If you enter a dash (–) in place of the name of an input file, the system accepts standard input, which means you can enter a number of `mount` commands in succession, then execute the commands all at once by pressing <CTRL-d>. This saves you from entering one `mount` command, waiting for the system to execute the command and return your prompt, entering another command, and so on.

The syntax of the `mountall` command is as follows:

```
mountall [-F fstype] [-l | -r] [- | file]
```

Options and operands are described below.

`-F` *fstype*	which indicates that the resources of the specified file system type with entries in the input file are to be mounted. If no `-F` option is given, then all resources with entries in the input file are mounted.
`-l`	which indicates that only local file systems listed in the input file are to be mounted.
`-r`	which indicates that only remote resources listed in the input file are to be mounted.

–	which indicates that the command should accept standard input.
file	which is the name of the file you created to be your input file.

If the `mountall` command is entered without arguments, it mounts all local file systems and remote resources in the `vfstab` file that have the *automnt* field set to "yes."

Example 1

You want to mount a set of remote resources. You create an input file called `mntlist` that lists three remote resources. The file looks like this:

```
#cat mntlist
docgroup:/usr/man - /usr/man nfs - yes ro
BLUEPRINTS - /usr/old.blues rfs - yes rw
frontoffice:/usr/share - /usr/local/tmp nfs - yes rw
```

To mount all the resources listed in the file you created, type

```
mountall mntlist
```

To mount only the NFS resources listed in the `mntlist` file, type

```
mountall -F nfs mntlist
```

Example 2

Your system mounts resources automatically when you entered run level 3, using the `/etc/vfstab` file. While resources are still mounted, you add entries to the `vfstab` file. Now you want to mount the resources you added to the file, without exiting run level 3 and unmounting currently mounted resources. Type

```
mountall
```

The system uses the updated `vfstab` file as the input file. It mounts all resources in the file that are not mounted currently and displays error messages for those resources that are mounted already.

Unmounting a Set of Resources—the umountall Command

DFS Administration utilizes the `umountall` command, which lets you unmount all the mounted resources on your system, except the root file system, or all the mounted resources of a specified file system type. The syntax of the `umountall` command is

```
umountall [-F fstype] [-k] [-l | -r]
```

Options and operands are described below:

`-F` *fstype*	which indicates the file system type of the resources to be unmounted. When you specify a file system type, the command unmounts all local and remote resources of the file system type you specified. If no `-F` option is specified, then the command unmounts all local file systems, except the root file system, and all remote resources.
`-k`	which indicates that all processes with files open when the `umountall` command is executed are to be killed.
`-l`	which indicates that only local file systems are to be unmounted.
`-r`	which indicates that only remote resources are to be unmounted.

Example 1

You want to unmount all the remote RFS resources currently mounted on your system. Type

```
umountall -F rfs
```

Example 2

You want to unmount all file systems currently mounted on your system, except the root file system. You want to kill all processes that have open files. Type

```
umountall -k
```

Displaying Information

This section tells you how to determine what resources are available to you from network servers, what resources are shared and mounted on your system, and what resources shared on your system have been mounted by network clients. The commands described in this chapter are the `share`, `mount`, `dfshares`, and `dfmounts` commands.

Displaying Shared Local Resources—the share Command

You use the `share` command to share resources on your system with network clients; however, you can also use the command to display information about shared resources on your system. The command's syntax when used in this capacity is:

```
        share [-F fstype]
```

If you enter the command without arguments, it displays all RFS and NFS resources on your system that are currently shared. If you specify a file system type and no other options, the command displays all resources of the specified type that are currently shared.

Displaying Mounted Resources—the mount Command

Generally, you use the `mount` command to mount local and remote resources on your system. However, if you enter the command without arguments, it displays a list of local and remote resources that are currently mounted on your system.

NOTE You cannot limit the display to remote resources or to resources of a specific file system type by entering arguments. If you enter `mount -F rfs`, for example, the system displays an error message.

Browsing Shared Remote Resources—the dfshares Command

To "browse" shared resources means to determine what resources are available from remote systems. For browsing, DFS Administration gives you the `dfshares` command, which displays information about the RFS and NFS resources that are available from network servers. The syntax of the `dfshares` command is:

`dfshares [-F` *fstype*`] [-h] [-o` *specific_options*`] [`*server ...*`]`

The arguments and options are described below:

`-F` *fstype*	which displays available resources of the distributed file system type specified—either `rfs` or `nfs`.
`-h`	which indicates that a header should not be printed. If this option is not given, then a header is printed by default.
`-o` *specific_options*	which indicates file-system-type-specific options; however, there are no RFS- or NFS-specific options at this time.
server ...	where *server* is a list of server names separated by commas. A server name indicates that information about the resources shared by the named server is to be displayed.

The syntax of the *server* option depends on the server's file system type. If you are browsing for shared RFS resources, the *server* field can be any name that is supported by the RFS command `nsquery`. You can use a *system* name, which specifies a system in the domain of the local system, a *domain* name, which specifies *all* the systems in the domain, or you can enter both the domain and the system, in the form *domain.system*, to display resources on a specific remote system in a remote domain.

To display resources on an NFS server, you can enter a *system* name for the server option, which specifies a system on the network.

When you enter the command, your system displays output consisting of the fields

resource server access transport description

where *resource* is the resource name that is given to the `mount` command; *server* is the system from which the resource is available; *access* refers to the access granted client systems—either read/write or read-only; *transport* is the transport provider over which communication is carried, such as STARLAN; and *description* is a description of the resource provided by an administrator when the resource was shared.

For NFS resources, the command cannot determine the access rights granted client systems or the transport provider; therefore, the access and transport fields are filled with hyphens (-) when `dfshares` displays a list of NFS resources. The description field does not appear at all in a display of NFS resources.

If you enter the `dfshares` command without arguments, the system displays all resources (both RFS and NFS) currently shared on your system.

Example 1

You want to browse the RFS resources available from a server named "lib" in a domain named "sales." You want the display to include a header. Type

```
dfshares -F rfs sales.lib
```

The system displays

```
RESOURCE       SERVER    ACCESS  TRANSPORT  DESCRIPTION

sales.CONV89   lib       ro      starlan    "material from 1989 sales convention"
sales.PROJ1    lib       ro      starlan    "1990 sales projections by product"
sales.PROJ2    lib       ro      starlan    "1990 sales projections by territory"
sales.TRAIN    lib       ro      starlan    "sales force training materials"
```

Example 2

You want to browse NFS resources on a server called "main." You do not want the display to include a header. Type

```
dfshares -F nfs -h main
```

The system displays

```
main:/export        main    -    -
main:/usr/man       main    -    -
main:/usr/share     main    -    -
```

Monitoring the Use of Local Resources by Remote Systems—the dfmounts Command

The `dfmounts` command shows you which local resources are mounted by clients. The command accepts the following syntax:

`dfmounts` [`-F` *fstypes*] [`-h`] [`-o` *specific_options*] [*restriction*]

Options and operands are described below:

`-F` *fstype*	which displays mounted resources of the distributed file system type specified—either `rfs` or `nfs`.
`-h`	which indicates that a header should not be printed. If this option is not given, then a header is printed by default.
`-o` *specific_options*	which indicates file-system-type-specific options; however, RFS and NFS do not provide specific options at this time.
restriction	where *restriction* is an NFS *server* or an RFS *resource*. When you include a restriction, `dfmounts` displays information only about the server or resource you specified.

If you enter `dfmounts` without arguments, the system displays all local resources (both RFS and NFS) currently mounted by remote systems.

The `dfmounts` command displays a header, which is optional, followed by lines consisting of the fields

resource server pathname clients ...

where *resource* is the resource name that must be given to the `mount` command; *server* is the system from which the resource is available; *pathname* is the pathname of the shared resource as it was given to the `share` command; and *clients* is a list (separated by commas) of clients that have mounted the resource.

Example 1

Your machine is called "lib"; it resides in domain "sales." You want to display the RFS resources on your system that are currently mounted by clients. RFS is the only file sharing package installed on your system. You want the display to include a header. Type

```
dfmounts
```

The system displays

```
RESOURCE    SERVER     PATHNAME              CLIENTS

CONV89      lib    /usr/share/conv89     sales.dept120
PROJ1       lib    /usr/share/proj1      sales.dept122
PROJ2       lib    /usr/share/proj2      sales.dept122
TRAIN       lib    /usr/share/train
```

Example 2

You want to see which clients are mounting the NFS resources on the system called "main." You do not want a header to be displayed. Type

```
dfmounts -F nfs -h main
```

The system displays

```
-     main    /export       dancer, runner
-     main    /usr/man      jogger
-     main    /usr/share    dancer, runner
```

APPENDIX C: DFS MANAGEMENT MENUS

C The Distributed File System Management Menus

Introduction

System V Release 4.0 provides a menu interface to system administration called System Administration Menus (or *sis-adam,* because it is accessed through the sysadm command). When you install DFS Administration Utilities, menus are added to the interface for administering distributed file systems. These menus are known collectively as the Distributed File System Management menus. This section describes the DFS Management menus and the options the menus provide.

A complete description of the menu interface, including a tutorial, appears in the *System Administrator's Guide.* If you need information about how to use the sysadm menu interface, see the *System Administrator's Guide.*

The DFS Management Menu Tree

All DFS Administration commands can be accessed through the Distributed File System Management menus in the System Administration Menus.

To display the DFS Management menus, do the following:

1. Type `sysadm network_services`.

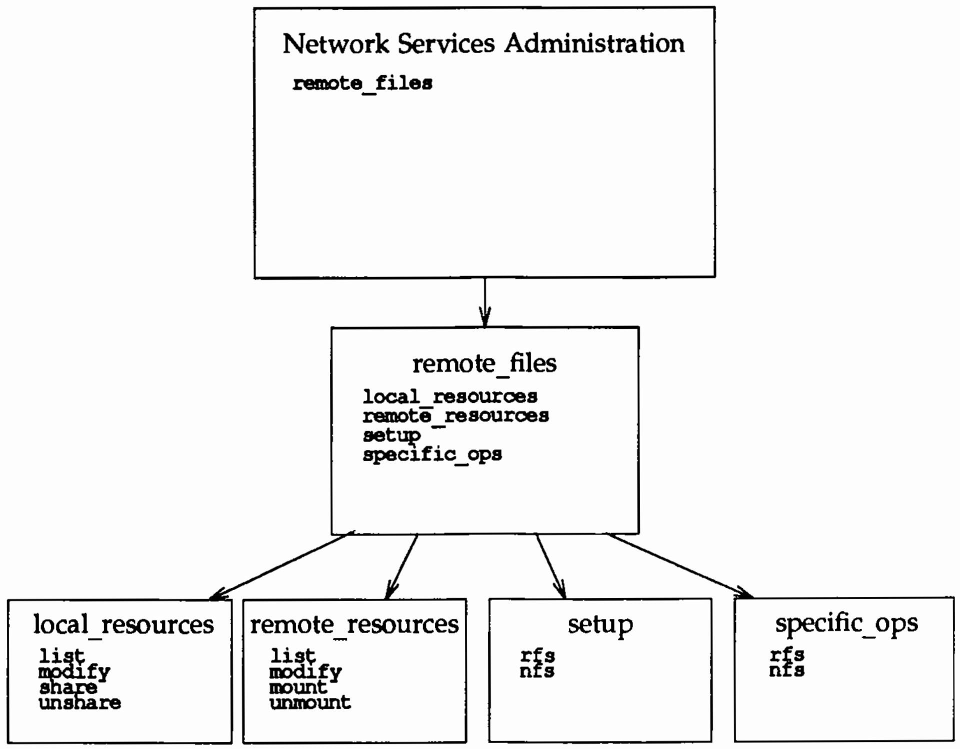
   The Network Services Administration menu is displayed.

2. From the Network Services Administration menu, select the option `remote_files`.

   The Remote Files menu is displayed.

From the Remote Files menu, you can access submenus that provide you with commands that correspond to the commands described in this guide.

Here is an illustration of sysadm menus that relate to distributed file system administration:



DFS Menu Options

The commands that correspond to the commands described in this guide are the commands for administering local and remote resources, which are accessed through the `local_resources` and the `remote_resources` submenus. (The `setup` submenu gives you package-specific commands for setting up RFS and NFS. The `specific_ops` submenu gives you package-specific commands for administering RFS and NFS.)

Menu options on the `local_resources` and the `remote_resources` submenus allow you to share, unshare, mount, and unmount resources *immediately*, or to specify resources to be shared and mounted and *automatically* whenever your system enters run level 3.

The `local_resources` submenu provides you with the following menu options.

- `list`, which allows you to list the resources on your system that are currently available to network clients or that are shared automatically whenever distributed file system operation begins.
- `modify`, which allows you to change the options with which resources on your system are currently shared, as well as the options with which resources are shared when they are shared automatically.
- `share`, which allows you to share selected resources on your system with network clients. The menu option lets you share resources immediately or specify resources to be shared automatically.
- `unshare`, which allows you to unshare resources on your system that are currently shared with network clients. The menu option also lets you cancel the automatic sharing of local resources.

The `remote_resource` submenu provides you with the following menu options:

- `list`, which allows you to list the remote resources that are currently mounted on your system and the remote resources that are mounted automatically.
- `modify`, which allows you to change the options with which remote resources are currently mounted on your system, as well as the options with which remote resources are mounted when they are mounted automatically.

- `mount`, which allows you to mount remote resources on your system and make them available to your users. The menu option lets you mount resources immediately and specify resources to be mounted automatically every time distributed file system operation begins.
- `unmount`, which allows you to unmount currently mounted remote resources. The menu option also lets you cancel the automatic mounting of remote resources.

Once you select a menu option from either submenu, screen messages and prompts lead you through the task you want to perform. If a message or a prompt is unclear to you, you can display a help screen, which gives you detailed information for completing the task.

DFS INDEX

Index

Contents

Figures and Tables

10. INTRODUCTION TO RFS

10. INTRODUCTION TO RFS

10 Introduction

About This Guide

This guide tells you how to administer Remote File Sharing (RFS). As a distributed file system package, RFS allows you to share resources on your computer with remote computers across a network of systems running UNIX System V.

Remote File Sharing is administered from the Release 4.0 System Administration Menus (sysadm). Once you access a sysadm menu, help screens provide you with all the information you need to complete a task. Therefore, this guide does not lead you step-by-step through the menu-based procedures. Instead, it provides background information and reference material you will need to administer and fine tune your RFS network.

Audience

This guide is directed to experienced administrators of stand-alone systems. It assumes an understanding of:

- mounting and unmounting local file systems
- assigning access rights to local resources
- maintaining system security

Organization

The organization of this guide is as follows:

Introduction To RFS	This section presents a general model of an RFS network, introduces some basic terminology, and describes the services that RFS provides.
Before You Begin	This section explains what you need to do before you can run RFS on your system and directs you to software installation procedures.
sysadm Interface	This section explains how to set up and maintain RFS Utilities on your computer, using the sysadm interface.

Command Line Interface	The remainder of this book. It will show you how to administer RFS through shell level commands and give you tips on troubleshooting problems. As you become a more experienced administrator, use this information as reference material to fine tune your RFS network.

Introduction To RFS

The primary function of RFS is to allow computers running UNIX System V to selectively share resources (directories containing files, subdirectories, devices, and/or named pipes) across a network. As an administrator of a computer on an RFS network, you can choose directories on your system you want to share and add them to a list of available resources on the network. From this same list, you can choose resources on remote computers that you would like to use on your computer.

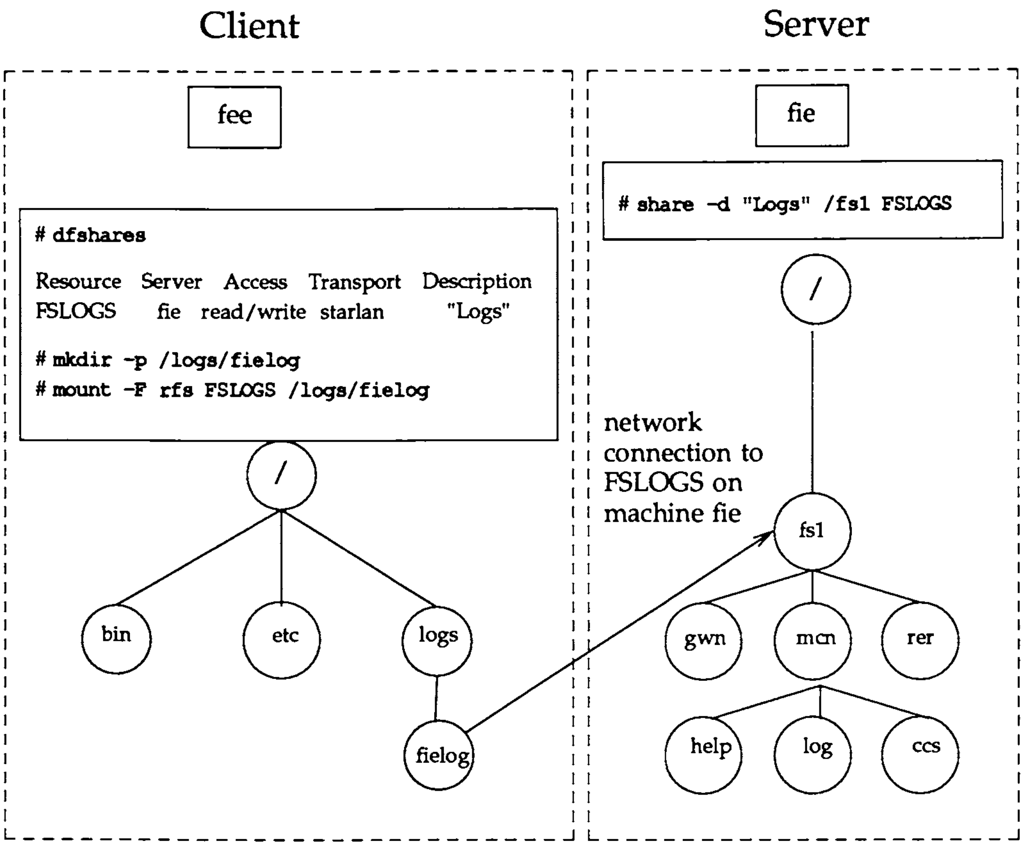
Resource Sharing

An RFS network generally consists of sub-groups of computers called domains. An RFS network may contain many domains. Sharing resources is a fundamental RFS operation that can be done between computers in the same domain, as well as between computers in different domains. It involves the notion of a server (a computer in the RFS network that offers a resource to others) and a client (a computer in the RFS network that uses a server's resource).

Sharing a resource on a Remote File Sharing system begins by identifying the pathname to a UNIX system directory you want to share. After identifying the directory you want to share, you then assign it a resource identifier (a name that other computers will use to reference that directory) and "share" it to other machines, using the `share` command. Computers that pass the security checks you have set up can then mount your resource as they would mount a file system locally. The `mount` command with the `-F rfs` option is used for mounting remote resources.

Figure 10-1 shows how two computers can share resources. In this example, the administrator of a computer named `fie` on a Remote File Sharing system wants to share all files and directories under `/fsl` on its file system tree. The administrator shares `/fsl` as a resource called FSLOGS.

Figure 10-1: Example — Sharing Resources



```
# dfshares

Resource  Server  Access     Transport  Description
FSLOGS    fie     read/write starlan    "Logs"

# mkdir -p /logs/fielog
# mount -F rfs FSLOGS /logs/fielog
```

```
# share -d "Logs" /fs1 FSLOGS
```

Another machine in fie's domain is called fee. The administrator from fee uses the dfshares command to see that FSLOGS is available on fie. The administrator then creates a directory called /logs/fielog on fee (mkdir command) and mounts FSLOGS on /logs/fielog.

Files or subdirectories from `/fs1` are now accessible to users on `fee`. Users can cd to the remote directory, list the contents, and run a remote program locally. If the resource contained the `/dev` directory, users could direct output to a remote device as though the device were on the local machine.

Domains

Each machine on a Remote File Sharing network must be assigned to a domain. The main reasons for domains are to simplify name service and provide a focal point for security of a group of machines.

Domain names act like telephone area codes. You can address all computers (nodes) and resources in your domain directly. For outside domains, you simply attach the domain name to the node name or resource identifier. This becomes increasingly valuable as RFS networks expand.

Name Service

Each domain must be assigned a primary and zero or more secondary domain name servers. These machines can share resources, like any other computer in the domain, but they have some special responsibilities.

Primary	The main duty of the primary domain name server is to keep track of all computers and resources within the domain it serves. It ensures that all resource identifiers and machine node names are unique within the domain. A required task of the primary domain name server is to add each computer to the Domain Member List and assign its RFS password. A list of shared resources are automatically stored on the primary, so any computer can see a complete list of available resources for the domain. Also, when a computer shares a resource, it registers its network address with the primary.

This way when a computer tries to mount another machine's resource the primary can tell the computer where the resource can be found on the network.

An optional function of the primary is to gather lists of each computer's users (`/etc/passwd`) and groups (`/etc/group`). Each computer in the domain can then use these lists to specifically define the permissions each machine's users will have to its resources.

A primary can also gather names and network addresses of other domains' name servers. Once the primary knows another domain name server's address, machines in its domain have the potential to access resources from any machine in the other domain.

Secondaries

If the primary crashes or gets shutdown, domain name service functions are automatically assumed by the first secondary domain name server defined in the `rfmaster` file. The secondary is intended to take over temporarily, until the primary comes back up.

While the secondary will have information needed to run the domain name server, domain information should not be modified on the secondary. When the primary comes back up, the secondary should be instructed to pass name server responsibility back to the primary (`rfadmin -p` command). Then the primary's administrator can change the domain member list, edit the `rfmaster` file, or gather optional user and group information again on the primary.

Transport Provider

The transport provider (TRANSPORT) provides the pathway used by Remote File Sharing to communicate with other machines. The term transport provider is used to refer to the physical network that connects the machines and the software needed to send messages across the network.

Remote File Sharing can communicate using any transport provider that is compatible with the AT&T Transport Interface Specification. The AT&T STARLAN NETWORK is one transport provider that can be used with RFS.

Although the transport provider is not considered part of the RFS package, RFS will not work if the transport provider is not functioning properly. Also, some information needed to configure RFS varies from one transport provider to another. For example, network addresses of the primary and secondaries and the network specification to identify the transport provider to RFS are dependent on the particular transport provider used.

Most of the scenarios in this guide show RFS running over a single transport provider. However, in UNIX System V Release 4.0, RFS can run over multiple transport providers at the same time. This optional feature is discussed in the "Multiple Transport Providers (Optional)" section of this guide.

The following sections describe transport provider information that relates to RFS administration: the Network Listener, Network Specification, and Network Addresses.

Network Listener

The network listener is part of the Networking Support Utilities package. Essentially, the listener's function is to wait for requests from the network. A call coming in from the network will request a particular service code. The service code will tell the listener to direct the call to a particular process.

Service code `105` is used to request RFS services. If all software installation was done as noted in the *Remote File Sharing Release Notes*, the RFS service code will be automatically configured for the listener of every transport provider you installed. Otherwise you will need to use the `nlsadmin` command to manually configure the listener. (See the "Setting Up RFS" section of this chapter.)

Network Specification

Since you could have several transport providers on one computer, you must tell RFS which transport provider will handle RFS on your machine. The network specification is the name you will use when you initially configure RFS to indicate its transport provider. The AT&T STARLAN NETWORK, for example, uses `starlan` as its network specification. This tells RFS that `/dev/starlan` is the device representing the transport provider to use.

Network Addresses

When Remote File Sharing is started on a machine, the machine tries to contact its domain's primary name server. In order to do that, the machine must know the primary's network address.

The form of the network address varies according to the transport provider used. The AT&T STARLAN NETWORK convention for network addressing is to use a machine's node name and append the string `.serve` to create the network address. For example, the network address for a machine whose node name is `charlie` would be `charlie.serve` on an AT&T STARLAN NETWORK. The `nlsadmin` command is used to obtain your machine address, regardless of the network used.

Security

RFS provides several mechanisms for ensuring the security of your resources. Some of these mechanisms, however, require diligence to set up and maintain. This is especially true if the machines, resources, and users are constantly changing on the network.

As a system administrator you can maintain strict control of your resources. No files, directories, or devices in an unshared file system can be accessed by other computers. Standard UNIX system file security measures can be used in combination with special RFS facilities to protect your resources.

Direct access to your computer is controlled because local users still have to log in as they always have. As for remote accessibility, you can set up security to allow only certain remote computers to access your resources.

The major mechanisms in RFS for protecting your resources are described in the sections "Verify Computers," "Restrict Resources," and "Map IDs."

Verify Computers

When a remote computer tries to mount a resource from your computer, and no other resources are mounted, it tries to set up a connection (virtual circuit) across the network to your machine. Once this virtual circuit is set up, the remote machine can mount any resource you have made available to it. This virtual circuit is closed when the last resource is unmounted.

Before this virtual circuit is created, you can verify that the computer is the one it claims to be by checking its RFS password. The following text describes what happens when verification is and is not used.

- No verify: any computer can connect

  If the computer is listed in the `/etc/rfs/auth.info/`*domain*`/passwd` file, your machine will check its password. Otherwise, your computer will accept it as the machine it claims to be.

- Verify: some computers can connect

  If you use the RFS verification feature, you can make sure that only specific machines can use any of your resources. Those machines must be listed in the proper `/etc/rfs/auth.info/`*domain*`/passwd` file and must match the password you have for them. (*domain* is the domain name of the requesting machine.) You can tailor this file if you only want a subset of machines to be allowed to connect. (A description of how to use this feature is contained in the "Setting Up RFS" section of this chapter.)

Restrict Resources

Once a remote computer has established a connection to your computer, the resources it can mount from your machine depend on how you shared each resource. These are your choices:

- any machine can mount

  You may have shared the resource so that any machine that can connect to your machine can mount it.

- some machines can mount

  You restricted access to the resource to certain machines. The remote computer trying to mount it must be one of those machines.

You also may have shared the resource as read-only. In that case, the remote computer must mount the resource read-only instead of read/write (default).

Map IDs

Remote users' permissions can be defined to provide another layer of security for a mounted resource. Remote users and groups can be mapped into your local computer's user and group list to set permissions of your resources.

You can set these mapping rules on a global or per-machine basis. The global rules set user and group permissions for all remote machines that do not have explicit mapping rules.

Here are the ways you can map remote machines' user-ids into your machine. These rules apply to both global and per-machine mapping.

- no mapping

  If you don't set any special mapping for any remote computer, all users are mapped into your machine as a "special guest" user ID/group ID. This is the easiest approach because you don't need to keep any records for the remote machine, create rules files or run the `idload` command.

- default mapping

  You can set default mapping so that all remote users are mapped into one of these permissions:

  - the local user ID number that matches each remote user's ID (`default transparent`)
  - a single local ID number
  - a single local ID name
  - the local user name that matches each remote user's name (`map all`)

  Group permissions can be mapped in the same way. Users and groups are mapped independently. If there are exceptions to the default mapping, you can `exclude` certain users and groups so they only have special guest permissions (for example, `exclude 0`).

- specific mapping

  You can map any user or group from any remote machine into a specific user or group on your machine. This can be done by user name or numeric ID.

Using these mapping techniques and standard methods for setting file permissions, you can keep strict controls over your resources, even after they are remotely mounted. (See the "Mapping Remote Users" section of this chapter for more details.)

RFS Features

Here are some RFS feature design considerations.

Compatibility	Once you mount a remote resource on your system it will look to your users as though it is part of the local system. You will be able to use most standard UNIX system features on the resource. Standard commands and system calls, as well as features like File and Record Locking, work the same on remote resources as they do locally. Applications should be able to work on remote resources without modification.
Flexibility	Since you can mount a remote resource on any directory on your system, you have a lot of freedom to set up your computer's view of the world. You do not have to open up all your files to every machine on the network. Likewise, you do not have to make all files on the network available to your computer's users.
Performance (Client Caching)	The client caching feature of RFS provides substantial performance improvements over non-caching systems by reducing

the number of times data must be read across the network. Client refers to the computer that is using a remote resource, while caching refers to the client's ability to store data in local buffer pools.

The first time a client process reads a block of data from a remote resource, it is placed in local buffer pools. Subsequent client processes reading a server file can avoid network access by finding the data already present in local buffers. This generally causes a large reduction in network messages, resulting in improved performance.

In order for client caching to work simply and reliably, the following features were built into it:

- Cache consistency. Checking mechanisms are used to ensure that the cache buffers accurately reflect the contents of the remote file the user is accessing.
- Transparency. The only difference users should see between caching and non-caching systems is improved response time. RFS-based applications do not have to be changed to run on a Remote File Sharing system that caches remote data.
- Administration. By default, client caching is on. However, options are available to turn off caching for an entire system or for a particular resource. (You would probably only do this if you have an application that does its own network buffering.) There are also some tunable parameters available to fine tune your system according to the way you use RFS. (See the "Monitoring" and "Parameter Tuning" sections of this chapter for more information.)

Before You Begin

Remote File Sharing services are provided in the Remote File Sharing Utilities, an optional package consisting of three floppy diskettes that is delivered with UNIX System V Release 4.0 software. Before you can run RFS, you must make sure that all your network hardware is in place, and that all the prerequisite software is installed on your system.

This section describes the hardware and software dependencies of Remote File Sharing and directs you to the UNIX System V Release 4.0 Installation Guide for complete software installation instructions. It applies for using the `sysadm` interface as well as the command interface.

Prerequisites

Before you can install Remote File Sharing Utilities, you must complete the following prerequisites.

Software

You must make sure the following software is installed before you install Remote File Sharing Utilities. The software must be installed in the order shown below or the network listener will not be configured properly. You would then have to configure the listener manually, as described later in this chapter.

NOTE For machines that use UNIX System V Release 4.0, only RFS Release 2.0 Software packages should be used when operating in a heterogeneous environment.

- Essential Utilities (UNIX System V Release 4.0)
- System Administration Utilities
- Directory and File Management Utilities
- Networking Support Utilities

- TCP/IP Utilities, a transport provider compatible with AT&T Transport Interface, or STARLAN.
- Distributed File System Utilities

You must also have the following space available on your system:

- / (root file system) needs 1248 blocks of free space.
- /usr needs 1913 blocks of free space.

NOTE 1248 of the blocks needed in root are just temporary space that is used to reconfigure the system. After the package is installed, only about 800 blocks are used up.

Hardware

The following hardware is required:

- Minimum of 4 megabytes of memory. (If you are upgrading to 4 megabytes of memory, make sure you update your tunable parameters to match that amount of memory. See the "System Setup" chapter of the *System Administrator's Guide* for more information.)
- Any communications hardware required by the network you are using.

Installation

The Remote File Sharing Utilities package is provided as an optional package with UNIX System V Release 4.0, along with all the prerequisite software. Refer to the installation manual you received with the UNIX System V documentation set for complete installation instructions.

11. RFS sysadm INTERFACE

11 RFS sysadm Interface

Overview

Here is an overview of the procedures you will need to set up and maintain your RFS network. The `sysadm` menu interface is used to accomplish these tasks. It not only lets you add all basic RFS configuration information, but it also acts as a tutorial by introducing and explaining key RFS concepts. Once you access a `sysadm` menu, help screens provide you with background information and explanations regarding menu selections. Access the help screens by using the "HELP" function key; use the "CANCEL" function key to exit the help mode. Continue making menu selections until you complete the particular task.

Procedure 1	`Set Up Remote File Sharing` To set up all basic information needed to run Remote File Sharing.
Procedure 2	`Start/Stop Remote File Sharing` To start and stop Remote File Sharing, check if it is currently running, and set up RFS to start automatically at system boot time.
Procedure 3	`Local Resource Sharing` To manage the local resources you make available to other machines.
Procedure 4	`Remote Resource Mounting` To manage remote resources made available to your machine.
Procedure 5	`Change RFS Configuration` To change your ID mapping, show your current RFS configuration, or update the domain member list.

These procedures are designed to help you set up and maintain Remote File Sharing (RFS) Utilities on your computer. Should you need more information as you are setting up RFS, you should consult the glossary of terms provided in the back of this chapter, or you should consult the corresponding section names for each procedure.

The `sysadm` interface lets you do everything necessary to set up and run RFS in a basic configuration. There are several optional features that are not available through the `sysadm` interface, however.

The optional features not available using the sysadm interface are described at the end of the "Setting Up RFS" section. The word "Optional" is placed in the heading of each optional feature. The features include the following:

Remote Computer Verification.
: By default, when a machine requests the use of one of your resources, your machine will process the request without verifying the remote machine's password. This procedure describes how to restrict access of all your resources to a limited group of remote machines whose names and passwords match those in lists you set up.

Complex user ID/group ID mapping.
: ID mapping defines the permissions remote users will have to your resources. The choices of ID mapping schemes are limited when you use the sysadm interface to set up mapping. This procedure gives you more flexibility in setting up permissions for remote users.

Multiple Domain Resource Sharing.
: The sysadm interface assumes that you are only sharing resources within one domain. However, it is possible to have more than one domain on a network. This procedure describes how to share resources among multiple domains on the same network.

Multiple Domain Name Service.
: When you define the primary and secondary name servers using sysadm, you are defining them to serve a single domain. You can, however, define the same set of machines to be the name server for several domains using this procedure.

More experienced RFS administrators will also be interested in the "Monitoring" and "Parameter Tuning" sections of this chapter. Information in these sections will help you fine tune your system so RFS can make the most efficient use of your system's resources.

Procedure 1: Set Up Remote File Sharing

This procedure is used to set up Remote File Sharing on your machine. When the procedure is done, you will have completed everything needed to run RFS on your system.

Prerequisites

In addition to the prerequisite steps you followed in the section, "Before You Begin," you should do the following before you begin setting up RFS:

- Choose one or more computers on the network to act as domain name servers. Exactly one primary is required. All domain administration is done from the primary. You can choose zero or more secondary domain name servers. These are defined simply to keep the name server running temporarily, should the primary fail. (You can configure RFS on your machine before the primary is configured and running RFS. However, you cannot start RFS until the primary begins running RFS.)
- Log in as `root`.

After you have accomplished the prerequisites, begin to set up RFS by typing `sysadm network_services` and selecting `remote_files`. Continue by selecting `setup` and then `rfs` to bring you to the following screen:

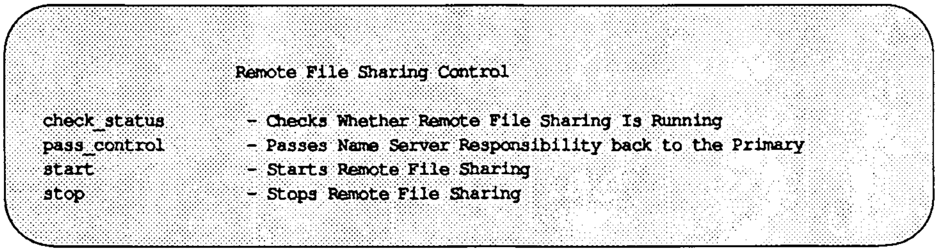
```
                       Initial Remote File Sharing Setup

set_networks               - Sets Up Network Support for RFS
set_domain                 - Sets the Current Domain for RFS
add_nameserver             - Adds Domain Name Servers
add_host                   - Adds Systems to the Domain Password File
start                      - Starts Remote File Sharing
share                      - Shares Local Resources via Remote File Sharing
mount                      - Mounts Remote Resources via Remote File Sharing
set_uid_mappings           - Sets Up UID Mappings
set_gid_mappings           - Sets Up GID Mappings
```

You should execute each of the tasks in the order listed. Continue making interactive menu selections until the job is done. Remember, the "HELP" function key will provide you with help messages along the way. After completing this procedure, you can check if RFS is running by returning to the shell and typing `rfadmin -q`, or, you can use the `check_status` menu function described in the next procedure. The chapter section "Setting Up RFS" will provide you with detailed background information in this area.

Procedure 2: Start/Stop Remote File Sharing

This procedure is used to start and stop RFS. It's also used to determine if RFS is running. As a prerequisite, you should have already set up RFS.

Begin by typing `sysadm network_services` and selecting `remote_files`. Continue by selecting `specific_ops`, `rfs`, and `control`. You are now at the following screen:



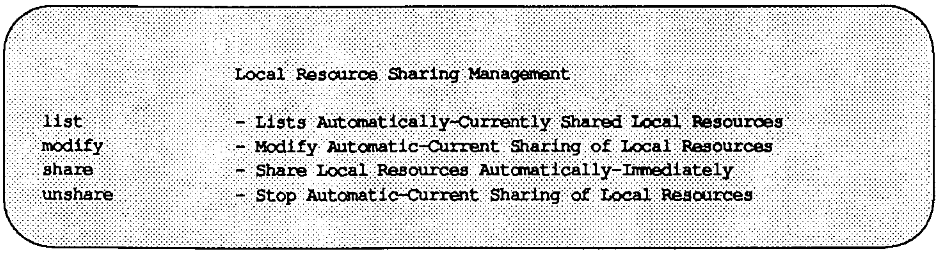
```
                    Remote File Sharing Control

check_status       - Checks Whether Remote File Sharing Is Running
pass_control       - Passes Name Server Responsibility back to the Primary
start              - Starts Remote File Sharing
stop               - Stops Remote File Sharing
```

Select `start` to start RFS. If RFS does not start, see the "Starting/Stopping Remote File Sharing" section in this chapter for a list of possible problems. Selecting `stop` will stop RFS and selecting `check_status` will report if RFS is running. Remember, the "HELP" function key will provide you with help messages along the way.

Procedure 3: Local Resource Sharing

This procedure allows you to selectively make your local resources available or unavailable (share/unshare) to remote systems. You can arrange this to happen automatically when RFS is started, to happen immediately, or both. You can also modify the options by which your local resources are currently or automatically shared, via RFS. Finally, this procedure enables you to list your local resources currently available to be shared by remote systems, via RFS.

As a prerequisite, RFS should be set up and running. Begin by typing `sysadm network_services` and selecting `remote_files`. Continue by selecting `local_resources`. You are now at the following screen:



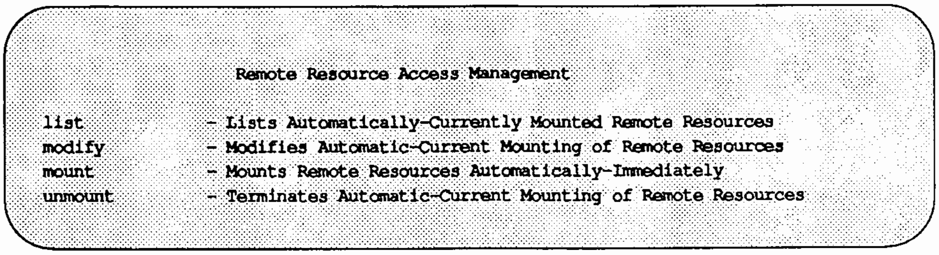
```
                    Local Resource Sharing Management

list                - Lists Automatically-Currently Shared Local Resources
modify              - Modify Automatic-Current Sharing of Local Resources
share               - Share Local Resources Automatically-Immediately
unshare             - Stop Automatic-Current Sharing of Local Resources
```

Select `list` then `rfs`. This gives you access to menu selections allowing you to list the local resources currently shared by RFS. Select `modify` then `rfs` to modify sharing permissions of local resources via RFS. Select `share` then `rfs` to share local resources via RFS. Select `unshare` then `rfs` to unshare local resources currently shared via RFS. Remember, the "HELP" function key will provide you with help messages along the way. Consult the "Sharing Resources" section in this chapter for further details on these tasks.

Procedure 4: Remote Resource Mounting

This procedure enables you to selectively make remote resources available or unavailable (mount/unmount) to your local computer. You can arrange to have this happen automatically, immediately, or both. You can also modify the options by which remote resources are currently or automatically mounted on your local computer. Finally, this procedure enables you to list the resources of remote systems that are currently available to users on your local computer system, via RFS.

As a prerequisite, RFS should be set up and running. Begin by typing `sysadm network_services` and selecting `remote_files`. Continue by selecting `remote_resources`. You are now at the following screen:



```
                    Remote Resource Access Management

list           - Lists Automatically-Currently Mounted Remote Resources
modify         - Modifies Automatic-Current Mounting of Remote Resources
mount          - Mounts Remote Resources Automatically-Immediately
unmount        - Terminates Automatic-Current Mounting of Remote Resources
```

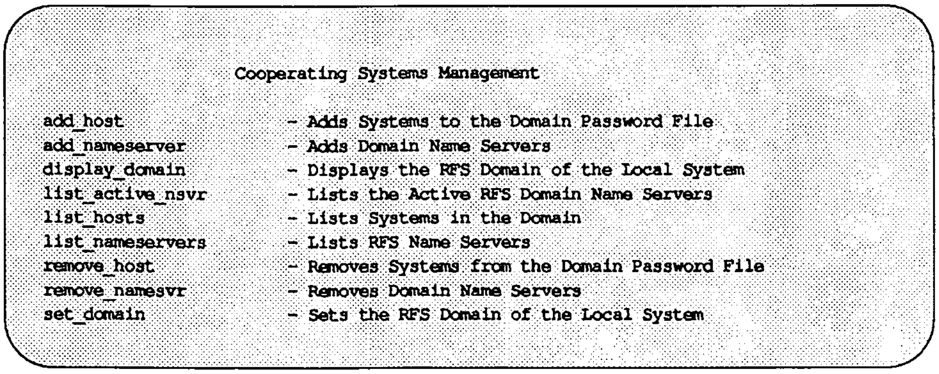
Select `list` then `rfs`. This gives you access to menu selections allowing you to list the remote resources currently mounted via RFS. Select `modify` then `rfs` to modify mount permissions of remote resources. Select `mount` then `rfs` to mount remote resources. Select `unmount` then `rfs` to terminate mounting of remote resources currently shared via RFS. Remember, the "HELP" function key will provide you with help messages along the way. Consult the "Remote Resource Mounting" section in this chapter for further details on these tasks.

Procedure 5: Change RFS Configuration

This procedure covers how to examine your RFS network configuration, how to update domain member lists, and how to change ID mapping. As prerequisites, RFS should be set up and running.

RFS Configuration/Domain Management

The task of examining your current RFS network configuration and effectively changing it can be accomplished using the `sysadm` menu `Cooperating Systems Management`. When you type `sysadm network_services` and select `remote_files`, `specific_ops`, `rfs`, and `systems`, the following menu will appear on your screen:



```
                    Cooperating Systems Management

add_host                 - Adds Systems to the Domain Password File
add_nameserver           - Adds Domain Name Servers
display_domain           - Displays the RFS Domain of the Local System
list_active_nsvr         - Lists the Active RFS Domain Name Servers
list_hosts               - Lists Systems in the Domain
list_nameservers         - Lists RFS Name Servers
remove_host              - Removes Systems from the Domain Password File
remove_namesvr           - Removes Domain Name Servers
set_domain               - Sets the RFS Domain of the Local System
```

The `list_`*name* selections, and `display_domain`, will show you how your RFS network is currently configured (set up). The other menu selections will allow you to configure the various domain parameters of your network. Here again, the "HELP" messages are extensive in their explanation of each menu selection. In addition, the "Domain Name Servers" section in this chapter will provide more detailed information.

Transport Providers

The software path over which your RFS network applications can communicate is important. The `Supporting Networks Management` menu will allow you to display and set the transport provider for your RFS network. You can access this menu by typing `sysadm network_services` and selecting `remote_files`, `specific_ops`, `rfs`, and `networks`. The following menu will appear on your screen:

```
                          Supporting Networks Management

display         - Displays Networks Supporting Remote File Sharing
set             - Sets Network Support for Remote File Sharing
```

ID Mapping

ID mapping, or, defining remote users' permissions to your files, is the final ingredient to configuring your RFS network. The `sysadm` menu `User and Group ID Mapping Management` allows you to display and set up some simple user and group ID mappings. (See the "Mapping Remote Users" section for more sophisticated mapping schemes.) You can access this menu by typing `sysadm network_services` and selecting `remote_files`, `specific_ops`, `rfs`, and `id_mappings`. The following menu will appear on your screen:

```
                      User and Group ID Mapping Management

display                 - Displays Current User and Group ID Mappings
set uid mappings        - Sets Up Standard UID Mappings
set gid mappings        - Sets Up Standard GID Mappings
```

12. RFS COMMAND INTERFACE

12. RFS COMMAND INTERFACE

12 RFS Command Interface

Setting Up Remote File Sharing

In most cases, you will not need the set of tasks described in this section because the basic Remote File Sharing (RFS) configuration and reconfiguration can be handled using the sysadm interface, as described in Procedure 1. These tasks are for those who want to go deeper into the workings of RFS or are having problems with particular components.

These tasks are run from the shell. They should be run initially in the order described.

Once these tasks are completed, go to the "Starting/Stopping RFS" section for information on starting RFS.

Prerequisites

See "Prerequisites" in the "Before You Begin" section.

Set Node Name

CAUTION Changing the node name of your computer requires careful coordination with all machines that communicate with yours using RFS or other communications packages that rely on node name.

Check to see if your computer's node name is set to the name you want (uname -n). If it's not, set it by typing:

```
uname -S nodename
```

A node name that is valid for RFS can consist of up to 8 characters of letters (upper and lowercase), digits, hyphens (-), and underscores (_). Some networks, such as AT&T STARLAN NETWORK, require that every node name in the network be different. RFS, however, only requires that every node name in a domain be different.

Set Up Network Listener

If you have installed the Networking Support Utilities, the AT&T STARLAN NETWORK, and RFS Utilities in the order described in the *Remote File Sharing Release Notes*, you can skip this task. The listener will already be installed and set up to run automatically, and RFS will be listed as an available service.

If you are using another transport provider, or suspect that your AT&T STARLAN NETWORK listener is improperly set up, this task shows how to manually set up the listener. In the following example the AT&T STARLAN NETWORK is used. To set up the listener for other networks compatible with the AT&T Transport Interface, you would replace `starlan` with the name of the network (network specification) you are installing. (For more details, see the `nlsadmin`(1M) manual page.)

To check if the listener is properly installed and set up for use by RFS, type the following:

```
        nlsadmin -v starlan
```

If service code 105 is listed, then the listener is configured to be used for RFS.

Run the following commands if the listener is not properly set up. If you run any of these commands and they have already been run, you will receive a message telling you so. This won't harm your listener configuration. Type:

```
        nlsadmin -i starlan
```

to initialize the files needed for the listener process for the network specified, in this case `starlan`.

Next type:

```
  nlsadmin -a 105 -c /usr/net/servers/rfs/rfsetup -y "RFS server" starlan
```

to add the RFS service (`rfsetup`) to the list of services available to the `starlan` listener.

Use the following command line to report the status of the `starlan` listener process installed on this machine (ACTIVE or INACTIVE):

```
nlsadmin -x
```

Next type:

```
nlsadmin -l "nodename.serve" -t "nodename" starlan
```

to register the network addresses of your machine. The listener will listen for requests for these addresses on the network. Only the `-l` address is required by RFS. The `-t` address is used only for terminal services and may not be needed on all networks. However, if the `-t` address is specified and remote login service is not defined for the terminal, a warning message will be produced. (Other networks will use other types of network addresses. See the description of the `rfmaster(4)` manual page in the *System Administrator's Reference Manual* for the syntax of different address types.)

To start the listener, type:

```
nlsadmin -s starlan
```

Normally, it will be started automatically when your machine enters multi-user mode (`init 2`).

Set the Domain Name

Set the domain name by typing:

```
dname -D domain
```

where *domain* is replaced by the domain your machine will be a member of. The domain name must:

- contain no more than 14 characters
- consist of any combination of letters (upper or lowercase), digits, hyphens, and underscores
- be different from the name of any other domain used on the network

You can check the current domain name by simply typing:

```
dname
```

Set the Transport Provider

To identify the network, you must tell RFS the network (transport provider) it should use. (In our example, this is `starlan` for the AT&T STARLAN NETWORK.)

```
        dname -N starlan
```

This command indicates the device, relative to the `/dev` directory, that is used for the transport provider. If RFS is going to be run over multiple transport providers, then the transport providers must be specified as a comma-separated list.

```
        dname -N starlan,tcp
```

Create rfmaster File

The primary and secondary name server assignments for a domain are stored in the `/etc/rfs/`*transport*`/rfmaster` file. The primary keeps the definitive copy of this file and distributes it automatically to each computer in the domain when each starts RFS. This file also contains the network address of other name servers.

The `rfmaster` file should only be created manually on the primary. If your machine is not the primary, you should skip this task; the `rfmaster` file for your domain will automatically be placed on your machine the first time you start RFS (`rfstart -p` *primary_addr*).

If you are on the primary, you can create an `rfmaster` file in the `/etc/rfs/`*transport* directory using any standard file editor. The contents of this file will define:

- the primary name server for your domain
- secondary name servers for your domain
- network addresses for each of these machines

If you specified multiple transport providers (dname command), you must have an `rfmaster` file in each `/etc/rfs/`*transport* directory. (See the section on "Multiple Domain Name Service" in this chapter for a description of other information you may want to put into the `rfmaster` file.)

Here is an example of an rfmaster file for a domain, called peanuts, whose primary is charlie and whose secondary name servers' node names are linus and lucy. Adding each machine's domain name (peanuts) to its node name, separated by a period, forms its full RFS machine name. Each line of the example translates as follows.

- For domain peanuts, the primary is peanuts.charlie.
- For domain peanuts, the first secondary is peanuts.linus.
- For domain peanuts, another secondary is peanuts.lucy.
- For computer peanuts.charlie, the network address is charlie.serve.
- For computer peanuts.linus, the network address is linus.serve.
- For computer peanuts.lucy, the network address is lucy.serve.

(The addresses shown examples of AT&T STARLAN NETWORK addresses. These addresses should be in the form *nodename*.serve.)

```
peanuts         p       peanuts.charlie
peanuts         s       peanuts.linus
peanuts         s       peanuts.lucy
peanuts.charlie a       charlie.serve
peanuts.linus   a       linus.serve
peanuts.lucy    a       lucy.serve
```

Each line in the example is an entry. The second field is the *Type* field, which indicates whether the entry defines a primary name server (p), secondary name server (s), or the network address (a) for one of these name servers. Here is the information needed for the first field, *Name*, and the third field, *Rdata*, for each type of entry.

p	Primary entry. *Name* is the domain name. *Rdata* is the full RFS machine name of the domain's primary name server (*domain*.*nodename*).
s	Secondary entry. *Name* is the domain name. *Rdata* is the full RFS machine name of the domain's secondary name server (*domain*.*nodename*).

a	Address entry. *Name* is the full RFS machine name (*domain.nodename*) of a name server computer. *Rdata* is the network address of the computer. The manuals that come with your network should describe how to find a computer's network address.

Here are some special considerations when creating the file.

- Fields in each entry must be separated by a blank or a tab.
- The address must be in ASCII text or hexadecimal notation. For hexadecimal, the field must begin with `\x` and contain an even number of digits. If the address contains tabs or spaces, the field must be surrounded by double quotes (" ").
- An entry can extend beyond one line if you enter a backslash (\), then a carriage return to continue to the second line.
- This file should be write protected from all but `root`, but all read permissions should be enabled (644 permissions).
- If you start a line with the # character in column 1, the entire line will be treated as a comment.
- There can be multiple `rfmaster` files when operating RFS over multiple transport providers.

Add/Delete Password For Domain Members

If your computer is the current primary name server for the domain, you must add each computer to the list of the computers that make up an RFS domain (domain member list). If a secondary has temporarily taken over, the secondary must pass name server responsibility back to the primary using the `rfadmin -p` command. To add members, use the following command:

```
# rfadmin -a domain.nodename
Enter password for nodename:
Re-enter password for nodename:
```

where *nodename* is replaced by the node name of the computer you want to add to your *domain*. (The two names must be connected by a period.)

You will be prompted for an initial password, which will be stored in the `/etc/rfs/auth.info/`*domain*`/passwd` file for your *domain*. When the computer you added starts RFS, the computer's administrator must enter this password. You can simply type a <CR> for a null password. Otherwise, the password must conform to the same criteria used with the `passwd` command. Repeat this command for each computer you want to add to the domain.

NOTE

Adding a primary and secondary to the `rfmaster` file does not automatically add them to the domain. You must do this procedure for each of those machines.

You can also use the `rfadmin` command to delete members from the domain member list, as follows

`rfadmin -r` *domain.nodename*

Remote Computer Verification (Optional)

NOTE

This procedure assumes you are starting RFS from the shell using `rfstart -v` or `init 3` manually or automatically at boot time to start RFS.

When you start RFS, you can indicate that all remote machine passwords be verified when they try to use your computer's resources. The `rfstart` command is run automatically when you go into RFS state (`init 3`).

If you use `rfstart` with the `-v` option, any machine that tries to mount your resources must match a name and password you have in the `passwd` file in the `/etc/rfs/auth.info/`*domain* directory on your machine, where *domain* is replaced by the name of the remote computer's domain. If the remote computer is not listed in this `passwd` file, if it is listed and the password doesn't match, or if no `passwd` file exists, the remote mount will fail. (This file is automatically on the primary, but it must be added to other machines, as described in this procedure, to use verification.)

If you don't use the `-v` option, the following validation occurs. If a `passwd` exists for the remote computer's domain on your computer and the remote computer is listed, but the password doesn't match, a mount request will fail. If the computer is not listed in the file or if the `passwd` file doesn't exist, the computer will be allowed to mount your resources without validation. (Of course, a remote mount could still fail if the resource was shared to a limited subset of machines or was shared read-only and the machine tried to mount it read/write.)

The following steps describe how verification is set up.

Step 1: Obtain UNIX `passwd` file(s). The `/etc/rfs/auth.info/`*domain* directory on the primary will contain this file. (*domain* is replaced by the domain name.) The file will have the name and encrypted password for each machine in the domain.

You must make the `/etc/rfs/auth.info/domain/passwd` file, plus the `passwd` file for any outside domains containing machines you want to verify, accessible to your machine in one of the following ways:

Step 1A: Place a copy of this file(s) in the same directory on your machine. The `passwd` file for each domain must be in the appropriate *domain* subdirectory.

`or`

Step 1B: Have the primary for each domain share the `/etc/rfs/auth.info` directory; then have it automatically mounted in the same location on your computer. This way you can automatically pick up any changes in machines or passwords. (See the description of `/etc/vfstab` in the "Automatic Remote Mounts" section of this chapter for information on setting up automatic mounts.)

Step 2: `rfstart -v`. You must edit the `/sbin/rc3.d/S21rfs` file to automatically run `rfstart` with the `-v` option. You will add the `-v` after the `rfstart` command, as shown in the following example.

```
'rfstart')
   trap 'rm -f /var/tmp/rfs$$;exit' 0 1 2 3 15
   stat=1
   retries=0
   while [ ${stat} -eq 1 ]
   do
     /usr/sbin/rfstart -v </dev/console >/dev/console 2>/var/tmp/rfs$$
     stat=$?
     case ${stat} in
```

Step 3: If you want to verify only a limited subset of these computers, you must use manually edited versions of the `passwd` files, removing any computers you want to prevent from using your resources. (You cannot edit this file if you are a primary or secondary name server or if you have mounted the file from the primary.)

Resource Sharing with Other Domains (Optional)

For computers in your domain to share resources with computers in other domains on your network, you must do the following.

Step 1: Find out:

- the primary name server for each domain
- the secondary name server(s) for each domain
- the network address for each of the above name servers

Step 2: You must see that the information in Step 1 is added to your domain's `/etc/rfs/`*transport*`/rfmaster` file on the primary. For a format description of the `rfmaster` file, see the "Create `rfmaster` File" section of this document.

The following example shows the information added to contact a domain called docs.

```
docs          p     docs.big
docs          s     docs.little
docs.big      a     big.serve
docs.little   a     little.serve
```

Step 3: Stop RFS on the primary (rfstop, init 2, or use the sysadm menus as described in the "RFS sysadm Interface" section).

Step 4: Restart RFS on the primary (rfstart, init 3, or use the sysadm menus as described in the "RFS sysadm Interface" section). Make sure start up has completed before going to the next step.

Step 5: If a secondary machine took over name service when the primary was stopped, pass name service responsibilities back to the primary by typing the following from the secondary:

```
rfadmin -p
```

Step 6: Mount resources from an outside domain. Once the name server machines have picked up the new domain names, you can mount a resource from a remote domain on your own machine. You would use the same method of mounting a resource from an outside domain as you would to mount a resource from your domain, with one exception. When you specify the resource to be mounted, you must prepend the domain name to the resource identifier. For example, the command:

```
mount -F rfs docs.INFO /usr/info
```

could be used to mount resource INFO, shared in domain docs, with read/write permissions, onto directory /usr/info.

Multiple Domain Name Service (Optional)

Once you have defined a set of primary and secondary name servers to serve a domain, that set of machines may also be name servers for another domain on the same network. The following procedure describes how this can be configured:

Step 1: Edit `rfmaster` file. You must add the information on the new domain's name servers to the `rfmaster` file on the primary. The following is an example of two sets of name servers that serve domains called `docs` and `peanuts`.

```
docs             p       docs.big
docs             s       docs.little
docs.big         a       big.serve
docs.little      a       little.serve

peanuts          p       peanuts.charlie
peanuts          s       peanuts.linus
peanuts.charlie  a       charlie.serve
peanuts.linus    a       linus.serve
```

Step 2: Stop and restart RFS. You must stop all machines served by the primary (`rfstop`, `init 2`, or use the `sysadm` menus as described in the "RFS `sysadm` Interface" section). You must then restart the primary (`rfstart`, `init 3`, or use the `sysadm` menus as described in the "RFS `sysadm` Interface" section). Then start each machine on the system, starting machines that previously had other machines as domain name servers with the `rfstart -p` *address*, where *address* is replaced by the network address of the new primary domain name server. This will ensure that the new information is picked up by each machine.

Complex User ID/Group ID Mapping (Optional)

ID mapping lets you control the access remote users will have to files and directories that make up your shared resources. This feature lets you assign each remote user the permissions of one of your local users (listed in `/etc/passwd`) or the permissions of a special "guest ID," with respect to your shared resources. The guest ID will never overlap with any of your local users. The same mechanism can be used to define group permissions (listed in `/etc/group`).

Use this procedure as a tutorial for ID mapping and as a procedure for setting up mapping. If you have questions about particular mapping components, refer to the "Mapping Remote Users" section of this chapter.

When Not to Map

In most cases, ID mapping is not necessary. If you never set up mapping using this or the `sysadm` procedures, all users will be mapped into a single special guest ID. This special guest ID is represented by an ID number that is one higher than the maximum allowed for your system. By default, the maximum number of users and groups on a system is 60000, so the special guest is ID number 60001.

No mapping, or the default mapping, provides the maximum security for your shared resources. When a remote user lists the permissions of your files (`ls -l`), all files will be owned by 60001 or 60002. 60001 means the file was created by a remote user and, therefore, is owned by every remote user that can access your resource. 60002 means the file was created by one of your local users and, therefore, remote users can only access the file if the "other" permissions are set (see the `chmod`(1) manual page).

When to Map

Using mapping increases the power and flexibility of RFS. The following are some reasons you may want to use mapping:

Special permissions.
: You may want to map some or all remote users into particular local users' permissions. For example, if you are the administrator of several machines, you may want to map all `root` logins together across the machines. That way you

would be able to modify any remote resources mounted on any machine you are working from.

Transparent mapping.

If you set up a group of computers to have the exact same `/etc/passwd` and `/etc/group` files, mapping transparently can be a very powerful technique. When a user creates a file, the user will maintain sole ownership to the file, whether or not the file resides on a remote resource.

With transparent mapping, you could share many resources that require a consistent view of user ownership. For example, you could share your `/var/mail` directory, mount it on `/var/mail` on other computers, and have one mail directory for the entire set of machines. The basic concept is that you can avoid duplication of many files and directories while maintaining consistent user permissions.

Mapping by machine.

You may want to map users from one machine differently than users from another machine. For example, you may want to map all users from one machine into user ID `600`, from another machine into `700`, and from a third into `800`. In that way you could monitor which remote machine's users were creating files within your resources.

NOTE The default mapping and transparent mapping can be set up using the `sysadm` interface described in the "RFS `sysadm` Interface" section of this chapter. For other mapping techniques, see the section "Mapping Tools and Files."

Mapping Tools and Files

The result of this procedure is "mapping translation tables." These tables will be used by your system to process requests from remote users for access to your resources that are mounted on their computers.

The command used to create the translation tables is `idload`. When `idload` is run with no options, it does the following:

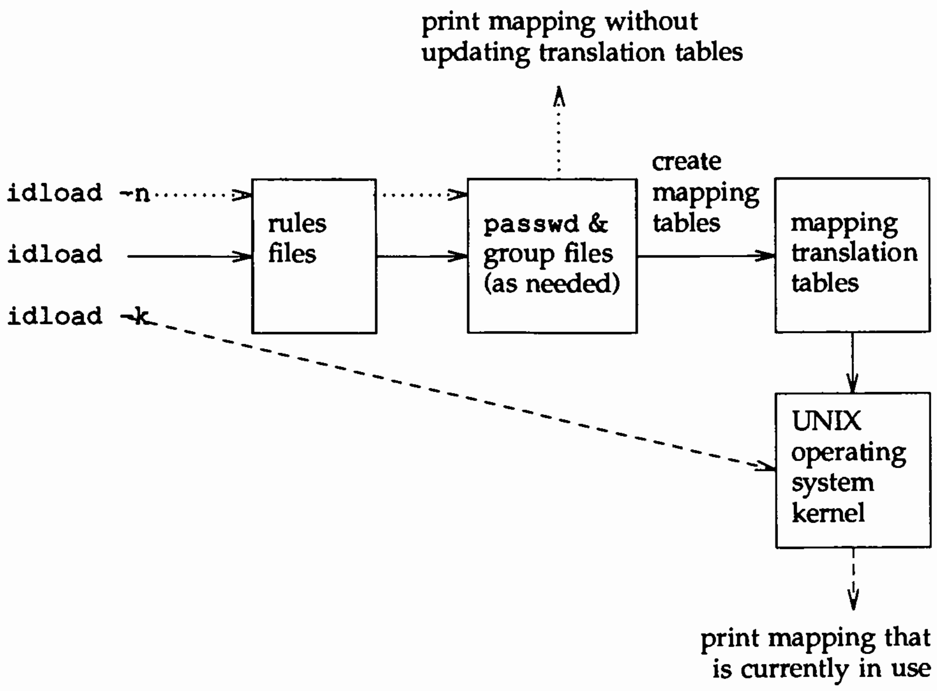
- reads the rules files to determine how you want to set up the mapping
- reads the `/etc/passwd` and `/etc/group` files on your computer, and copies them from other computers, if needed
- creates translation tables

There are two options to `idload` you also may want to use when setting up translation tables.

`idload -n`	Before you run `idload` with no options, the `-n` option lets you do a trial run without actually changing the mapping tables. The result is a listing at your terminal of the tables you would create if you ran `idload` with no options.
`idload -k`	After you run `idload` with no options, the `-k` option lets you read the mapping that is currently in effect on your computer.

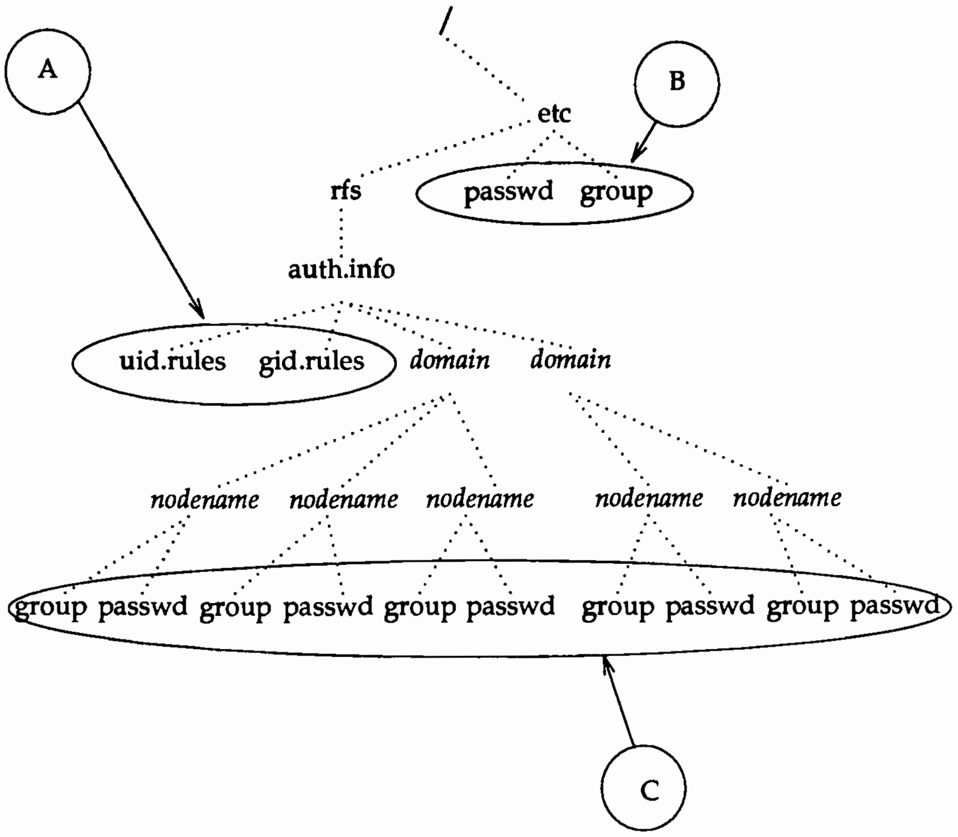
Figure 12-1 illustrates the components described in the previous paragraphs.

Figure 12-1: ID Mapping Components


print mapping without updating translation tables
idload -n
idload
idload -k
rules files
passwd & group files (as needed)
create mapping tables
mapping translation tables
UNIX operating system kernel
print mapping that is currently in use

The files that are involved in setting up ID mapping are illustrated in Figure 12-2.

Figure 12-2: ID Mapping Files



The files used for ID mapping are divided into the following three groups, as shown in Figure 12-2.

A. Rules Files

The `uid.rules` and `gid.rules` files are located in the `/etc/rfs/auth.info` directory. The information you add to these files tells the `idload` command how to create the mapping tables.

B. Local `/etc/passwd` and `/etc/group` Files

The `/etc/passwd` and `/etc/group` files contain lists of the local users on your system. Though you don't modify these files to do ID mapping, you will be interested in the information that is in these files. The first field in each line of your `/etc/passwd` and `/etc/group` files contain local user and group names, respectively. The third field contains the related ID number. If you map by local name in the rules files, these files are read to translate the names into numbers.

C. Remote `/etc/passwd` and `/etc/group` Files

Because mapping translation tables are sets of numbers, if you want to map a remote user by name you must have a copy of the `/etc/passwd` and/or `/etc/group` files for the remote user's machine. These files should be placed in the `/etc/rfs/auth.info/`*domain*/*nodename* directories, where *domain* and *nodename* are replaced by the remote computer's domain and node names, respectively.

Step 1: Create uid.rules File

The following steps describe how to create the rules used to map remote users.

Using any standard file editor (`ed` or `vi`, for example), create or edit the `uid.rules` file in the `/etc/rfs/auth.info` directory. Steps 1A-1D will help you set up a `global` block of mapping information; steps 1E-1H are for `host` blocks of mapping information. The `global` block defines the permissions that will apply to the users on all computers that do not have specific mapping. Note that all lines within a `global` block are optional.

Step 1A: Add the `global` line. (Only add this line if you want to define a block of global information.) The global block of information must begin with the following keyword on a line by itself:

```
global
```

Step 1B: Add a `default` line. (Only add this line if you want to define default information for a global block.) Following the `global` line, you can choose the default permissions that will apply to users from all machines that are not specifically mapped. If this line is not used, the system assumes `default 60001`. (In most cases, `default 60001` is fine.) The two types of default lines are illustrated below.

The line `default transparent` means that each user will have the permissions of the user with the same ID number on your system. (This strategy is most valuable when the `/etc/passwd` files are identical on the two machines.) In the line `default` *local*, the word *local* can be replaced by a local ID number or ID name. This means that any users that are not specifically mapped will have the permissions of a particular user on your system. (Use only one `default` line in a `global` block.)

```
default transparent
      or
default local
```

Step 1C: Add `exclude` line(s). (Only add this line(s) if you want to exclude certain users.) The `exclude` lines let you exclude certain users from having the permissions defined in the default line. For example, if you used `default transparent`, you may want to use `exclude 0` to make sure that the `root` user doesn't have permission to modify the restricted files owned by `root` in your resources. The two types of exclude lines are illustrated below.

In `exclude` *remoteid*, *remoteid* is replaced by a remote user ID number. The remote user would then have the permissions of the guest user (UID 60001) to your resources. The `exclude` *remoteid–remoteid* line lets you specify a range of remote IDs to exclude. For example, `exclude 0-100` could be used to exclude all administrative logins from your default mapping.

```
exclude remoteid
      or
exclude remoteid-remoteid
```

Step 1D: Add map line(s). (Only add this line if you want to map specific users from global machines.) map lines let you take specific remote user IDs and map them into the permissions of one of your local users. The two types of map lines are illustrated below.

In map *remoteid*:*local*, *remoteid* is replaced by a remote user ID number and *local* is replaced by a local user's name or ID number. For example, the line `map 20:root` would map the remote user with ID number 20 into your machine's `root` permissions (UID 0). The line map *remoteid* says give the remote user the permissions of the user with the same ID number on the local system. For example, `map 0` would give `root` from a remote machine the same permissions as `root` on your machine.

```
map remoteid:local
        or
map remoteid
```

Once `global` mapping is done, you may want to add `host` mapping information to the `uid.rules` file. A `host` block defines the permissions that will apply to the users on particular remote machines. You can have one `host` block for each remote machine you want to map specifically. Note that all lines within a `host` block are optional.

Step 1E: Add a `host` line. (Only add this line if you want to define a block of host information.) The host block of information must begin with the following keyword on a line by itself:

```
host domain.nodename
```

where *domain* is replaced by the remote machine's domain name and *nodename* is replaced by the machine's node name.

Step 1F: Add a `default` line. (Only add this line if you want to define default information for a host block.) Following the `host` line, you can choose the default permissions that will apply to all users on the remote machine that are not specifically mapped or excluded. If this line is not used, the system assumes `default 60001`. (In most cases, `default 60001` is fine.) The two types of default lines are illustrated below.

The line `default transparent` means that each user will have the permissions of the user with the same ID number on your system.

(This strategy is most valuable when the `/etc/passwd` files are identical on the two machines.) In the line `default` *local,* the word *local* can be replaced by a local ID number or ID name. This means that any users that are not specifically mapped will have the permissions of a particular user on your system.

```
default transparent
      or
default local
```

Step 1G: Add `exclude` line(s). (Only add this line(s) if you want to exclude certain users from default permissions.) The `exclude` lines let you exclude certain users from having the permissions defined in the default line. For example, if you used `default transparent`, you may want to use `exclude 0` to make sure that the `root` user doesn't have permission to modify the restricted files owned by `root` in your resources. The two types of default lines are illustrated below.

In `exclude` *remote, remote* is replaced by a remote user name or UID number. The *remote* user would then have the permissions of the guest user (UID 60001) to your resources. The `exclude` *remoteid–remoteid* line lets you specify a range of remote IDs to exclude. For example, `exclude 0-100` could be used to exclude all administrative logins from your default mapping.

```
exclude remote
      or
exclude remoteid-remoteid
```

Step 1H: Add `map` line(s). (Only add this line(s) if you want to map particular users.) The `map` lines let you map specific remote users from specific remote machines into the permissions of one of your local users. The two types of map lines are illustrated below.

The `map all` line says to map all user names into the permissions of the users with the same names on your system. In `map` *remote*`:`*local, remote* is replaced by a remote user ID name or number and *local* is replaced by a local user's name or ID number. For example, the line `map 20:root` would map the remote user with ID number 20 into your machine's `root` permissions (UID `0`). The line `map` *remoteid* says give the remote user the permissions of the user with the same

ID number on the local system. For example, `map 0` would give `root` from a remote machine the same permissions as `root` on your machine.

> `map all`
> or
> `map` *remote*:*local*
> or
> `map` *remoteid*

Repeat steps 1E-1H for each specific computer whose users you want to map.

THE `uid.rules` FILE IS NOW COMPLETE!

Figure 12-3 is an example of what your rules file may look like.

Figure 12-3: Example uid.rules File

```
global
default 1000
exclude 0

host peanuts.snoopy
default transparent
exclude 0

host peanuts.linus
default 60001
map 0:100
```

Step 2: Create gid.rules File

The following steps describe how to create the rules used to map remote groups.

Create `gid.rules` file. Using any standard file editor (`ed` or `vi` for example), edit the `gid.rules` file in the `/etc/rfs/auth.info` directory. The `gid.rules` file follows the same format as the `uid.rules` file. Therefore, you can use Steps 1A through 1H to set up the `gid.rules` file, replacing any references to users with references to groups.

NOTE If you create a `uid.rules` file you should also create a `gid.rules` file. Though `idload` will still work without the `gid.rules` file (`idload` will use defaults for mapping groups) a warning message will be produced.

Step 3: Add passwd and group Files

If, when you edited the `uid.rules` and `gid.rules` files, you referenced any remote users by name, you must have copies of the `passwd` file from the remote users' computers in the `/etc/rfs/auth.info/`*domain*/*nodename* directories on your machine. The same is true of the `group` file for groups referenced by name. (Note that `map all` maps by name.)

The best way to obtain these files is as follows:

Step 3A: Obtain files. Have each machine whose users you want to map by name send you its `/etc/passwd` and `/etc/group` files using any standard file transfer method (such as uucp).

Step 3B: Create directories. You must create a separate directory on your machine for each computer whose users and groups you map by name. Each directory must be created using the path `/etc/rfs/auth.info/`*domain*/*nodename*, where *domain* is replaced by the remote machine's domain name and *nodename* is replaced by the remote machine's node name. For example, you must create the following directory for a machine called `linus` in domain `peanuts`:

```
/etc/rfs/auth.info/peanuts/linus
```

Step 3C: Place the remote machines' `passwd` and `group` files in the directory you created in the previous step.

Step 4: Run idload

Step 4A: Run `idload -n`. This command will print a listing of the mapping rules you set up, without creating translation tables. Figure 12-4 is the output from `idload -n` using the `uid.rules` file shown after Step 1H and a `gid.rules` file with simply `default 60001` in the global block.

Figure 12-4: Example Output from idload -n

```
TYPE  MACHINE         REM_ID    REM_NAME     LOC_ID       LOC_NAME

USR   GLOBAL          DEFAULT   n/a          1000         n/a
USR   GLOBAL          0         n/a          60001        guest_id
USR   peanuts.snoopy  DEFAULT   n/a          transparent  n/a
USR   peanuts.snoopy  0         n/a          60001        guest_id
USR   peanuts.linus   DEFAULT   n/a          60001        n/a
USR   peanuts.linus   0         n/a          100          n/a

GRP   GLOBAL          DEFAULT   n/a          60001        n/a
```

Step 4B: Run `idload`. If the output from `idload -n` was acceptable, type the `idload` command with no options to create the translation tables. The `global` and `host` rules for any computer that currently has your resources mounted will immediately take effect. Rules for any other computer that you mapped will take effect as soon as that computer mounts one of your resources.

Step 4C: Run `idload -k`. This will print the mapping that is currently in use on your computer. (Remember that rules for any other computer that you mapped will not be in effect until that computer mounts one of your resources.)

ID MAPPING IS NOW COMPLETE!

Once mapping is set up, it can be changed. You can edit rules files and run `idload` again at any time. It doesn't matter if resources are mounted or if RFS is running.

Multiple Transport Providers (Optional)

This section describes how to run RFS over several transport providers simultaneously. You will find this capability useful in some of the following cases.

- Sharing resources between machines, regardless of the networks they are using. A server machine using multiple transport providers can make its resources available to client machines on different networks (similar to a gateway).
- Establishing a backup network. If your main network goes down, you will still be able to use RFS.
- Better matching network speed to bandwidth. Machines with high traffic between them can use a faster (often more expensive) network, while machines with little traffic between them can use slower (often less expensive) networks.
- Migrating to new networks by initially adding the new network to the server machines, while introducing client machines to the new network as time permits.
- Spreading your networking load over multiple networks.

Compatibility

System V Release 4.0 (SVR4.0) RFS can run simultaneously over multiple transport providers. Because of this added functionality, you will notice minor differences from older System V systems. These differences are described in the following sections.

File Location Changes

Older System V RFS versions allowed using one transport provider and `rfmaster` file at a time. In SVR4.0 RFS, multiple transport providers, with their associated `rfmaster` file, can be used simultaneously.

`rfmaster` files are now in directory `/etc/rfs/`*transport* and contain only entries for their associated transport provider. In addition, each transport provider now has an associated `loc.passwd` file in directory `/etc/rfs/`*transport*. See `rfmaster(4)`.

Finally, `/usr/nserve/rfmaster` and `/usr/nserve/loc.passwd` files are not used in SVR4.0 RFS.

Command-Line Argument Changes

Several commands now have new or modified options allowing transport providers to be specified.

The `rfadmin` command now has a `-t` *transport,transport* ... option that specifies the transport provider(s) to use. Without the `-t` option, all the transport providers are used. See `rfadmin(1M)`.

The `-N` option of the `dname` command now accepts a comma-separated list of transport providers. See `dname(1M)`.

The `-p` option of the `rfstart` command now has an additional format for when multiple transport providers are used. `-p` *address* is the format when a single transport provider is used. `-p` *transport1:address1,transport2:address2* ... is the format when multiple transport providers are used. Using the `-p` *address* format to specify multiple transport providers produces an error message. See `rfstart(1M)`.

Command Output Changes

The `nsquery` "SERVER" field now contains just the machine name. The "ACCESS" field now contains mnemonics, like "rw", instead of "read/write". In addition, the `nsquery` command now displays a "TRANSPORT" field listing transport providers that resources are available on. With multiple transport providers, a resource can be available on more than one transport provider. When this is true, that resource is listed twice in the "TRANSPORT" field. For example, if resource "XYZ" is advertised and two transport providers are used, `nsquery` will list two lines for resource "XYZ"; one for each transport provider. See `nsquery(1M)`.

For each transport provider, the `rfadmin` command now prints one line containing the name of the transport provider. See `rfadmin(1M)`.

Upon initial invocation, the `rfstart` command now prompts for an RFS password for each transport provider, with the the particular transport provider specified in the message prompt.

There is now one global name server process (called "nserve") and separate transport provider-specific name server processes (called "TPnserve"). These additional processes are displayed by the `ps` command.

When appropriate, RFS error messages now contain the name of the transport provider associated with that particular error message. This makes debugging problems much easier in a multi-transport provider environment.

NOTE Local programs that depend on the output of any of the above commands may need to be modified.

Administrative Tips

Administering RFS with multiple transport providers is very similar to administering RFS with a single transport provider, but there are some items that need special attention when running RFS with multiple transport providers.

Error Messages

Most RFS messages that deal with transport providers list the transport provider(s) causing the problem; some messages do not. For example, the message, `warning: no secondary name servers active` does not specify which transport provider(s) have no secondary name servers. For instances like this, it may help to debug the problem by stopping RFS, setting up RFS to use a single transport provider (using the `dname -N transport` command), and then restarting RFS. Doing this for all the transport providers will usually show which transport provider is causing the problem.

Passwords

It is easiest to make all the RFS passwords for a given machine the same. Having different passwords for each transport provider could lead to problems.

Suppose you have two machines, "MACH1" and "MACH2", with two transport providers ("TP1" and "TP2") running on each machine. Since there is only one `/etc/rfs/auth.info/domain/passwd` file, each machine can only have one password for each machine in the domain. Let's assume the RFS passwords on machine "MACH1" were "PASSWD1" and "PASSWD2" for transport providers "TP1" and

"TP2" respectively. If "MACH2"'s `/etc/rfs/auth.info/domain/passwd` file contained "PASSWD1", then any time you tried to mount one of "MACH2"'s resources on "MACH1" using "TP2", it would fail.

Specify Transport Provider
: Currently there is no way to specify the transport provider to use to mount a resource for entries in the `/etc/vfstab` file. If you want certain resources to use specific transport providers, you must mount those resources manually. It will help to put the necessary `mount` commands in a shell script and execute that shell script when you need the resources. Also note that by default, the `mount` command uses the order of the transport providers in the `/etc/rfs/netspec` file (which is the same as the output from `dname -n`) to determine which transport provider to use. If a resource cannot be mounted with the first transport provider, then the second one is used, and so on. Be sure you list them in the order you want them used. In most cases this means listing the fastest or most reliable networks first.

`rfmaster`
: To avoid receiving confusing messages, be sure the primary and secondary name servers listed in the `rfmaster` files are correct for each transport provider. A special case arises when your RFS network contains both System V Release 4 (SVR4) and System V Release 3 (SVR3) machines.

Here's an example of what could happen when you incorrectly set up the `rfmaster` files. Suppose you have two machines in separate domains. One machine is running SVR4 RFS over both the AT&T STARLAN NETWORK and TCP/IP and is in domain D4. The other machine is running SVR3 RFS over only the AT&T STARLAN NETWORK in domain D3, but the machine also has TCP/IP running (i.e., there are two listener processes; one for the AT&T STARLAN NETWORK and one for TCP/IP). The SVR4 machine is the primary name server for domain D4, while the SVR3 machine is the primary name server for domain D3. On the SVR4 machine, the `/etc/rfs/starlan/rfmaster` file correctly contains an entry for the SVR3 machine. Let's also assume the SVR4 machine incorrectly contains an entry in the `/etc/rfs/tcp/rfmaster` file for the SVR3 machine. (This is incorrect because the SVR3 machine is not running RFS on TCP/IP, hence cannot be the primary name server for TCP/IP in its domain.) The output of `nsquery D3.` on the SVR4 machine will incorrectly show that the SVR3 machine's resources are available over both the AT&T STARLAN NETWORK and TCP/IP. Trying to mount one of those resources using TCP/IP will fail, which could lead to much confusion. Removing the incorrect entry from the `/etc/rfs/tcp/rfmaster` file on the SVR4 machine and restarting RFS will resolve the problem. While this example may seem far-fetched, it's easy to incorrectly set up the `rfmaster` files; especially if you copy one transport provider's file to edit for use with another transport provider. You must delete the unneeded lines.

Name Servers

Since each transport provider has its own `rfmaster` file, it's now possible to have different name servers within one domain. For example, "MACH1" could be the the AT&T STARLAN NETWORK primary name server, while "MACH2" could be the TCP/IP primary name server. Although this is allowed, it is usually easier to have the same primary and secondary name servers for all transport providers. For example, if "MACH1" is the primary name server for the AT&T STARLAN NETWORK, it would also be the primary name server for TCP/IP. Doing this makes it easier to administer the RFS network and makes it easier to remember what machine(s) are name servers.

Starting/Stopping RFS

Before a non-primary machine can start RFS, RFS must be configured on the machine and the primary must be up and running RFS.

Is RFS Running?

If you're not sure if RFS is running, type `rfadmin -q` or use `sysadm` as described in the "RFS `sysadm` Interface" section. These will tell you if RFS is or is not running.

Another way is to check if processes related to RFS are active. To do this, type `ps -ef`. These processes should be active:

```
rf_daemon
nserve
rfudaemon
rf_recovery
rf_server
TPnserve
rf_tmo
```

There may be multiple processes, with these names, running. For example, the number of `rf_server` processes depend on the `MINSERVE` parameter. Also, there is a `TPnserve` process for each netspec in `/etc/rfs/netspec`.

Initial RFS Start

The first time you start RFS on a non-primary machine, if you are not using the `sysadm` interface, you should use the `rfstart` command. (A primary can start RFS initially by using `init 3`.)

```
# rfstart -p primary_ns_address
rfstart: Please enter machine password for transport:
```

The *primary_ns_address* is replaced by the network address of the primary name server for your domain. (The network addresses for the AT&T STARLAN NETWORK are in the form *nodename*`.serve`, where *nodename* is replaced by the machine's node name.) If you are running multiple transport providers and reside on a non-primary machine, use the following `rfstart` command:

```
# rfstart -p transport1:primary_ns_address,transport2:primary_ns_address
rfstart: Please enter machine password for transport1:
  rfstart: Please enter machine password for transport2:
```

primary_ns_address is replaced by the network address of the primary name server for each transport provider.

RFS Password

You will be prompted for a password the first time you start RFS. The password must match the password entered when your machine was added to the domain member list in the primary name server (the `rfadmin -a` command). If password verification succeeds, your computer will save this password automatically in `/etc/rfs/`*transport*`/loc.passwd`. You do not have to enter it again.

NOTE When operating RFS over multiple transport providers, there can be multiple `loc.passwd` files. This means that there can be multiple prompts for passwords (one for each `loc.passwd` file).

Likewise, your machine will save the network address of the primary name server. Therefore, the next time you start up RFS, you will be able to do it via `init 3`.

RFS Password Mismatches

Any time you start RFS and your password doesn't match the one on the current domain name server, you will receive a warning, but `rfstart` will NOT fail.

Though RFS will be active, you may have a problem if the *domain*/`passwd` file from the primary domain name server is shared with other machines to use for verification. In that case, your remote mount requests will fail if the passwords don't match. For this reason, it is recommended that RFS passwords always be kept up to date on each computer and the primary name server. If passwords aren't important to you, you can simply enter a carriage return for the passwords on each computer and the primary.

If you do get warnings that your password is out of sync with the current domain name server and you want to fix it, you should handle it differently if the primary is the current domain name server than if the secondary has temporarily taken over.

First find out which machine is the current name server, and whether it is the primary or the secondary, by doing the following:

```
# rfadmin
the acting name server for domain domain on transport is domain.nodename
# cat /etc/rfs/transport/rfmaster
domain  P  domain.nodename
domain  S  domain.nodename
domain.nodename A network_address
domain.nodename A network_address
```

NOTE When operating RFS over multiple transport providers, there will be one line for each transport provider from the **rfadmin** command.

Then, depending on which machine is the current name server, do one of the following:

- secondary is the current name server

  If the primary went down and a secondary took over as domain name server, the secondary may not have a *domain*/passwd file or may have one that is out of date. In this case, do not try to correct your password until the primary takes over as domain name server again.

- primary is the current name server

  Try to correct your password by reentering it with the rfpasswd command. If that doesn't work, follow the sequence shown below, replacing *domain.nodename* with your computer's RFS machine name.

  From the primary name server:

```
# rfadmin -r domain.nodename
# rfadmin -a domain.nodename
Enter password for nodename:
```

From your computer:

```
# rfstop
# rm /etc/rfs/transport/loc.passwd
# rfstart
rfstart: Please enter machine password for transport:
```

You should then make sure that any computer that verifies your computer's password copies the new *domain*/`passwd` file from the primary.

Changing RFS Password

If you want to change your RFS password later, you must use the `rfpasswd` command. This will change your RFS password, both on your computer and on the primary domain name server.

Since changing passwords requires communication with the primary domain name server, RFS must be running on both your computer and the primary domain name server. You cannot change your RFS password if the primary is down and a secondary is the current domain name server.

CAUTION When you change your password, computers that are authenticating your computer may not automatically receive the change. If you are unable to mount a resource from a remote machine after you change your password, check that the remote machine has copied the latest version of your domain's `passwd` file from your primary domain name server.

Automatic RFS Startup (init 3)

There are several steps involved in starting up RFS and sharing resources. To simplify this procedure, a special RFS run level has been defined: run level 3.

When you enter run level 3 using the `init 3` command, RFS is automatically started via `/sbin/rc3` from a shell procedure in your computer's `/sbin/rc3.d` directory. This script starts RFS, shares local resources, and mounts remote resources. When you leave run level 3 (using `shutdown` or `init 2`, for example), RFS processes will be stopped.

You can add your own shell scripts to those that start run level 3. You can also tailor the run level 3 shell scripts to suit the way you use RFS.

NOTE Before you can enter RFS mode, you must have already installed and configured RFS. See Procedure 1 for information on setting up RFS.

Entering Run Level 3

You can go into `init` level 3 in one of three ways:

1. From single-user mode (run level `s`)

   RFS mode is also a multi-user mode. Therefore, when you type `init 3` from single user mode, all multi-user processes (`ttymons`, `cron`, etc.) will be started, followed by RFS mode processes.

2. From multi-user mode (run level 2)

   When `init 3` is run from run level 2, init checks that all multi-user processes are running, then starts the RFS mode processes. (`init 3` will not spawn another process for a level 2 script that is already running.)

3. At boot time

   By default, your system will enter run level 2 at boot time. You can change that to have run level 3 start automatically at boot time by changing the value for `initdefault` in the `/sbin/inittab` file so it reads as follows:

   ```
   is:3:initdefault:
   ```

init 3 Processing

When `init 3` is run, all entries in the `inittab` file that indicate level 3 are started, including `/sbin/rc3`. `/sbin/rc3` executes all shell procedures in `/sbin/rc3.d` that begin with `S`.

The RFS file in `/sbin/rc3.d` is `S21rfs`. This file is linked to the `rfs` file in `/sbin/init.d`, the `K50rfs` file in `/sbin/rc2.d`, and the `K65rfs` file in `/sbin/rc0.d`.

The `S21rfs` shell procedure does the following:

- Validates that the domain name has been defined for your machine.
- Validates that the `rfmaster` file has been created. (This may have been created automatically the first time you ran `rfstart -p` if your machine is not the primary. The latest copy is then sent to your machine from the primary domain name server.)
- Executes the `rfstart` command continuously, with 60 second sleep intervals, until it succeeds or returns a fatal error.
- Executes `/sbin/init.d/adv` to share all system resources you set up in the `/etc/dfs/dfstab` file. (The `/etc/dfs/dfstab` file contains an entire `share` command line for each shared resource.)
- Executes `/usr/sbin/rmountall` to mount all remote resources you listed in your `/etc/vfstab` file. (See "Automatic Remote Mounts" in this chapter for the format of `/etc/vfstab`.) Any remote mount that does not succeed will be tried every 15 minutes via `rmnttry`. The time interval can be changed by modifying the `/etc/rfs/rmnttry` entry in the `crontab` file owned by root.

When you leave run level 3, `/sbin/init.d/rfs` is executed with the `stop` option. This will kill the `rfstart` and `rfudaemon` daemons.

NOTE If for some reason RFS fails to terminate `rfudaemon`, RFS may continue to run in the lower run state. You can always bring down RFS by running `fumount` and `unshare` for each shared resource or `rumountall` to unmount all resources, and `rfstop`.

Changing init 3 Processing

Going to `init 3` makes some assumptions about how you use your RFS system. Here is a description of how to change some of the processing that takes place.

- retry `rfstart`

  By default, `init 3` will keep trying to execute `rfstart` until it succeeds. If you want it to try a limited number of times, you must edit the `/sbin/rc3.d/S21rfs` file. Find the line retries=0 and change the number 0 (try forever) to the number of times you want it to retry.

- retry mounts

  When you enter `init 3`, the system tries separately, every 15 minutes, to mount each resource listed in `/etc/vfstab` until it succeeds or you leave state 3. To change this behavior, login as root and execute `crontab -e`. This will change the time interval `rmnttry` is running at.

Adding RFS Mode Scripts

All RFS files in `/sbin/rc3.d` and other `/etc/*.d` directories are shell procedures, so you can read them to see what they do. You can modify the existing files, though it is preferable to add your own since the delivered scripts may change in future releases. To create your own scripts you should follow these rules:

- Place the file in `/sbin/init.d`.
- Link the file to files in appropriate run level directories using the naming convention described below.
- Have the file accept the `start` and/or `stop` options.

You should name the files using the following conventions:

S*00name*
or
K*00name*

The filenames can be split into three parts:

`S` or `K`	The first letter of each file defines whether the process should be started (`S`) or killed (`K`) upon entering the new run level.
00	The next two characters represent a number from 00 to 99. These numbers indicate the order in which the files will be started (S00, S01, S02, etc.) or stopped (K00, K01, K02, etc).
name	The rest of the filename is the `/sbin/init.d` filename this file is linked to.

Stopping RFS

If you started RFS using `init 3`, you can stop it by going to a lower run state (`init 2`, `init S`, or `shutdown`). If you started RFS using `rfstart`, you can stop it by typing `rfstop` while in run state `init 2` or `init 3`.

Before you can use `rfstop`, you must:

- unshare all your resources (`unshare`)
- unmount everything you have mounted from remote machines (`rumoun-tall`)
- make sure all your shared resources are unmounted from remote machines (can be forced by using `fumount`)

NOTE You can use the `dfmounts` command to determine if any of your shared resources are being used.

These steps will happen automatically when you leave `init` state 3.

If you are the primary name server, you should not stop RFS unless a secondary is up and ready to take over. If the primary goes down and no secondary is available to take over, computers in the domain that are not the primary or a secondary will not be able to start RFS. Computers that are already running RFS will continue to run RFS; however, they will not be able to mount or share new resources.

Sharing Resources

This section describes how to share your local resources with other computers (sharing) on a RFS system and how to use the resources other machines have made available (mounting).

Local Resource Sharing

The `share` command is used to make a local directory (one that physically resides on your machine) accessible to other machines. When you share a directory you must assign it a resource name. This resource must have a unique name within your domain.

When `share` is executed, the resource is registered with your domain name server. Any computer that has access to your domain can find a listing of your resource from your domain's share table (`dfshares` command). A remote computer will not know the exact location of the resource on your machine. All the remote computer will know is its resource name, the short description you assign, that it resides on your computer, the access permissions, and the transport provider over which it is available.

Below are two examples of `share` command lines:

```
share -F rfs -d "Department news" -o ro=peanuts /var/news DNEWS

share -F rfs -d "My devices" -o ro=lucy:linus:doc.compl /dev MDEV
```

The first example shares your `/var/news` directory with read-only permissions under the resource name `DNEWS` to all computers in the `peanuts` domain. The second shares your `/dev` directory as `MDEV` to computers `lucy` and `linus` in your domain and `compl` in the doc domain. See the `share`(1M) manual page for more details.

Automatic Sharing

You can set up all your `share` commands to start automatically when the system enters the RFS state (`init 3`). Do this by placing each full `share` command line in the `/etc/dfs/dfstab` file. As soon as `init 3` successfully starts RFS, all `share` commands in `/etc/dfs/dfstab` will be run.

The following is an example of an `/etc/dfs/dfstab` file to automatically share the two resources shown previously:

```
# cat /etc/dfs/dfstab
share -F rfs -d "Department news" -o ro=peanuts /var/news DNEWS
share -F rfs -d "My devices" -o ro=lucy:linus:doc.compl /dev MDEV
#
```

Resource Security

These are the levels of security that protect your resource once you have shared it:

Verify computers	Only those remote computers that pass your security checks can connect to your machine. You may have indicated that only those machines you have a record of can connect (see `rfstart -v`).
Restrict Resource	You may have shared your resource so only selected remote computers can mount it (see the *client* option of the `share`(1M) manual page).
Map IDs	The permissions remote users will have access to your resources are set on a computer-by-computer basis. In other words, the user and group mappings you set up for a remote computer will apply to any of your resources that computer mounts.
UNIX system security	Normal UNIX system access security, governing read, write, and execute permissions, will apply to any shared resource.

Local Share Table

All shared resources for your computer are contained in your computer's local share table. Any user can use the `share` command with no options to display the local share table. The output will be a listing like the one that follows:

```
# share
CUSTOMER  /usr/bin/cust  read-only   "Atlanta customers"    lucy linus doc.tick
SCCS      /sccs          read/write  "Project Y source"     unrestricted
CALENDAR  /usr/bin/cal   read-only   "UNIX System calendar" peanuts.compgrp
```

The information will match what you entered using the `share` command, with appropriate options, for each resource. Some of the information shown was implied when the resource was shared. Clients are not limited to certain machines (`unrestricted`) when no clients are identified. The *clients* listed in the last field can be:

- computer names in your domain (`lucy`)
- computer names in other domains (`doc.compl`)
- domain names (`peanuts.`)
- `unrestricted` if the resource is not restricted to certain machines (default)

Domain Share Table

All shared resources for your domain are listed in the domain share table on your domain name server. The `dfshares` command is available to all users to list any or all of the shared resources in a domain.

When invoked with no options, `dfshares` will print a list of all shared resources in your domain. Here is an example of output from a `dfshares` command:

```
# dfshares peanuts.lucy
RESOURCE          SERVER  ACCESS  TRANSPORT     DESCRIPTION
peanuts.GRAPHICS  lucy     ro      starlan     Domain files
peanuts.CALENDAR  lucy     ro      starlan     Monthly meetings
peanuts.USERHELP  lucy     rw      starlan     System help information
```

For each available resource, `dfshares` will list the resource name (RESOURCE), the computer that owns the resource (SERVER), access permissions of the resource (ACCESS), the transport provider (TP), and the description of the resource (DESCRIPTION).

NOTE The output from `dfshares` does not indicate whether you have permission to mount the resource.

Shared Resources in Use

You can use the `dfmounts` command to find out what remote computers have mounted your shared resources. This command can print output for all your resources or the one you choose. Here is an example of the output from the `dfmounts` command:

```
# dfmounts
RESOURCE      SERVER   PATH        CLIENTS
DNEWS         lucy     /var/news   peanuts.linus peanuts.lucy
MDEV          lucy     /dev        peanuts.linus
SPECIAL       lucy     unknown     peanuts.charlie
```

The output shows the resource name given when the resource was mounted, the system from which the resource was mounted, the pathname given when the resource was shared, the resource access permissions given to the clients, and a list of systems (clients) that mounted the resource.

NOTE If `unknown` appears in the pathname field, it means you have unshared the resource, but it is still mounted on the listed remote machines.

Unshare

You can unshare any of your computer's shared resources using the unshare command. It will remove the resource from the share tables on your computer and the domain name server.

The domain administrator can use unshare to unshare any resource within the domain.

Unsharing does not remove a currently-mounted resource from a remote computer (see the umount(1M) manual page). It does, however, prevent additional machines from mounting the resource. There are two reasons you may want to use unshare.

1. Before you can unmount (umount or fumount) one of your file systems containing an shared directory, it must be unshared.
2. If you want to restrict a previously shared directory to only local access, you will want to unshare it.

Because share commands can be set up to run automatically in init 3, you may have to remove them if you want them permanently unshared. (See the "Share Resources" section of this chapter for information on how to modify the /etc/dfs/dfstab file.). See the unshare(1M) manual page for further information.

Forced Unmount

You cannot unmount a local file system using the umount command if any part of that file system is mounted remotely. Normally, you should tell each administrator whose machine has mounted such a resource to unmount it. In this way, a resource can be removed in an orderly fashion.

When you have to unmount a local file system immediately, however, you can use the fumount command. This command will remove a remotely-mounted resource from all machines that have mounted it. You should only do this in cases where it is urgent that the resource be removed, because you may be cutting off remote processes that are accessing the resource.

When you execute fumount, this is what happens:

1. The resource is unshared.
2. If the `fumount` command is executed with a grace period of several seconds, the following shell script is run on all client machines currently using the resource.

   ```
   /etc/rfs/rfuadmin fuwarn resource sec
   ```

   By default, this shell script will write to all terminals on all client machines:

   ```
   resource will be disconnected from the system in sec seconds.
   ```

   (You can edit `/etc/rfs/rfuadmin` to tailor the action taken in response to `fumount`.)
3. After the grace period of *sec* seconds, the resource is removed from all remote machines it is mounted on. The following message is then sent to all terminals:

   ```
   resource has been disconnected from the system.
   ```

4. On each client machine, `rfuadmin` then executes `rmount` to queue the resource for mounting. The `rmnttry` command will try to remount the resource every 15 minutes until it succeeds.

See the `rfuadmin(1M)` and `rmount(1M)` manual pages for further information on the processing of these commands.

Remote Resource Mounting

You can attach another computer's shared resource to your system using the `mount` command. You simply choose an existing directory, preferably empty, or create a directory to use as a mount point and `mount` the resource.

When you try to `mount` a remote resource, a request is sent to the computer that shared the resource. If you have permission to mount the resource, the resource will be added to your mount table and connected to the mount point you specified. When used with no options, `mount` lists the remote resources, as well as local file systems, mounted on your computer. See the `mount(1M)` manual page for further details.

Automatic Remote Mounts

You can set up your `mount` commands to run automatically when your computer enters the `init 3` state. You do this by adding mount information to the `/etc/vfstab` file. See the `/etc/vfstab`(4) manual page for the format of remote mounts.

Mounting Guidelines

Below are some guidelines that apply to resources.

- Once you have shared a resource from your computer, you can:
  - Mount a local file system on a subdirectory of the shared resource. The new file system will become part of the shared resource. (You cannot mount directly on the shared mount point, however.)
  - Mount a remote resource on subdirectories of your shared resource. The remote resource you mount will not become part of the resource, however. Only your local users will be able to access it. (Remote users will be able to see this mount point directory, but will get a "multihop" error message if they try to access the directory in any way.)

NOTE You cannot mount a remote resource directly on a shared directory.

- If a resource was shared with read-only permissions, you must mount it read-only. If it was shared read/write, you have a choice of mounting it read-only or read/write.

If RFS is running over multiple transport providers, you can set the NETPATH environment variable to define the order of the transport providers `mount` will use to attempt a connection to a server machine. If the NETPATH variable is not set, `mount` will try each transport provider in turn, as they were specified with the `dname -N` command.

Mounting Rules

There are some rules you must follow to avoid unexpected results when mounting remote resources.

Rule #1 Mounting over basic directories

A directory containing files that define your local machine should not be used as a mount point for a remote resource. This will result in essential local files being inaccessible to your system.

For example, you shouldn't mount a remote `/dev` on your machine's `/dev` directory or you will make your machine's console inaccessible (`/dev/console`). As another example, if you mounted an `/etc` directory on your `/etc` directory, you would cover your local `inittab`, `passwd`, and `mnttab` files, to name a few.

Some other directories that fall into this category are: `/`, `/usr`, `/usr/bin`, `/etc/rfs`, `/usr/net`, and `/usr/lib`.

Rule #2 Mounting spool and work directories

Like Rule #1, Rule #2 has to do with mounting a directory from one computer on the same directory on another. In this case the problem is spool files and workspace directories. Applications such as `uucp` and `lp` can run into problems when multiple machines are trying to create spool files or lock files in the same directory. For example, if you share the `/var/spool/locks` directory, by using a tty device for `uucp` on one machine, you would prevent use of a device of the same name on another machine. Also, mounting `/tmp` can cause collisions among temporary files.

Rule #3 File systems on remote devices

When a remote machine shares a directory containing a device and that device contains a file system, you would not be able to mount the file system by simply mounting the resource containing the device. To access the file system on the remote device, the remote machine would have to mount the device locally, then share that mount point. (You can access the remote as a raw device, however, by simply mounting the resource containing the device.)

Rule #4 Using remote sticky bit programs

Mounting remote resources that contain executable files with the sticky bit on can improve performance of those files. When executed on your machine, the text portion of the sticky bit program will remain in main memory on your machine, thereby reducing the network overhead on future executions. From your perspective as a client, you should be careful not to mount too many sticky bit programs or you could unknowingly gobble up a lot of memory.

If your machine is a server sharing sticky bit files, you should be aware that they are treated differently from strictly local sticky bit files. Before removing sticky bit programs from a shared resource, you must unmount the resource from all client machines (`fumount`), remove the program, then reshare the resource. You should do this to prevent out-of-date text for recompiled or deleted files from remaining in memory on client machines. (See the "Performance Management" and "Security" chapters in the *System Administrator's Guide* for more discussion of sticky bit usage.)

Local Mount Table

You can list the remote resources that are mounted on your computer as you would list local file systems: the `mount` command with no options. Remote resource output from this command will appear in the form:

directory `on` *resource permission* `on` *date*

where *directory* is the name of the directory where the remote *resource* is mounted, the *permission* is `read only/remote` or `read/write/remote`, and *date* is the time and date the resource was mounted.

The following is an example of output from the mount command, with no options. The last two entries in this example are remote resource mounts.

```
# mount
/ on /dev/root read/write/setuid on Thu Aug 31 11:30:14 1989
/proc on /proc read/write on Thu Aug 31 11:30:16 1989
/dev/fd on /dev/fd read/write on Thu Aug 31 11:30:18 1989
/stand on /dev/dsk/c1d0s3 read/write on Thu Aug 31 11:30:19 1989
/var on /dev/dsk/c1d0s8 read/write/setuid on Thu Aug 31 11:30:22 1989
/home on /dev/dsk/c1d0s9 read/write/setuid on Thu Aug 31 11:34:25 1989
/usr on /dev/dsk/c1d0s2 read/write/setuid on Thu Aug 31 11:34:27 1989
/s/codes on LCODE read/write/remote on Thu Aug 31 15:20:09 1989
/s/timing on TEMPO read only/remote on Thu Aug 31 15:20:15 1989
#
```

Remote Resource Disconnected

When a machine that shares its resources with you goes down or the network connection is broken, resources that you have mounted from that server will be disconnected.

An RFS daemon process (`/usr/lib/rfs/rfudaemon`) runs an administrative shell script (`rfuadmin`) that kills all processes using a particular resource and then unmounts that resource. It then queues the remote resource to be mounted by `rmnttry`.

rfudaemon

The `rfudaemon` process is run automatically when RFS is started and continues to run until it is stopped. The `rfudaemon` process listens for one of the following events to occur, then passes that information to the `rfuadmin` administrative shell script.

DISCONNECT	When a link is cut to a remote resource, `rfudaemon` sends a disconnect message and the resource name to the `rfuadmin` shell script.
FUMOUNT	When a resource is unmounted by `fumount` on the server, `rfudaemon` sends a fumount message and the resource name to the `rfuadmin` shell script.

GETUMSG	When a server sends a message that a resource is about to be unmounted with (`fumount`), `rfudaemon` sends a `fuwarn` message, the resource name, and the number of seconds before the resource will be unmounted to the `rfuadmin` shell script.
LASTUMSG	The local machine wants to stop rfudaemon (`rfstop`). This causes `rfudaemon` to exit.

rfuadmin

When links to resources are disconnected, the response to the disconnect is handled at the user level by the `/etc/rfs/rfuadmin` shell script. By editing this shell script, you can tailor the response your system makes when the connection to a remote resource is lost. The `rfudaemon` process starts `rfuadmin` with one of the following arguments.

`disconnect` *resource*

When `rfuadmin` is started by `rfudaemon` with these arguments, `rfuadmin` sends this message to all terminals using the `wall` command:

resource `has been disconnected from the system.`

Then it executes `fuser` to kill all processes using the resource, unmounts the resource (`umount`) to notify the kernel, and starts `rmount` to queue the resource for mounting. The assumption is that the link was either broken by mistake or that as soon as the server makes the resource available again, the client will want to mount it.

`fumount` *resource*

When `rfuadmin` is started by `rfudaemon` with these arguments, a forced unmount has occurred. The processing is similar to a disconnect.

`fuwarn` *resource seconds*

When `rfuadmin` is started by `rfudaemon` with these arguments, `rfuadmin` sends this message to all terminals:

resource `is being removed from the system in` *#* `seconds.`

There are many reasons you may want to change the `rfuadmin` shell script. If access to a resource is lost, you may want to respond by trying to mount another resource. You may want to send different messages when a resource is lost.

NOTE When a resource is disconnected, `rfuadmin` queues the resource using `/usr/sbin/rmount`. The `rmnttry` command tries to remount the resource every 15 minutes. To change this behavior, you must edit `/etc/rfs/rfuadmin` so it no longer does an `rmount`.

Unmounting

You can unmount any remote resource you have mounted with the `umount` command. Before you run `umount`, you should make sure none of your users are using the resource with the `fuser` command. When the `fuser` command is run, it lists the processes on your computer that are accessing a mounted remote resource. It can then be used to kill all processes relating to a resource. See the `umount`(1M) and `fuser`(1M) manual pages for further information.

Sharing Printers

If you have access to other systems via the Remote File Sharing Utilities (RFS), you may want to share printers with those systems by running the print service over RFS. You can do so by following these instructions.

On the server machine:

1. Set up the LP print service on the server machine as you would on any machine. (The server is the computer that does all the spooling.) Make sure that the printer works and that you are able to print text on it.

2. Share /var/spool/lp, /etc/lp, and /var/lp with all the *client(s)* that will be using this printer.
3. In /etc/rfs/auth.info/uid.rules, map the user ID (UID) of lp to itself, so the entry in uid.rules appears as follows: :map lp.

On the client machines:

1. Do not run the scheduler on the client machines. On the client machines you need only lp, lpstat, and other LP print service commands.
2. Mount the resource that was shared by the server on the client's /var/spool/lp, /etc/lp, and /var/lp.

On client machines, the -c option of lp should be used for any user file not in a shared resource. This will force a copy of the file to be sent to the server machine. The LP print service cannot access local files that are not in a shared resource.

Mapping Remote Users

NOTE The "Complex User ID/Group ID Mapping" procedure in this chapter is designed to act as a tutorial for setting up ID mapping. This section provides further reference information to support that section.

Your computer has a set of users, defined in the `/etc/passwd` file. These users can also be members of groups that are defined in the `/etc/group` file. The user and group ID assignments are used by the system to evaluate requests by the user for access to local files, directories, and devices.

When you share your directories with other computers using RFS, you have the ability to define the permissions each remote user will have to your resources. You do this by mapping remote users and groups into the permissions of existing users and groups on your computer. You also have the option of mapping remote users and groups into a special "guest ID" that doesn't map into permissions of any existing users and groups on your system.

If you don't want to map remote users, you don't have to. The default treats all remote users as the special guest ID, which has the ID number of MAXUID plus one. MAXUID is the maximum ID number defined for the system, so MAXUID+1 is always guaranteed not to overlap with any current or future users (by default, MAXUID+1 is 60001).

When a remote user checks the ownership of one of your resources, the user might see another special ID: MAXUID+2 (or 60002). No files will ever be owned by 60002 on the system where a file resides. MAXUID+2 is simply a way of telling remote users that a file or directory is not owned by them or any other users from their system. For example, if all users on a remote system were mapped to 60001 any files created by one of your local users would appear to be owned by user ID 60002.

How Mapping Works

When you set up your remote user and group mapping for a remote computer, you define how requests from users and groups will be handled. This mapping has an impact on the remote users' access to files and directories on your resources, as well as each remote user's view of ownership.

For example, say you map user ID `101` from machine `abc` into user ID `115` on your machine. When `101` from `abc` tries to create a file in a directory of one of your shared resources, your machine will translate the request from `abc`'s `101` into a request from `115`. If local ID `115` has permissions to create a file in that directory, then the file will be created.

If you tried to `stat` the file on your machine (`ls -l`, for example) you would see that user ID `115` was the owner. However, if a `stat` comes from machine `abc`, your machine would do inverse mapping. Therefore, the user from `abc` would see the file as being owned by user ID `101`.

Inverse mapping from the machine that owns the resource (the server) provides the most consistent file system view to a remote user. It could potentially cause confusion, though. Continuing with the example, say that instead of just mapping `101` into `115`, you also mapped `102` from `abc` into `115` on your machine. A file created by `102` would correctly create the file as owned by `115` on your machine. However, when a user from `abc` `stat`s the file, it would always show ownership by the smaller numeric value: `101` user ID.

NOTE This same result would occur if you gave several local user names the same numeric user ID.

If users are confused when files they create do not seem to belong to them, the situation described above could be the reason. This does not cause any problems with each user's ability to access the resource. However, it could break some programs that are dependent on local IDs. The most consistent way to map, however, is one-to-one remote to local IDs.

Multiple Groups

Another feature that can be used to extend your user privileges on the remote machine (server) is multiple groups. This feature allows your single user process to have the combined privileges of all group user processes to which it is a member of, up to to a local system maximum.

An instance of note occurs when a client machine has a larger maximum group setting than a server machine. When this occurs, you, the user on the client machine accessing a file on the server machine, will have your multiple groups privileges limited by the server's lower value of the maximum number of groups to which you can belong. That is, you may be able to access a file on the client side, however, because you are a member of that file's group, you may be denied access to that file on the server side. This is because your user process already belongs to the maximum number of group processes you can belong to, as set by that server machine. If you should recognize this behavior occurring, simply use the `newgrp` command to change your group id to match the group id of the file in question.

The multiple group feature uses the `/etc/rfs/auth.info/gid.rules` file (see the section entitled "Mapping Components" later in this chapter).

NOTE The RFS multiple groups feature is only available when UNIX System V Release 4.0 systems, or later versions, are used.

Mapping Components

You must use the `idload` command to do the user and group mappings. This command reads the user and group mapping rules you create, reads your computer's `/etc/passwd` and `/etc/group` files, if needed, and maps the remote users into your users' permissions. If you are using remote user and group names to map into your computer, you must have access to user and group lists from the remote computers, so `idload` can read the files and translate those names into the appropriate numeric ID numbers.

Rules Files

The rules files you create will tell `idload` how to map remote users. Both files are in `/etc/rfs/auth.info` under the names `uid.rules` and `gid.rules`

Figure 12-5 shows how the user rules file can be structured. The format of the group rules files is exactly the same. All lines in each file are optional.

Figure 12-5: Format of uid.rules and gid.rules Files

```
global
default local_id | transparent
exclude [remote_id-remote_id ...] | [remote_id]
map [remote_id:local ...]

host domain.nodename ...
default local | transparent
exclude [remote_id-remote_id ...] | [remote_id ...] | [remote_name ...]
map [remote:local ...] | remote | all
```

The following notation is used in the previous figure:

local_name = a local user name
local_id = a local user ID number
remote_id = a remote user ID number
remote_name = a remote user name
local = a local name or ID number
remote = a remote name or ID number

A rules file is divided into blocks of information. Each block is either a `global` or `host` block. There is only one `global` block per file, but there can be one `host` block for each computer mapped.

`global` This line starts the block of global information. Each line of definitions after `global` and before the first `host` line will be applied to all computers that are not explicitly defined in `host` blocks. You can use `default`, `exclude`, and `map` inside `global` blocks.

You cannot map or exclude names in `global` blocks. You must use ID numbers.

`host` *domain.nodename ...*
This line starts a block of information for a particular computer. Each line of definitions following this line and

before the next `host` line will be applied to the *domain.nodename* specified. You can use `default`, `exclude`, and `map` inside `host` blocks.

If you want to map more than one computer from a single set of `passwd` and `group` files, you can put several computer names on one line. In this case, `idload` will read the `passwd` and `group` files for the first computer referenced (if you map by name) and use the information in those files for all computers that are referenced.

A computer can only be mapped once in each rules file.

Each of the following lines of information can appear in either a `host` block or a `global` block. A name or an ID should only be mapped once in each block. If one is mapped more than once, the first reference is in effect and the others will produce warning messages from `idload`.

1. `default` *local* | `transparent`

   One `default` line can be put in each block to indicate how to handle remote users and groups that aren't explicitly mapped or excluded.

   `transparent` means use the same numeric ID on your machine that the user had on the remote machine for undefined users. So if a request comes from remote uid `101`, that request will have the permissions of local uid `101`.

   local is replaced by a local user name or ID number. By default, all remote users will be mapped into the permissions of the local user indicated by name or ID. If a default line does not appear in a block, MAXUID+1 permission will be assigned.

2. `exclude` [*remote_id-remote_id*] | [*remote_id*] | [*remote_name*] ...

   Optional `exclude` lines can go into a block to exclude certain users from the default mapping. Zero or more ranges of ID numbers (*remote_id-remote_id*), single *remote_names*, or single *remote_id* numbers can be excluded. (*remote_name* is not available in the global block.)

   A user who is excluded will still have access to your resources but will only have permissions of the MAXUID+1 user. All `exclude` lines must go before any `map` lines in a block.

3. map [*remote:local*] | *remote* | `all`

   You can use map lines in each block to assign local permissions to particular remote users. There are several ways to use the map command. You can set any remote user's permissions to any local user's permissions by either local user *id#* or *name*, separated by a colon (:). By entering a single *remote_id* or *remote_name*, the remote user who matches will have the permissions of the local user of the same ID or name. For example:

   ```
   map mcn
   ```

   would give the remote user mcn the same permissions of the local user mcn.

   The literal entry `all` maps all users by user name into the permissions of users with the same name on your computer.

   Multiple map lines are valid. You cannot map by remote name in `global` blocks.

NOTE: `map all` and mapping by name are not allowed in a global block. `map all` will usually produce warning messages, since multiple administrative logins will have uid 0, and `idload` will try to map each 0 to 0. There is no harm in this.

idload Command

Once the rules files are created, use the `idload` command to read your rules files and create mapping translation tables. When you run `idload`, the rules in `global` blocks and any `host` blocks that have resources currently mounted immediately take effect. All other `host` block rules will take effect when the remote machine mounts one of your resources.

You must then run `idload` for the rules to go into effect. See the `idload`(1M) manual page for further details.

Remote Computer passwd and group Files

If you are mapping remote users by name, you will need lists of these users from each remote computer. These lists should be copies of the `/etc/passwd` and `/etc/group` files from each computer.

If `idload` finds a request for a remote user name in a `host` information block, it will check the directory for that computer for `passwd` and `group` files. The pathname to the remote computer's directory will be `/etc/rfs/auth.info`*/domain/nodename* on your system, where *domain* and *nodename* are replaced by the remote computer's domain and the remote computer's nodename, respectively (unless you overrule this using the `-g` and `-u` options).

NOTE Mapping by name can be a very useful feature. However, if you map only by ID number or local name, and avoid mapping by remote names, you will avoid the need to coordinate distributing and updating remote `passwd` and `group` files and rerunning `idload`.

Example Rules Files

This section describes some strategies you can use to map users. It describes the easiest way to deal with remote user permissions and progresses to more complicated methods. Read through each example to decide what strategy is best for your computer.

No Mapping

If you don't run `idload` to map users, all remote users will have the permissions of the user ID number MAXUID +1. Because there are no users on your system with that user ID number, remote users will only have access to files created by your users that are open to all users.

Mapping Remote IDs

If you map remote users using remote ID numbers and local ID numbers and names, you do not need to get any `passwd` and `group` files from remote computers. The following displays contain some simple examples of mapping that only involve remote ID numbers.

In Figure 12-6, all remote user IDs will be mapped into the same user ID permissions on your computer, except for `root` (ID number 0), which would only have special guest permissions. This would apply to all remote computers.

CAUTION The `exclude 0` line is strongly recommended to prevent possible security breaches from `root` users on other systems.

Figure 12-6: uid.rules File: Setting Global Defaults

```
global
default transparent
exclude 0
```

In Figure 12-7, users have the same permissions as in the previous example, except remote user IDs 0 through 100 will have MAXUID +1 permissions, and any user ID 732 would have the same permission as local user ID 106.

Figure 12-7: uid.rules File: Global Mapping by Remote ID

```
global
default transparent
exclude 0-100
map 732:106
```

In Figure 12-8, the users from computer lucy in domain peanuts will not be mapped by the global rules. Instead, all users will have the permissions of local user mpg except that user IDs 0 through 50 will have MAXUID +1 permissions.

Figure 12-8: uid.rules File: Host Mapping by Remote ID

```
global
default transparent
exclude 0-100
map 732:106

host peanuts.lucy
default mpg
exclude 0-50
```

Mapping Remote Names

If you want to use specific remote user names to map into your local users' permissions you will need to have access to passwd and group files from those computers on your system. Below are some examples of ways you can map remote user names.

map all

If you have the same set of user names on different machines, but the user IDs differ, you may want to use map all as shown in Figure 12-9.

Figure 12-9: uid.rules File: Mapping by Name with map all

```
global
default transparent
exclude 0

host peanuts.lucy
exclude mary 0 uucp
map all
```

In the above example, each user name from computer `lucy` in domain `peanuts` will have the same permissions as the same user name on your computer. The only exceptions will be users `mary`, `root`, and uucp, who will have MAXUID +1 permissions.

map name:name

You can also map particular remote user names into local user names or user IDs on your computer. Figure 12-10 is an example:

Figure 12-10: uid.rules File: Mapping Specific Users by Name

```
global
default transparent
exclude 0

host peanuts.lucy
default transparent
exclude 0
map mcn:jcb ral gwn:103
```

Here all users from the computers will be mapped into their same user ID with the following exceptions. Remote user mcn will have the permission of local user `jcb`, remote user `ral` will have permissions of local user `ral`, remote user gwn will have permissions of local user ID `103`, and `root` will have `MAXUID+1` permissions.

List Current Mapping

There are two ways to list the mapping you have set up: `idload -n` and `idload -k`. The `-n` option inspects the rules files and prints a listing of what would be in effect were you to load them. The `-k` option prints the mapping that is currently in effect in the kernel.

Figure 12-11 shows the result of `idload -n` used for the example shown previously. (The `gid.rules` file simply has the global block set at `default transparent`.) The `-n` option says to print the mapping that is set up in the rules file. You should do this before you run `idload` without options so you can see the mapping that will take effect.

Figure 12-11: Output from idload -n

```
# idload -n
TYPE  MACHINE       REM_ID    REM_NAME    LOC_ID       LOC_NAME

USR   GLOBAL        DEFAULT   n/a         transparent  n/a
USR   GLOBAL        0         n/a         60001        guest_id
USR   peanuts.lucy  DEFAULT   n/a         transparent  n/a
USR   peanuts.lucy  0         n/a         60001        guest_id
USR   peanuts.lucy  100       mcn         105          jcb
USR   peanuts.lucy  102       gwn         103          n/a
USR   peanuts.lucy  191       ral         101          ral

GRP   GLOBAL        DEFAULT   n/a         transparent  n/a
```

If you were to then run `idload`, the mapping shown above would take effect. If you were then to run `idload -k`, and the machine called `peanuts.lucy` did not have a resource mounted, you would see the output in Figure 12-12 printed.

Figure 12-12: Output from idload -k

```
# idload -k
TYPE  MACHINE         REM_ID      REM_NAME      LOC_ID        LOC_NAME

USR   GLOBAL          DEFAULT     n/a           transparent   n/a
USR   GLOBAL          0           n/a           60001         guest_id

GRP   GLOBAL          DEFAULT     n/a           transparent   n/a
```

All mapping to `peanuts.lucy` would be active as soon as one of your resources is mounted there.

NOTE The output from `idload` with the `n` and `k` options could be different if you have changed the rules files, but not yet run `idload` without options.

Domain Name Servers

One machine in each RFS domain must be chosen to be the primary name server and zero or more can be secondary name servers. The duties of these machines are described briefly under the "Name Service" heading in the "Overview" section of this chapter. This section describes the "how-to" of being a name server.

Before you run any of the Domain Name Server tasks, you will want to know which machines are assigned as name servers and which machine is the current name server. To find out the current name server, type:

```
rfadmin
```

To find out the name server assignments, type:

```
cat /etc/rfs/transport/rfmaster
```

The line in the `rfmaster` file that has a `P` in the second field designates the primary name server. If an `S` is in the second field, the entry designates a secondary name server. (Lines with `A` in the second field designate the network address of a primary or secondary.)

Primary Name Server

If your machine is the primary domain name server, you are responsible for maintaining domain information. The "Setting Up Remote File Sharing" section of this chapter describes primary name server responsibilities as you set up your machine and domain. You may want to refer to the following paragraphs in that section if you want to change your RFS configuration after initial configuration.

- Create `rfmaster` File
- Add/Delete Domain Members (Optional)
- Resource Sharing with Other Domains
- Multiple Domain Name Service

NOTE

If you want to change the primary and secondary designations in the `rfmaster` file for a domain that is currently running, you must follow this procedure to make sure those changes are properly put in place.

1) Stop RFS on all primary and secondary domain name servers for the domain (`rfstop` or `init 2`). 2) Change the `rfmaster` file on the old primary and the new primary. 3) Start the primary designated in the new `rfmaster` file (`rfstart` or `init 3`). 4) Start the secondaries designated in the new `rfmaster` file (`rfstart` or `init 3`).

Once changed on the name servers, each individual computer will pick up the change the next time it starts RFS.

Secondary Name Server

Because a secondary is only intended to take over domain name service temporarily, its main responsibility is to pass name server responsibility back to the primary as soon as possible. It does not happen automatically! Most domain maintenance cannot be done while the secondary is acting domain name server. The secondary simply maintains name server information that machines need to mount and share resources.

To pass name server responsibility back to the primary once it is again running RFS, type the following from the secondary:

```
rfadmin -p
```

The `rfadmin -p` command will pass the domain name server information to the primary, or one of the other computers listed in the domain's `rfmaster` file if it can't contact the primary. (Note that name service will automatically be passed off when the current name server goes down.)

Recovery

As a domain name server, computers in your domain, and other domains, rely on your machine for information on domain resources and domain member machines. RFS is designed to recover quickly when communication is cut between machines and the name server. The following sections describe RFS events that can occur and the recovery mechanisms designed to handle them.

Primary Goes Down

All essential domain records are maintained on the primary domain name server. The primary regularly distributes the most critical of these records to secondary domain name servers. (These records do not include files and directories under `/etc/rfs/auth.info`.)

If the primary goes down, domain name server responsibilities are passed to the first secondary name server listed in the `rfmaster` file. The secondary is only intended to take over temporarily. The reason is that a secondary has limited name service capabilities. This is done to maintain the definitive domain records on the primary. Changing the name server does not affect any currently mounted resources.

While a secondary is acting domain name server, these functions cannot be done:

- maintaining domain member lists

  Computers cannot be added or deleted from domain member lists while a secondary is acting domain name server.

- changing RFS passwords

  Neither the secondary nor another computer can change RFS authentication passwords while a secondary is acting domain name server.

The secondary will maintain lists of shared resources for the domain and continue basic name server functions so activities can continue. In most cases, the computers in the domain won't be aware the primary is down. When the primary comes back up, the secondary should pass name server responsibilities back to the primary.

NOTE When a server machine crashes, the domain name server (primary or secondary) still has resources listed as being shared. The domain name server can unshare resources to clean up.

Primary and Secondaries Go Down

If all primary and secondary name servers go down at once, all information on shared resources will be lost. Active mounts and links, however, are not disturbed. The problem is that when the primary comes back up, each computer will still think its resources are shared but the primary will have no record of these shared resources.

As soon as the primary is running, each computer can make sure its shared resources are in sync, with those listed on the primary, by restarting RFS. To restart RFS, you can bring down RFS, bring it back up, and then reshare your resources. This can be done automatically by going from `init 3` to `init 2` to `init 3`.

Monitoring

This section describes the commands used to monitor RFS activity, the reports they produce, and possible action you can take to make sure that your system is operating at peak efficiency. In general, these reports can help you decide if you want to:

- change parameter settings to match the way your system is used
- move resources from machines with heavy RFS traffic to machines with lighter traffic

  (See the "Mounting Guidelines" section of this chapter for special rules relating to sharing sticky bit programs.)

A description of all RFS tunable parameters and suggested initial settings appear at the end of this section.

The `-D` option of `sar` is used to produce RFS-specific information along with standard `sar` reports (`c`, `u`, and `b` options).

NOTE The `sar` command, along with other performance analysis tools, are in the System Performance Analysis Utilities. You must install these utilities before you can use the performance analysis tools.

Remote System Calls (sar -Dc)

Your computer collects data each time a system call sends a message across an RFS network to access a remote file. You can print this information using `sar -Dc` (see Figure 12-13).

The report produced by `sar -Dc` contains the average system calls per second and the average read and write system calls per second (this includes the average characters read and written per second). In System V Release 4.0, RFS does not count the average incoming (in) and outgoing (out) `execs` per second; they are always zero.

Information is divided into three categories: incoming requests (another computer's request for your resources), outgoing requests (your computer's request for a remote resource), or strictly local system calls.

Figure 12-13: Output from sar -Dc

```
$ sar -Dc

UNIX System V lucy 4.0 2 3B2   02/14/89

00:00:04     scall/s  sread/s  swrit/s  fork/s  exec/s  rchar/s  wchar/s
01:00:04
   in            4        1        2             0.00      350      220
   out           3        2        1             0.00      240      300
   local       133       30       12     0.73    1.33    11202     3813
02:00:04
   in            4        1        2             0.00      350      220
   out           3        2        1             0.00      240      300
   local       133       30       12     0.73    1.33    11202     3813
03:00:02
   in            4        1        2             0.00      350      220
   out           3        2        1             0.00      240      300
   local       133       30       12     0.73    1.33    11202     3813
04:00:02
   in            4        1        2             0.00      350      220
   out           3        2        1             0.00      240      300
   local       133       30       12     0.73    1.33    11202     3813

Average
   in            4        1        2             0.00      350      220
   out           3        2        1             0.00      240      300
   local       133       30       12     0.73    1.33    11202     3813
$
```

NOTE Some statistics will not reflect the actual number of messages sent across the network, since the client caching feature allows some remote read requests to be satisfied from data in local buffers. Outgoing scall/s, sread/s, and rchar/s fields include statistics for these read "hits" of remote data in the client cache. Though these reads do not result in actual messages to the remote machine, they are still categorized as outgoing, since they access remote data.

The following paragraphs describe how information from the `sar -Dc` report can be useful to you. If performance is poor, you can see how efficiently system read and write calls to and from your computer are using the RFS network. For in and out system calls, divide the characters read or written by the reads and writes, respectively.

If your computer is attempting more than about 30 remote system calls per second (in and out scall/s), you are probably nearing capacity. Performance problems will probably result from this much demand.

You may want to consider moving resources to machines where they are most in demand. (Use the `fusage` command to determine what resources are being used most heavily.)

RFS Operations (sar -x)

System V Release 4.0 RFS is a file system type that performs file system operations. These operations are reported by the `sar -x` command, as shown below:

```
$ sar -x

UNIX System V ginger 4.0 2 3B2    09/13/89

18:03:38 open/s create/s lookup/s readdir/s getpage/s putpage/s other/s
18:04:38
  in     0.67     0.20     5.48     0.27     0.05     0.00     18.50
  out    0.32     0.07     3.08     0.10     0.12     0.00      1.13
18:05:38
  in     0.57     0.20     5.72     0.27     0.08     0.00     11.61
  out    0.77     0.27     4.99     0.23     0.00     0.00      4.12
18:06:38
  in     0.23     0.00     1.00     0.34     0.00     0.00      7.23
  out    0.58     0.13     5.56     0.53     0.00     0.00      4.18

Average
  in     0.48     0.13     4.03     0.29     0.04     0.00     12.39
  out    0.56     0.16     4.54     0.29     0.04     0.00      3.14
```

The report gives the average number of in and out operations per second. The reported operations are:

open/s	The number of open operations per second.
create/s	The number of create operations per second.
lookup/s	The number of lookup operations per second. The number of out lookups can be compared to the number of namei/s given by `sar -a` to see if a large percentage of the lookups are due to servicing remote requests.
readdir/s	The number of readdir operations per second.
getpage/s	The number of getpage operations per second. These are the page fault operations the server sees from the client.
putpage/s	The number of putpage operations per second. A putpage operation flushes pages to file systems on the server.
other/s	The number of other operations per second. This count does not include the above operations or the close, read and write operations. `sar -Dc` reports the read and write operations as read and write system calls.

High incoming (in) values indicate that your machine is accessing files on server machines frequently. If they are especially high, you may want to determine if there are files you are using remotely that should be copied to your local machine.

High outgoing (out) values indicate that your machine is spending a large amount of its resources servicing clients. If your computer is a server machine, this is to be expected. However, if your computer is not primarily a server machine, the local users could be experiencing degraded response time. `sar -S` and `sar -Du` can verify whether a machine is spending too much of its resources servicing clients. To reduce the time spent servicing remote requests, you may want to move resources that are in demand to another computer (see the `fusage(1M)` manual page for details).

CPU Time (sar -Du)

You can list the percent of total central processing unit (CPU) time spent on system calls from remote computers (%sys remote) with the `sar -Du` command (see Figure 12-14).

Figure 12-14: Output from sar -Du

```
$ sar -Du

UNIX System V lucy 4.0 2 3B2   02/14/89

00:00:04    %usr    %sys    %sys    %wio    %idle
                   local  remote
01:00:04       7      21      10      28      44
02:00:04      11       9      10       4      76
03:00:02       8      18      10      17      57
04:00:02       2       4      10       1      93
05:00:03       1       4      10       1      93
06:00:02       2       5      10       2      91
07:00:02       1       4      10       1      94
08:00:02       2       5      10       2      91
08:20:02      26      16      10      11      48
08:40:02      18      11      10       9      62
09:00:17      25      21      10      13      41
09:20:18      23      21      10      11      45
09:40:20      21      24      10      15      39
10:00:09      21      29      10      17      33
10:20:14      29      28      10      13      31
10:40:18      19      20      10       7      54

Average        9      12      10       8      71
$
```

If the percent of CPU time spent servicing remote system calls is high, your local users may be suffering. (However, if the computer is a server machine, you would expect %sys remote to be high.)

To reduce the time spent servicing remote requests, you may want to place the resource(s) in demand on another computer (see the `fusage`(1M) manual page for details) or limit resource access by changing some of the tunable parameters. (See the section titled "Parameter Tuning.") You may also want to make sure clients are doing I/O in an efficient way (see `sar -Dc`).

Client Caching (sar -Db and sar -C)

The client caching feature of RFS improves RFS performance by reducing the number of times data are retrieved across the network. With client caching, the first read of data will bring the data into local buffers. Once data are in the local buffer, they will remain there so subsequent reads can get the data locally.

Client caching is assigned by default on a system-wide basis (RCACHETIME parameter) and when you mount a remote resource. You will almost always want to take advantage of the improved performance of client caching. There are only two very rare occasions when you may not want to use client caching.

- If buffer space is limited on your system, you may choose to turn off client caching for some resources or the entire system.
- If you are using programs that do their own private network buffering, you may not want to use client caching.

You can produce two `sar` reports to monitor caching activities.

Caching Buffer Usage

The `-b` option of `sar` reports the buffer pool usage for local (disk) reads and writes. The `sar -Db` option reports the same information, plus information on buffer pool usage of locally mounted remote resources (see Figure 12-15).

Figure 12-15: Output from sar -Db

```
$ sar -Db

UNIX System V charlie 4.0 2 3B2    09/03/89

14:37:15 bread/s lread/s %rcache bwrit/s lwrit/s %wcache pread/s pwrit/s
14:37:18
   local    2      40      93      1       3       64      0       0
   remote   1      11      92      1       1        0
14:37:21
   local    2      39      92      1       3       63      0       0
   remote   0      10      94      1       1        0
14:37:24
   local    2      40      93      1       3       64      0       0
   remote   1      12      93      1       1        0

Average
   local    2      40      93      1       3       64      0       0
   remote   1      11      93      1       1        0
```

The fields on this report are as follows:

bread/s	The number of read buffer misses per second. (Each miss results in a read message to the server.)
lread/s	The number of read cache accesses per second.
%rcache	Read cache hit ratio (100 – ((breads)/(lreads) * 100)).
bwrit/s	Number of write buffer misses per second. All writes are sent to the server. Cache buffers affected by the writes are updated (write-through policy). This field indicates the numbers of lwrits that did not require a write-through. If data did not require a write-through it means that no data affected by the lwrit were present in the cache. (The information in this field has no performance implications when comparing using caching versus not using caching.)
lwrit/s	The total write cache accesses per second.

%wcache	Write cache hit ratio (100 – ((bwrits)/(lwrits) * 100)).
pread/s	Not reported for remote use.
pwrit/s	Not reported for remote use.

Cache Consistency Overhead

Information on the overhead related to maintaining cache consistency is listed with `sar -C` (see Figure 12-16).

Figure 12-16: Output from sar -C

```
$ sar -C

UNIX System V charlie 4.0 2 3B2    09/03/89

14:36:56  snd-inv/s  snd-msg/s  rcv-inv/s  rcv-msg/s dis-bread/s  blk-inv/s
14:36:59        0.0        1.1        0.0        1.5         0.0        0.2
14:37:02        0.0        0.6        0.0        0.5         0.0        0.4
14:37:05        0.3        0.6        0.0        0.5         0.0        0.1

Average         0.1        0.9        0.0        0.8         0.0        0.2
```

The fields on this report are as follows:

snd-inv/s	The number of invalidation messages sent by the server per second to inform client machines about changes to server files.
snd-msg/s	The total number of outgoing RFS messages sent per second.
rcv-inv/s	The number of invalidation messages received by the client from the server. Each message informs the client that the contents of one or more of its cache buffers may have been modified by a write on the server. The client machine reacts by invalidating data in the affected buffers, so the buffers can be used for other purposes.

rcv-msg/s	The total number of incoming RFS messages received per second.
dis-bread/s	When an invalidation message is received, caching is turned off until the writing process closes or until a time interval has elapsed (set by the tunable parameter RCACHETIME). This counter tracks the number of buffer reads that normally would be eligible for caching in a resource with caching turned on, but that are not added to the buffer pool because caching for this resource is temporarily turned off. It indicates the penalty of running uncached and provides a basis for tuning the RCACHETIME parameter.
blk-inv/s	The number of buffers removed from the client cache as a result of receiving an invalidation message while a remote file is open or re-opening a remote file that has been modified since the last close on the client.

Server Processes (sar -S)

Every request from a remote computer to access your resources is handled by a server process. When there are too many requests for the servers to handle, they are delayed and placed on the request queue. Requests leave the request queue when servers are available. Information on server availability and requests awaiting service are listed with `sar -S` (see Figure 12-17).

Figure 12-17: Output from sar -S

```
$ sar -S

UNIX System V lucy 4.0 2 3B2    02/14/89

00:00:04  serv/lo-hi   request      request  server   server
               3 - 6     %busy     avg lgth  %avail avg avail
01:00:04     3              0            0      100        3
02:00:04     3              0            0      100        3
03:00:04     4             80            8       20        2
04:00:04     6            100           24        0        0

Average      4             45            8       55        2
$
```

As an administrator you can set the number of server processes available to service remote system calls (see "Parameter Tuning"). There are two server variables you can set: `MINSERVE` and `MAXSERVE`. `MINSERVE` is the number of servers that are initially running to service remote requests.

`MAXSERVE` is the maximum number of servers that may ever exist. If demand goes beyond what the `MINSERVE` servers can handle, extra servers can be dynamically allocated so the total number of servers can be as high as the value of `MAXSERVE`. These processes disappear when they are no longer needed.

Information from `sar -S` can be used to tune your server parameters as shown below.

Too Few Servers

If the receive queue is almost always busy (request %busy), you may want to raise the number of servers. Here is how to decide the parameter to raise:

- Raise the `MAXSERVE` if the total average servers is high.
- Raise the `MINSERVE` if the total average servers is low.

Too Many Servers

If servers are available nearly 100% of the time (server %avail), you may have allocated too many servers. To decide the parameter to lower:

- Check the number of total servers. If this number is near the `MINSERVE` value, you can lower `MINSERVE`. Try reducing it by 50% or by the number of idle servers.
- Check the total servers that are idle. If this number is near the `MAXSERVE` value, you can lower `MAXSERVE`. Try reducing it by 50%.

Resource Usage (fusage)

You can find out how extensively remote computers are using your resources with the `fusage` command. It reports how many kilobytes were read and written from your resources, as categorized by remote computers that mounted the resources. The form for `fusage` for reporting on a resource you have shared is:

`fusage` *advertised resource*

where *advertised resource* is the resource you have shared. `fusage` with no options produces a full report of data usage for all disks and shared directories on your system, as shown in Figure 12-18.

Figure 12-18: Output from fusage

```
# fusage

FILE USAGE REPORT FOR charlie


          /dev/root             /dev/root

                                /
                                        charlie         292 KB

          /proc                 /proc

                                /proc
                                        charlie           0 KB

          /dev/fd               /dev/fd

                                /dev/fd
                                        charlie           0 KB

          /dev/dsk/c1d0s3       /dev/dsk/c1d0s3

                                /stand
                                        charlie        1039 KB

          /dev/dsk/c1d0s8       /dev/dsk/c1d0s8

                                /var
                                        charlie           0 KB

          /dev/dsk/c1d0s2       /dev/dsk/c1d0s2

                 CHUCKUSR         /usr
                                  peanuts.linus         649 KB
                                  peanuts.lucie          51 KB
                                      Sub Total         700 KB

                                /usr
                                        charlie         988 KB

          /dev/dsk/c1d0s9       /dev/dsk/c1d0s9

                                /home
                                        charlie           0 KB
```

If a remote computer's requests for your resources are high, it may be causing performance problems on your computer. With the output from `fusage`, you can see what resources are being particularly hit hard. You may want to move or copy a resource to a computer that is constantly accessing it.

Remote Disk Space (df)

You can use the standard `df` command with a remote resource name to see the space left on the disk on which the remote resource resides. The form of the command to report on a remote resource is:

`df` *resource*

where *resource* is the name of a remote resource mounted on your machine. (`df` with no options will produce information for all mounted remote resources, plus all locally mounted devices.) Figure 12-19 contains an example of the `df` command using resource names as options.

Figure 12-19: Output from df

```
# df USERsrc USERmail

/usr/src     (USERsrc     ):    5436 blocks     2202 files
/var/mail    (USERmail    ):    5436 blocks*    2202 files
```

NOTE

When multiple remote resources are reported that reside on the same disk, all listings of space on that disk, after the first, will be noted with an asterisk.

If you have write permission to a resource, you have as much access to file system space as a user on the system who owns a resource. This command will tell you the potential disk space available for you to write in. (Note that the space reported will only be for the top file system related to each resource.)

Parameter Tuning

There are several parameters you can tune to best suit the way you use RFS. RFS parameters control the amount of resources you devote to Remote File Sharing service. Each network transport provider may also have some tunable parameters that may affect performance characteristics of that particular network. See the network documentation for your network for more details.

All parameters have set default values that should work well for an average system (see the table at the end of this section). The following paragraphs describe these parameters and cases where you may want to change them.

RFS Parameters

RFS parameters define the extent remote computers can use your resources, but they also control your own remote access to remote resources. If the values are too small, you may not be providing enough resources to properly handle your RFS load. Requests for mounts, shares, or even a file could fail if either of those values reach the maximum number allowed for your machine. If these parameters are too large, you could be allocating more system resources than you need to use.

The parameters described below are in `/etc/master.d/rfs`. Read these files for default values. If you change any of these values, see the "System Setup" chapter of the *System Administrator's Guide* for information on how to make the updated parameters take effect.

`NRCVD` (maximum number of receive descriptors)
: Your system creates one receive descriptor for each file or directory being referenced by remote users and one for each process on your machine awaiting response to a remote request. If you limit the number of receive descriptors, you limit the number of local files and directories that can be accessed at a time by remote users. The result of exceeding the limit would be error messages for remote user commands.

`NSNDD` (maximum number of send descriptors)
: For each remote resource (file or directory) your users reference, your system creates a send descriptor. A send descriptor is also allocated for each server process and each message waiting on the receive queue. You can change this value to limit how many remote files and directories your machine can access at a

time. This would, in effect, limit the amount of RFS activities your users can perform. The result of exceeding the limit would be error messages for user commands.

`NSRMOUNT` (server mount table entries)
: Each time a remote machine mounts one of your resources, an entry is added to your server mount table. This number limits the total number of your resources that can be mounted at a time by remote machines.

`MAXGDP` (virtual circuits)
: There are up to two connections (virtual circuits) set up on the network between you and each machine with which you are currently sharing resources. There is one for each computer whose resources you mount and one for each computer that mounts your resources. A virtual circuit is created when a computer first mounts a resource from another, and it is taken down when the last resource is unmounted.

  This parameter limits the number of RFS virtual circuits your computer can have open on the network at a time. It limits how many remote computers you can share resources with at a time. Note that a given network may have a limited number of circuits on any one computer, so this parameter influences the maximum percentage of those that might be used for RFS.

`MINSERVE` (minimum server processes)
: Your system uses server processes to handle remote requests for your resources. This parameter sets how many server processes are always active on your computer. (See the `sar -S` command for information on monitoring server processes.)

`MAXSERVE` (maximum server processes)
: When there are more remote requests for your resources than can be handled by the minimum servers, your computer can temporarily create more. This parameter sets the maximum total server processes your system can have (`MINSERVE` plus the number it can dynamically create).

`NRDUSER` This value specifies the number of receive descriptor `user` entries to allocate. Each entry represents a client machine's use of one of your files or directories. While there is one receive descriptor allocated for each file or directory being accessed remotely (`NRCVD`), there can be multiple receive descriptor `user` entries for each client using the file or directory (`NRDUSER`). These entries are used during recovery when the network or a client goes down. This value should be about one and one half times the value of `NRCVD`.

`RCACHETIME` (caching time off)
This parameter can be used in two ways:

1. to turn off caching for your entire machine;
2. to define the number of seconds that network caching is turned off when a file is modified.

To turn off caching for your entire machine, the parameter must be set to -1.

The second use of `RCACHETIME` requires some explanation. When a write to a server file occurs, the server machine sends invalidation messages to all client machines that have the file open. The client machines remove data affected by the write from their caches. Caching of that file's data is not resumed until the writing processes close the file or until the seconds in this parameter have elapsed.

The assumption is that write traffic is "bursty" and that the first write may be closely followed by other writes. Turning off caching avoids the overhead of sending invalidation messages for subsequent writes.

`RF_MAXKMEM` (limit persistent use of kernel memory)
The default value for this parameter is 0, which means there is no limit. It cannot be applied to allocations that are otherwise limited (NRCVD, NSNDD, NRDUSER, MAXGDP, NSRMOUNT), to stream messages that are not persistent, or to

certain allocations that are directly controlled by an administrator (for example, the number of resource advertisements allowed). A suggested value for a moderately used RFS server machine would be 10000. This means that 10000 bytes of kernel memory could be devoted to persistent use by RFS.

Figure 12-20 lists the RFS parameters and recommended values for different uses of RFS. "Client Only" means that your machine will only be using remote resources, not sharing any from your own machine. "Server Only" means you will only offer your resources to other machines without mounting any remote resources. "Client+Server" means you will both offer local resources and use remote resources.

Figure 12-20: RFS Tunable Parameter Settings

RFS Tunable Parameter Settings
(file `/etc/master.d/rfs`)

Parameter	Client Only			Server Only			Client+Server			Default Value	Size per Entry in Bytes
	2M	3M	4M	2M	3M	4M	2M	3M	4M		
NSRMOUNT	20	20	20	20	20	20	20	20	20	20	28
MAXGDP	10	15	20	24	32	32	24	32	32	24	144
NRCVD	40	60	80	150	250	350	300	400	500	150	40
NRDUSER	0	0	0	225	375	525	450	600	700	250	36
NSNDD	150	250	350	30	30	30	150	250	300	150	144
MINSERVE	0	0	0	3	3	3	3	3	3	3	284
MAXSERVE	0	0	0	6	6	6	6	6	6	6	284
RCACHETIME	10	10	10	10	10	10	10	10	10	10	-

Glossary

share To make a local resource available to other computers using Remote File Sharing. The `share` command is used by administrators to share a resource.

share table
An internal list of available resources. A share table on each computer running RFS has the name of each resource the computer has made available.

automatic share list
A list of local resources that are automatically offered to other computers when Remote File Sharing is started. The list consists of full `share` command lines placed inside the `/etc/dfs/dfstab` file. The command lines are added to `/etc/dfs/dfstab` using the `sysadm` command or by any standard file editor.

automatic mount list
A list of remote resources that are mounted on the local system when Remote File Sharing is started. The list is contained in the `/etc/vfstab` file. (See the `vfstab`(4) manual page in the *System Administrator's Reference Manual* for the format of the file.) Automatic mount information is added to `/etc/vfstab` using any standard file editor.

client A Remote File Sharing computer that is using a remote resource.

client caching
The ability of an RFS computer that is using a remote resource to store remote data blocks in its local buffer pools. This technique improves RFS performance by reducing the number of times data must be read across the network.

client list
When an RFS administrator shares a resource, the administrator can restrict the resource so only certain remote machines can use it. This list of machines is added to the `share` command line when a resource is shared.

client permissions
When an RFS administrator shares a resource, the administrator can set permissions for the resource. The permissions are assigned on the `share` command line. If the permissions are read-only, the client computers can only mount the resource with read

permissions. If they are read/write, a client can mount the resource read/write or read only.

current name server
When a domain is set up, a primary and zero or more secondary domain name servers are assigned. Only one of those machines is actually handling domain name server responsibilities at a time. That machine is referred to as the current name server. Normally, the primary will be the current name server. However, if the secondary has taken over temporarily, it is the responsibility of the secondary's administrator to pass the responsibility back to the primary whenever the primary resumes running RFS. (See also *domain, primary name server,* and *secondary name server.*)

pathname
RFS will ask you for a full pathname to a directory in two instances. When you share a local resource, you will need the full pathname of the directory you are sharing. When you mount a remote resource, you will need the full pathname of the directory where the remote resource should be attached.

domain
A logical grouping of computers in an RFS environment. A domain name is like a telephone area code, acting as an addressing prefix to attach to a computer name or a resource. Assigning a domain name server for a domain provides a central location where lists of resources and network addresses for the group of computers can be stored. Domains also provide a level of security.

domain information (rfmaster)
The primary and secondary name server assignments for a domain are stored in the `/etc/rfs/`*transport*`/rfmaster` file. The primary keeps the definitive copy of this file and distributes it automatically to each computer in the domain when each starts RFS. This file also contains the network address of each name server.

domain member list
The list of the computers that make up an RFS domain. This list is stored in the `/etc/rfs/auth.info/`*domain*`/passwd` file on the primary name server, where *domain* is replaced by the name of the domain. Members are added on the primary using the `rfadmin -a` command.

forced unmount
To unmount one of your local resources from all remote machines that have mounted it. This has the effect of killing all processes that are currently using the resource on all client machines.

ID mapping
To define the permissions remote users and groups have to your shared resources. The tools available for mapping let you set permissions on a per-computer basis and on a global basis. You can then map individual users or groups by ID name or number. When you map IDs for Remote File Sharing, it is easiest to do so with ID numbers since mapping by name requires that you have copies of the remote machines' `/etc/passwd` and `/etc/group` files.

local resource
A directory that resides on your machine that you have made available for other computers running RFS to use. You must share the directory (`share`) to offer it to other computers. If a remote machine mounts your resource, it could have access to all subdirectories, files, named pipes, and devices within your directory (depending on file permissions you set up).

mount The special use of `mount` in RFS is to attach a remote resource to a directory on your system so local users can access the remote resource. Once mounted, the remote resource appears to be just another part of the local UNIX operating system file tree.

network specification
The name that identifies a networking product that is compatible with the AT&T Transport Interface (TI). This is also referred to as the transport provider. RFS requires a transport provider to communicate with other machines. The network specification is used to tell RFS the exact device to use for communications. For example, you would enter `starlan` to tell RFS to use the `/dev/starlan` device if the AT&T STARLAN NETWORK is the transport provider.

network listener
The process used by a transport provider to wait for any type of incoming requests from the network. Once a request comes in, the listener directs it to one of the processes registered with the listener. The process represents a service, such as uucp or RFS.

networking support utilities
A software package that contains the network listener. This package must be installed in order to use Remote File Sharing.

network address
The address by which a computer is known to a particular network. An RFS administrator needs to know the network address of the primary to start RFS for the first time. Address information of other machines is handled internally by RFS. For the AT&T STARLAN NETWORK, the network address is in the form *nodename*.`serve`, where *nodename* is the machine's node name.

node name
The name you assign to your computer to use for communications needs. Networking software, such as Basic Networking Utilities and Remote File Sharing, use this name to identify your machine. A full RFS computer name is *domain*.*nodename*, where *domain* is the name of the computer's RFS domain.

primary name server
The computer that is assigned to provide a central location for addressing and information collection for an RFS domain. Information includes a list of domain members, resources offered by domain members, and optional user ID mapping information. Secondary name servers can be assigned to continue limited name service when the primary is down. For example, a secondary cannot add or delete domain members.

RFS automatic startup
Remote File Sharing can be set to start automatically when your machine is booted. This is done by changing the `initdefault` line in the `/sbin/inittab` file from `2` to `3`. (`init` state `3` is the Remote File Sharing/Multi-user state.)

RFS daemon (rfudaemon)
A daemon process that runs when RFS is running. When network connections to remote resources are broken, `rfudaemon` sends a message to `rfuadmin`, which then executes `rmount` to queue the resource for remounting. (See `rfudaemon`(1M), `rfuadmin`(1M), and `rmount`(1M) manual pages for further information.)

RFS password
A password assigned by the primary name server for every computer in its domain. Each computer must enter its password the first time it starts RFS. After that the password is stored locally in `/etc/rfs/`*transport*`/loc.passwd`. By copying the domain password file from the primary (`/etc/rfs/auth.info/`*domain*`/passwd`), a computer can verify that a remote machine trying to mount its resource is the machine it claims to be.

remote file sharing state (init 3)
The special initialization state used to start RFS. When you type `init 3` or set the `initdefault` line in `/sbin/inittab` to 3, your system will start RFS, share all resources in your automatic share list, and mount all resources in your automatic mount list.

remote resource
A directory that resides on a remote machine that is available for you to connect to using RFS. You must mount the resource (`mount`) to make it available to users on your system. Once you mount the remote resource, your users could have access to all subdirectories, files, named pipes and devices related to your directory (depending on file permissions the remote machine set up).

resource
See *remote resource* and *local resource.*

resource identifier
The name assigned to a resource when it is shared. The name is limited to 14 printable ASCII characters. Slash (/), period (.), and white space may not be used.

secondary name server
A computer designated to take over name server responsibilities temporarily should the primary domain name server fail. The secondary cannot change any domain information. It can, and should, only pass name server responsibility back to the primary when RFS is running on the primary again.

server A Remote File Sharing computer that offers a resource to others.

transport provider

The software that provides a path through which network applications can communicate. RFS can communicate over multiple transport providers simultaneously. These transport providers must meet the AT&T Transport Interface Specification. (AT&T STARLAN NETWORK is one AT&T Transport Interface-compatible network.)

user/group name

The names associated with each local user and group that is allowed access to your computer. This information can be found in the first field of the `/etc/passwd` or `/etc/group` files, respectively. Remote users and groups can be assigned the same permissions as the local users and groups by using RFS ID mapping.

user/group ID number

Every user and group name has a corresponding number that is used by the UNIX operating system to handle permissions to files, directories, devices, etc. These numbers are defined in the third field of the `/etc/passwd` or `/etc/group` files, respectively. Remote users and groups can be assigned the same permissions as the local users and groups by using RFS ID mapping.

RFS INDEX

Index

NETWORK FILE SYSTEM ADMINISTRATOR'S GUIDE

NETWORK FILE SYSTEM ADMINISTRATOR'S GUIDE

Contents

Figures and Tables

13. INTRODUCTION TO NFS

13 Introduction

About This Guide

UNIX® System V Release 4.0 allows you to share files, file hierarchies, and entire file systems across a network using the Network File System (NFS).

This document explains how to administer the System V Release 4.0 implementation of Network File System. It explains how NFS works, and provides the procedures for sharing resources on the network and for accessing resources shared by other machines.

Since this document is an introductory guide, the more sophisticated applications of the commands presented here are not discussed. For additional information about the commands presented in this guide, see the corresponding manual pages.

Audience

This guide is intended to be used by experienced administrators of stand-alone systems. You should be familiar with such administrative tasks as:

- managing local file systems
- assigning access rights
- maintaining system security.

Organization

This guide assumes you are setting up your machine both to share local resources on the network, and to mount remote resource shared by other systems. If you are administering a machine that will mount but not share resources, you can limit your reading to the NFS overview in this chapter, and to the chapters that explain how to mount resources, set up network security, and diagnose problems on an NFS client.

This guide is organized as follows:

- “"Introduction," which describes this guide and provides an introduction to NFS.

- "Setting Up NFS," which directs you to installation instructions and tells you how to start NFS operation and set up automatic sharing and mounting.
- "Sharing and Mounting Resources Explicitly," which tells you how to share and mount resources on an "as needed" basis, using the command line.
- "Obtaining Information," which tells you how to get information about resources that have been shared and mounted across the network.
- "Handling NFS Problems," which provides solutions to problems you may encounter when using NFS.
- "Setting Up Secure NFS," which introduces options to NFS that provide for encryption and other security measures.
- "Using the Automounter," which tells you how to use the automatic mounting facility of NFS.
- "Glossary," which defines many of the terms used in this guide.
- "Appendix: Using the NFS sysadm Interface," which tells you how to set up and administer NFS through a series of menus.

An Introduction to the Network File System

NFS is a service that enables machines of different architectures running different operating systems to share resources across a network. NFS makes it possible for a machine to share local files and directories, and permits remote users to access those files and directories as though they were local to the user's machine.

NFS provides file sharing in a heterogeneous environment, potentially containing many different operating systems; it has been implemented on operating systems ranging from MS-DOS™ to VMS™.

NFS can be implemented on different operating systems because it defines an abstract model of a file system. On each operating system, the NFS model is mapped into the local file system semantics. As a result, normal file system operations, such as read/write, operate in the same way that they operate on a local file system.

The benefits of NFS are many. NFS allows multiple machines to use the same files, so the same data can be accessible to everyone on the network. Storage costs can be kept down by having machines share applications, and database consistency and reliability is enhanced by having all users read the same set of files.

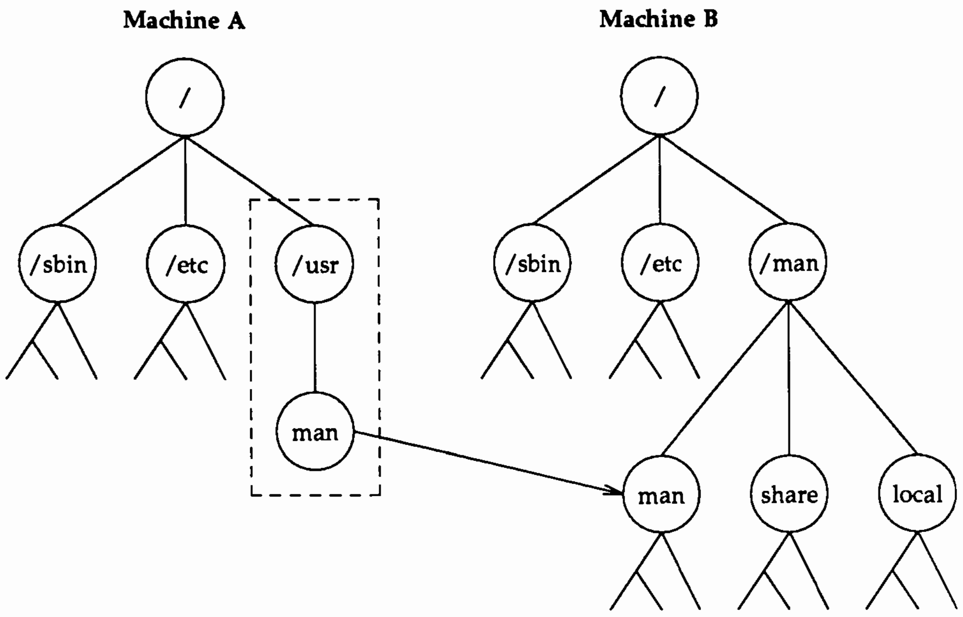
NFS Resources

The objects that can be shared through NFS include any whole or partial directory tree or file hierarchy—including a single file. A machine cannot share a file hierarchy that overlaps one that is already shared. Special device files, as well as ordinary files, can be shared over NFS; however, peripheral devices such as modems and printers cannot be shared.

In most UNIX system environments, a shareable file hierarchy corresponds to a file system or to a portion of a file system; however, NFS works across operating systems, and the concept of a file system may be meaningless in other, non-UNIX environments. Therefore, the term *resource* is used throughout this guide to refer to a file or file hierarchy that can be shared and mounted over NFS.

When you mount a local file system on a local mount point, you mount the entire file system, starting at its root. When you mount a remote resource through NFS, you are not restricted to mounting the entire file system. You can mount any directory or file in the file system tree, and gain access only to that directory or file and anything beneath it. For example, in the illustration, Machine A is sharing its entire /usr file system. If Machine B wants access only to the files and subdirectories in /usr/man on Machine A, it can mount /usr/man, rather than /usr; then, nothing above usr/man on Machine A appears in Machine B's directory tree.

Figure 13-1: Mounting a Remote Resource



It would be illegal for Machine A to share both /usr and /usr/man if both resources reside on the same disk partition. In this case, you would have to share /usr and leave it to network machines to decide whether to mount /usr or /usr/man.

If you want to mount a single file, you must mount the file on a directory. Once it is mounted, you cannot removed it (using rm) or move it to another directory (using mv); you can only unmount it.

NFS Servers and Clients

A machine that makes a local resource available for mounting by remote machines is called a *server*. A machine that mounts a resource shared by a remote machine is a *client* of that machine. Any machine with a disk can be server, a client, or both at the same time.

A server can support a *diskless* client, a machine that has no local disk. A diskless client relies completely on the server for all its file storage. A diskless client can act only as a client—never as a server.

Clients access files on the server by mounting the server's shared resources. When a client mounts a remote resource, it does not make a copy of the resource; rather, the mounting process uses a series of remote procedure calls that enable the client to access the resource transparently on the server's disk. The mount looks like a local mount, and users enter commands just as if the resources were local.

An Overview of NFS Administration

Your responsibilities as an NFS administrator depend on your site's requirements and the role of your machine on the network. You may be responsible for all the machines on your local network, in which case you may be responsible for installing the software on every machine, determining which machines, if any, should be dedicated servers, which should act as both servers and clients, and which should be clients only.

If your site has a network administrator, and you are the administrator of a client-only machine, you may be responsible only for mounting and unmounting remote resources on that machine.

Maintaining a machine once it has been set up involves the following tasks:

- Starting and stopping NFS operation.
- Sharing and unsharing resources as necessary during a server's work session.
- Mounting and unmounting resources as necessary during a work session.
- Modifying administrative files to update the lists of resources your machine shares and/or mounts automatically.
- Checking the status of the network.
- Diagnosing and fixing NFS-related problems as they arise.
- Setting up optional security features provided by Secure NFS.
- Setting up maps to use the optional automatic mounting facility called the Automounter.

14. SETTING UP NFS

14 Setting Up NFS

Introduction

Setting up NFS involves the following tasks:

- Installing NFS software on your servers and clients
- Setting up automatic sharing on servers
- Setting up automatic mounting on clients
- Starting NFS operation.

Remember, a machine can be both a server and a client—both sharing local resources with remote machines and mounting remote resources. Throughout this guide, it is assumed you are setting up your machine as both a server and a client. If you are setting up machines that are to be dedicated servers or clients, follow the instructions for sharing and mounting as appropriate.

Installing the Software

Network File System software is packaged on floppy diskettes and distributed with UNIX System V Release 4.0. If you installed all the software when you received Release 4.0, you already have NFS installed on your system. If you did not install all the utilities, you must install a number of packages before you install NFS. Below are the System V utilities that must be installed before you install NFS.

- Remote Procedure Call (RPC) Utilities
- Network Support Utilities
- TCP/IP Utilities
- Distributed File System Administration Utilities.

Instructions for installing these utilities, plus the NFS diskettes, appear in the System V Release 4.0 *Release Notes*.

Starting and Stopping NFS Operation

By default, NFS becomes operational automatically whenever you enter run level 3. You can take your system to run level 3 by entering the `init` command, or you can set up your system to enter run level 3 immediately on rebooting (for information, see the *System Administrator's Guide*).

NFS allows you to share and mount resources *explicitly*, by entering commands at the command line. You can also set up the system to share and mount resources *automatically*, by editing a number of administration files. If you set up automatic sharing and mounting, a pre-determined set of resources is shared and/or mounted whenever you start NFS operation.

To stop NFS operation, you exit run level 3. When you exit run level 3, the resources you shared and mounted are automatically unshared and unmounted.

To start and stop NFS using the command line, type

```
sh /sbin/init.d/nfs start
```

and

```
sh /sbin/init.d/nfs stop
```

Setting Up Automatic Sharing

If you want to set up your system so that certain files and directories are shared automatically whenever you start NFS operation, you need to edit an administrative file. It is recommended that you set up automatic sharing when you first set up NFS if you need to share the same set of resources on a regular basis. For example, if your machine is a server that supports diskless clients, you need to make your clients' root directories available at all times.

The file you need to edit to set up automatic sharing is `dfstab`, located in `/etc/dfs`. `dfstab` is installed by the Distributed File System Administration Utilities.

The `dfstab` file lists all the resources that your server shares with its clients and controls which clients may mount a resource. If you want to modify `dfstab` to add or delete a resource, or to modify the way sharing is done, simply edit the file with any supported text editor (such as `vi`). The next time the machine enters run level 3, the system reads the updated `dfstab` to determine which resources should be shared automatically.

Each line in the file consists of a `share` command—the same command you enter at the command line to share a resource explicitly. The `share` command is located in `/usr/sbin`. When it is used to share a resource over NFS, it has the following syntax:

`share [-F nfs] [-o` *specific-options*`] [-d` *description*`]` *pathname*

where `-F nfs` indicates that the resource is to be shared through NFS; *specific-options* is a comma-separated list of options that regulates how the resource is shared; *description* is a comment that describes the resource to be shared; and *pathname* is the full name of the resource to be shared, starting at root (/).

If you have only one distributed file system package installed, `nfs` is the default, and you can omit the `-F` option.

If you enter the `-d` option, the description is stored in your `sharetab` file. However, clients will not see the description displayed when they use the `dfshares` command to list the resources shared on your system.

Specific options that can follow the `-o` flag include

`rw`	which shares *pathname* read/write to all clients (by default), except those that are specified under `ro=`.

`ro`	which shares *pathname* read-only to all clients, except those that are specified under `rw=`.
`ro=`*client* [`:`*client*]	which shares *pathname* read-only to the listed clients (overriding `rw`).
`rw=`*client* [`:`*client*]	which shares *pathname* read/write to the listed clients (overriding `ro`).

NOTE You cannot specify both `rw` and `ro` without arguments, and you cannot specify the same client in the `rw=` list and the `ro=` list. If no read/write option is specified, the default is read/write for all clients.

`anon=`*uid*	which allows you to specify a different *uid* for "anonymous" users—users whose *uid* is 0—when accessing *pathname*. By default, anonymous users are mapped to username nobody, which has the uid `UID_NOBODY`. User nobody has ordinary user privileges, not superuser privileges.
`root=`*host* [`:`*host*]	which allows a user from host *host*, whose *uid* is 0, to access *pathname* as root; root users from all other hosts become anon. If this option is not specified, no user from any host is granted access to *pathname* as root.

CAUTION Granting root access to other hosts has far reaching security implications; use the `root=` option with extreme caution. See the discussion below for more information.

`secure`	which allows you to share a resource with additional user authentication required (see the chapter called "Setting Up Secure NFS," later in this guide, for more information).

NOTE Arguments that accept a client or host list—`ro=`, `rw=`, and `root=`—are guaranteed to work over UDP, but may not work over other transport providers.

Under NFS, a server shares resources it owns so clients can mount them. However, a user who becomes the superuser at a client is denied access as the superuser to mounted remote resources. When a user logged in as `root` on one host requests access to a remote file shared through NFS, the user's ID is changed from 0 to the user ID of the username nobody. The access rights of user *nobody* are the same as those given to the public for a particular file. For example, if the public only has execute permission for a file, then user nobody can only execute that file.

When you share a resource, you can permit `root` on a particular machine to have root access to that resource by editing `/etc/dfs/dfstab` on the server, or by specifying the appropriate options on the command line. For example, suppose you want the machine "samba" (but no others) to have superuser access to the shared directory `/usr/src`. You enter the following command in `/etc/dfs/dfstab`, or at the command line.

```
        share -F nfs -o root=samba /usr/src
```

If you want more than one client to have root access, you can specify a list, as follows:

```
        share -F nfs -o root=samba:raks:jazz /usr/src
```

If you want all client processes with uid 0 to have superuser access to `/usr/src`, you enter

```
        share -F nfs -o anon=0 /usr/src
```

anon is short for "anonymous." Anonymous requests, by default, get their user ID changed from its previous value (whatever it may be) to the user ID of username nobody. NFS servers label as anonymous any request from a root user (someone whose current effective user ID is 0) who is not in the list following the `root=` option in the `share` command. The command above tells the kernel to use the value 0 for anonymous requests. The result is that all root users retain their user ID of 0.

Setting Up Automatic Mounting

Once a resource has been shared on a server through NFS, it can be accessed from a client. Mounting can be done automatically when NFS operation begins on the client (when the client enters run level 3), or explicitly, by using the command line during a work session. If you need to mount certain remote resources on a regular basis, it is recommended that you set up automatic mounting when you first set up NFS operation.

The resources to be mounted automatically are determined by the contents of the file `/etc/vfstab`.

NOTE A server can be a client of another server on a local network, in which case, the server's `vfstab` may need to include both local and remote mounts. For information about adding local mounts to the `vfstab` file, see the *System Administrator's Guide.*

Entries in the `/etc/vfstab` file have the following syntax:

special fsckdev mountp fstype fsckpass automnt mntopts

The parameters are

special	which is the name of the server sharing the resource the client wants to mount, followed by a colon and the pathname of the resource to be mounted.
fsckdev	which is the name of a raw device; for a remote mount, the parameter is not applicable, and a hyphen (–) should be entered instead.
mountp	which is the mount point on the client through which the user accesses the resources mounted from the server.
fstype	which is the type of mount taking place. An NFS mount is indicated by `nfs`.
fsckpass	which is the pass number for multiple `fsck`'s. For a remote mount, the parameter is not applicable, and a hyphen (–) should be entered instead.
automnt	which indicates whether the entry should be mounted automatically (yes) or not (no) when the client enters run level 3. [For more information, see the `mountall`(1M) manual page.]

mountopts	which is a list of comma-separated options identical to the options passed to mount(1M).

The contents of /etc/vfstab remain the same until you change them.

Example

To specify that the directory /usr/local of the server dancer should be mounted automatically on the client's directory /usr/local/tmp with read/only permission, you add the following line to the client's vfstab.

```
dancer:/usr/local  -  /usr/local/tmp nfs - yes ro
```

15. SHARING AND MOUNTING RESOURCES

15. SHARING AND MOUNTING RESOURCES

15 Sharing and Mounting Resources Explicitly

Sharing and Unsharing Resources

In addition to sharing and unsharing resources automatically, you can share and unshare *explicitly* during a work session by entering commands at the command line.

To share resources explicitly, you use the `share` and `shareall` commands; to unshare explicitly, you use the `unshare` and `unshareall` commands.

Sharing Resources with the share Command

The `share` command lets you share resources as needed. You should use `share` at the command line when you want to share a resource for a brief period of time, or on an irregular basis.

The `share` command is located in `/usr/sbin` and has the following syntax:

```
share [-F nfs] [-o specific-options] [-d description] pathname
```

where `-F nfs` indicates that the resource should be shared through NFS; *specific-options* is a comma-separated list of options that regulates how the resource is shared; *description* is a comment that describes the resource to be shared; and *pathname* is the full name of the resource to be shared, starting at root (/).

If NFS is the only file sharing package installed on your machine, `nfs` is the default, and you can omit the `-F` option.

For descriptions of the specific options that can follow the `-o` flag, see "Setting Up Automatic Sharing" in the preceding chapter, or see the NFS share(1M) manual page.

If you enter the `-d` option, the description is stored in your `sharetab` file. However, clients will not see the description displayed when they use the `dfshares` command to list the resources shared on your system.

Example 1

To share the directory `/usr` with all the server's clients, you enter the command:

```
share -F nfs /usr
```

The resource is shared read/write by default.

Example 2

If you want to make sure that client dancer can access /usr read-only, you enter

```
        share -F nfs -o rw,ro=dancer /usr
```

Example 3

If you want users or processes on clients jogger and jumper with an effective UID of 0 to access /usr with superuser permission, you enter:

```
        share -F nfs -o root=jogger:jumper /usr
```

Example 4

If you want to permit root access on /usr by any user or process whose user ID is 0, you enter

```
        share -F nfs -o anon=0 /usr
```

You would share a resource this way only if you are in a trusting environment.

Example 5

If you want to see a listing of the remote resources currently available to you through NFS, you enter

```
        share
```

If you have more than one distributed file system package installed, the share command without arguments displays a list of *all* the resources shared on your system, including the resources shared through NFS.

Sharing a Set of Resources with the shareall Command

The shareall command allows you to share a set of resources. To use the command, you first create a file that lists the resources you want to share. The syntax of the entries in your file is the same syntax as the NFS-specific share command and the entries in the dfstab file, as follows:

share [-F nfs] [-o *specific_options*] [-d *description*] [*pathname*]

Once you create the file, you specify it as the input file when you enter the `shareall` command.

If you do not specify an input file, the `/etc/dfs/dfstab` file is used by default. If you enter a dash (–) in place of the name of an input file, the system accepts standard input, which means you can enter a number of `share` commands in succession, then execute the commands all at once by pressing <Ctrl-d>. This saves you from entering one `share` command, waiting for the system to execute the command and return your prompt, then entering another command, and so on.

The syntax of the `shareall` command is as follows:

`shareall [-F nfs] [- | `*file*`]`

Options and operands are described below.

`-F nfs`	which indicates that resources should be shared over NFS; If NFS is the only file sharing package you have installed, you can omit the `-F` option.
–	which indicates that the command should accept standard input.
file	which is the name of the file you created to be your input file.

Example

You need to share the same set of resources on a fairly regular basis, but you do not want the resources shared automatically. You create an input file called `misc` that looks like this:

```
#cat misc
share -F nfs -o ro,rw=art.dept /export/graphics
share -F nfs /usr/man
share -F nfs -o rw,ro=dancer,root=jogger:jumper /local
```

To share the resources listed in the `misc` file, type

```
shareall misc
```

This example assumes that NFS is the only package installed; the `-F` option is omitted from the `shareall` command and from the `share` commands in the input file.

Unsharing Resources with the unshare Command

Resources that are shared either explicitly or automatically (through the `/etc/dfs/dfstab` file) can be unshared at any time by using the command `unshare`.

`unshare` is located in `/usr/sbin`, and has the following syntax:

```
        unshare [ -F nfs ] pathname
```

where `-F nfs` indicates the kind of share, and *pathname* is the full name of the shared resource, beginning with root (/).

Example

If you want to unshare the directory `/usr`, you enter the command

```
        unshare -F nfs /usr
```

Unsharing Resources with the unshareall Command

When you want to unshare all the resources that are currently shared on your system through NFS, use the `unshareall` command, located in `/usr/sbin`. If NFS is the only distributed file system installed on your system, simply enter

```
        unshareall
```

If you have more than one distributed file system installed, include the `-F` option, as follows:

```
        unshareall -F nfs
```

Mounting and Unmounting Resources

Once a resource has been shared on a server through NFS, you can explicitly mount and unmount the resource at any time, using the `mount` and `umount` commands.

Mounting Resources with the mount Command

You can use the `mount` command any time during client operations to mount a remote resource, provided the resource is shared and you can reach the server over the network. You must be the superuser to use `mount`.

NFS supports two types of mounts—hard mounts and soft mounts. If a mount is a hard mount, an NFS request affecting any part of the mounted resource is issued repeatedly until the request is satisfied. When a mount is a soft mount, an NFS request returns an error if it cannot be satisfied, then quits.

Before you issue the `mount` command, use the `mkdir` command to create a mount point for the remote resource. As with a local mount, if you mount a remote resource on an existing directory that contains files and sub-directories, the contents of the directory are obscured.

The syntax of the `mount` command, as it relates to an NFS mount, is:

`mount [-F nfs ] [-o` *specific-options*] *resource mountpoint*

These parameters are

`-F nfs`
: which is is the type of mount you want to perform—in this case, an NFS mount. If the `-F` option is not specified, but *resource* or *mountpoint* is, then `mount` looks in `/etc/vfstab` for the corresponding entry and mounts the resource according to the file system type specification there.

`-o` *specific-options*
: which is a list of options specific to NFS mounts. Some of the options are described below. The full set of options is described on the `mount`(1M) manual page.

  `[rw | ro]` which indicates whether the mounting is to be done read-only or read/write. The default is `rw`.

`[suid	nosuid]`	which indicates whether set-uid and set-gid bits are to be obeyed or ignored on execution, respectively. The default is `suid`.
`[soft	hard]`	which indicates whether the resource should be mounted hard or soft. The default is `hard`.
`[bg	fg]`	which indicates that `mount` should retry either in the background or the foreground if the server does not respond. The default is `fg`.
`intr`	which allows keyboard interrupts to kill a process that is hung while waiting for a response on a hard-mounted file system.
`retry=`*n*	which indicates the number of times `mount` should retry. The default for *n* is 10,000 times.
`timeo=`*n*	which sets the timeout to *n* tenths of a second. The default is 11.

resource	which is in the form *server*:*pathname*, where *server* is the name of the machine sharing the resource and *pathname* is its location on the server.
mountpoint	which is the pathname on the client through which users access the resource.

Resources accessed through the `mount` command stay mounted during a work session, unless you unmount them with the `umount` command. If you exit and re-enter run level 3, the resource will no longer be mounted (unless you edited the `vfstab` file to mount the resource automatically.)

When you mount an NFS resource, it is suggested that you do the following:

- Consider soft mounting resources, so that client processes accessing the resources will not hang if the server goes down.
- Use the `hard` option with any resource you mount read-write. Then, if a user is writing to a file when the server goes down, the write will continue when the server comes up again, and nothing will be lost.

- Use the `nosuid` option with any resource you mount read-write, unless you have good reasons to do otherwise.
- Use the *intr* option with any resource you mount `hard`, so that you can interrupt the current operation if the server goes down.

Example 1

You want to soft mount on-line manual pages from remote machine `dancer` on the local directory `/usr/man`. You want the pages mounted read-only. Type

```
mount -F nfs -o ro,soft dancer:/usr/man /usr/man
```

Example 2

You want to hard mount the resource `/usr/local` from the remote machine `dancer` on the mountpoint `/usr/local/dancer`. You want the resource mounted read-write, with the set-uid bits ignored and the keyboard interrupt enabled. Enter the following, all on one line:

```
mount -F nfs -o hard,nosuid,intr
     dancer:/usr/local /usr/local/dancer
```

Unmounting Resources with the umount Command

The `umount` command allows you to unmount a remote resource, whether the resource was mounted automatically or explicitly. You must be superuser to use `umount`.

The syntax of `umount`, as it relates to NFS, is:

`umount` { *resource* | *mountpoint* }

where *resource* is the name of the server sharing the resource, followed by a colon and the pathname of the resource on the server; and *mountpoint* is the directory on the client where you have the remote resource mounted.

16 Obtaining Information

Introduction

System administrators have at their disposal a variety of tools to obtain information about the status of the networked file system services.

This chapter tells you how to determine what resources are available to you from remote servers, what resources you have shared and mounted on your machine, and what shared resources have been mounted by remote clients.

Browsing Available Resources with the dfshares Command

The command `dfshares` allows you to "browse" remote servers to find out the names of remote resources available to your client machine. The syntax of the command is:

```
        dfshares [ -F nfs ] [ -h ] [ server ... ]
```

where `-F nfs` indicates that resources shared over NFS should be displayed; `-h` indicates that a header should not be printed in the display; and *server* is a list of servers, separated by white space, whose shared resources should be displayed.

If NFS is the only file sharing package you have installed, you can omit the `-F` option.

Unless it is suppressed with the `-h` option, the output from the `dfshares` command is preceded by the header

```
        resource server access transport
```

where *resource* is of the form *server* : *pathname; server* is the name of the server sharing the resource.

At this time, NFS does not populate the *access* and *transport* fields; a hyphen appears in place of access and transport information.

Displaying Shared Local Resources with the share Command

You can obtain a list of all the NFS resources currently shared on your machine by entering

```
share
```

If you have more than one distributed file system package installed, the share command without arguments displays *all* the resources shared on your system, including the resources shared through NFS.

You do not have to be the superuser to use the share command in this capacity.

Monitoring Shared Local Resources with the dfmounts Command

The command `dfmounts` lets you display a list of shared resources by server that tells you which resources are currently mounted over NFS, and by which clients. The syntax of the command is

```
dfmounts [ -F nfs ] [ -h ] [ server ... ]
```

where `-F nfs` indicates that resources mounted over NFS should be displayed; `-h` indicates that a header should not be printed in the display; and *server* is a list of servers, separated by white space, whose shared resources should be displayed.

If NFS is the only file sharing package you have installed, you can omit the `-F` option.

Unless it is suppressed with the -h option, the output from the `dfmounts` command is preceded by the header

```
resource server pathname clients
```

where *resource* is of the form *server:pathname; server* is the name of the server from which the resource was mounted; *pathname* is the pathname of the shared resource as it appears in the second part of *resource;* and *clients* is a comma-separated list of clients that have mounted the resource.

The *server* can be any system on the network. If a *server* is not specified, `dfmounts` displays the resources of the local system that have been mounted by remote clients.

17. HANDLING NFS PROBLEMS

17. HANDLING NFS PROBLEMS

17 Handling NFS Problems

Introduction

This chapter describes problems that may occur on machines using NFS services. Included are

- A summary of NFS sequence of events
- Strategies for tracking NFS problems
- NFS-related error messages.

Before trying to clear NFS problems, you need some understanding of the issues involved. The information in this chapter contains enough technical details to give experienced network administrators a thorough picture of what is happening with their machines. If you do not yet have this level of expertise, note that it is not important to understand fully all the daemons, system calls, and files. However, you should be able to at least recognize their names and functions. Before you read this chapter, familiarize yourself with the following man pages:

```
mount(1M)
share(1M)
mountd(1M)
nfsd(1M)
biod(1M)
```

An Overview of the Mount Process

This section describes the remote mount process. It is not critical that you understand in depth how the daemons mentioned here work; however, you need to know that they exist, because you may need to restart them should they stop for any reason.

Here is the sequence of events when you first enter run level 3.

1. The appropriate file in `/sbin/rc3.d` starts the `mountd` daemon and several `nfsd` daemons (the default is four). The `nfsd` daemons are used in server operation.
2. The same file in `/sbin/rc3.d` executes the `shareall` program, which reads the server's `/etc/dfs/dfstab` file, then tells the kernel which resources the server can share and what access restrictions—if any—are on these files.
3. The appropriate file in `/sbin/rc3.d` starts several (the default is four) `biod` daemons. The `biod` daemons are used in client operation.
4. The same file in `/sbin/rc3.d` starts the `mountall` program, which reads the client's `vfstab` file and mounts all NFS-type files mentioned there in a manner similar to an explicit mount (described below)

Here is the sequence of events when an administrator mounts a resource during a work session, using the command line:

1. The administrator enters a command, such as

   ```
   mount -F nfs -o ro,soft dancer:/usr/src /usr/src/dancer.src
   ```

2. The `mount` command validates that the administrator has superuser permission, that the mount point is a full pathname, and that there is a file `/usr/lib/nfs/mount`. If all three conditions are valid, `/sbin/mount` then passes all the relevant arguments and options to `/usr/lib/nfs/mount`, which takes control of the process (when we say `mount` from now on in this section, we will be referring to this last file.)
3. `mount` then opens `/etc/mnttab` and checks that the mount was not done automatically at the start of the work session.
4. `mount` parses the argument *special* into host `dancer` and remote directory `/usr/src`.

5. mount calls dancer's rpcbind to get the port number of dancer's mountd.
6. mount calls dancer's mountd daemon and passes it /usr/src, requesting it to send a file handle fhandle for the directory.
7. The server's mountd daemon handles the client's mount requests. If the directory /usr/src is available to the client (or to the public), the mountd daemon does a NFS_GETFH system call on /usr/src to get the fhandle, and it sends it to the client's mount process.
8. mount checks if /usr/src/dancer.src is a directory.
9. mount does a mount(2) system call with the fhandle and /usr/src/dancer.src.
10. The client kernel looks up the directory /usr/src/dancer.src and, if everything is in order, ties the file handle to the hierarchy in a mount record.
11. The client kernel looks up the directory /usr/src on dancer.
12. The client kernel does a statfs(2) call to dancer's NFS server nfsd.
13. Mount's mount(2) system call returns.
14. mount opens /etc/mnttab and adds an appropriate entry to the end, reflecting the new addition to the list of mounted files.
15. When the client kernel does a file operation, once the resource is mounted, it sends the NFS RPC information to the server, where it is read by one of the nfsd daemons to process the file request.
16. The nfsd daemons know how a resource is shared from the information sent to the server's kernel by share. These daemons allow the client to access the resource according to its permissions.

Determining Where NFS Service Has Failed

When tracking down an NFS problem, keep in mind that there are three main points of possible failure: the server, the client, or the network itself. The strategy outlined in this section tries to isolate each individual component to find the one that is not working.

The mountd daemon must be present in the server for a remote mount to succeed. Make sure mountd will be available for an RPC call by checking that /sbin/init.d/nfs has lines similar to the following:

```
if [ -x /usr/lib/nfs/mountd ]
then
        /usr/lib/nfs/mountd > /dev/console 2>&1
fi
```

Remote mounts also need a number of nfsd daemons to execute on NFS servers (the default is four). Check the server's file /sbin/init.d/nfs for the following (or similar) lines:

```
if [ -x /usr/lib/nfs/nfsd ]
then
        /usr/lib/nfs/nfsd 4 > /dev/console 2>&1
fi
```

To enable these daemons without rebooting, become the superuser and type:

```
/usr/lib/nfs/nfsd 4
```

The client's biod daemons are not necessary for NFS to work, but they improve performance. Make sure the following (or similar) lines are present in the client's file /sbin/init.d/nfs:

```
if [ -x /usr/lib/nfs/biod ]
then
        /usr/lib/nfs/biod 4 > /dev/console 2>&1
fi
```

To enable these daemons without rebooting, become the superuser and type:

```
/usr/lib/nfs/biod 4
```

Clearing Server Problems

When the network or server has problems, programs that access hard mounted remote files will fail differently than those that access soft mounted remote files. Hard mounted remote resources cause the client's kernel to retry the requests until the server responds again. Soft mounted remote resources cause the client's system calls to return an error after trying for a while. `mount` is like any other program: if the server for a remote resource fails to respond, and the `hard` option has been used, the kernel retries the mount until it succeeds. When you use `mount` with the `bg` option, it retries the mount in the background if the first mount attempt fails.

When a resource is hard mounted, a program that tries to access it hangs if the server fails to respond. In this case, NFS displays the message

`NFS server` *hostname* `not responding, still trying`

on the console. When the server finally responds, the message

`NFS server` *hostname* `ok`

appears on the console.

A program accessing a soft mounted resource whose server is not responding may or may not check the return conditions. If it does, it prints an error message in the form

. . .*hostname* `server not responding: RPC: Timed out`

If a client is having NFS trouble, check first to make sure the server is up and running. From the client, type

```
/usr/sbin/rpcinfo server_name
```

to see if the server is up. If it is up and running, it prints a list of program, version, protocol, and port numbers, similar to the following:

```
program  version  netid      address         service
 100000     3      icmp      0.0.0.0.0.111
 100000     2      icmp      0.0.0.0.0.111
 100000     3       udp      0.0.0.0.0.111
 100000     2       udp      0.0.0.0.0.111
 100000     3       tcp      0.0.0.0.0.111
 100000     2       tcp      0.0.0.0.0.111
 100000     3    ticotsord   sfrjn.rpc
 100000     3    ticots      sfrjn.rpc
 100000     3    ticlts      sfrjn.rpc
 100000     1       udp      0.0.0.0.4.8
 100000     1       tcp      0.0.0.0.4.133
```

If the server fails to print a list, try to log in at the server's console. If you can log in, check to make sure the server is running `rpcbind`.

If the server is up but your machine cannot communicate with it, check the network connections between your machine and the server.

Clearing Remote Mounting Problems

This section deals with problems related to mounting. Any step in the remote mounting process can fail—some of them in more than one way. Below are the error messages you may see and detailed descriptions of the failures associated with each error message.

`mount` can get its parameters explicitly from the command line or from `/etc/vfstab`. The examples below assume command line arguments, but the same debugging techniques work if mounting is done automatically through `/etc/vfstab`.

- `mount: ... server not responding: RPC_PMAP_FAILURE - RPC_TIMED_OUT`

  Either the server sharing the resource you are trying to mount is down, at the wrong run level, or its `rpcbind` is dead or hung. You can check the server's run level by entering (at the server) the command

  ```
  who -r
  ```

  If the server is at run level 3, try going to another run level and back, or try rebooting the server to restart `rpcbind`. Try to log in to the server from your machine, using the `rlogin` command. If you can't log in, but the server is up, try to log in to another remote machine to check your network connection. If that connection is working, check the server's network connection.

- `mount: ... server not responding: RPC_PROG_NOT_REGISTERED`

  `mount` got through to `rpcbind`, but the NFS mount daemon `mountd` is not registered. Check the server's run level and make sure its daemons are running.

- `mount: ...: No such file or directory`

  Either the remote directory or the local directory does not exist. Check the spelling of the directory names. Use `ls` on both directories.

- `mount: not in share list for ...`

  Your machine name is not in the list of clients allowed access to the resource you want to mount. From the client, you can display the server's share list by entering

  `dfshares -F nfs` *server*

  If the resource you want is not in the list, log in to the server and run the `share` command without options.

- `mount: ...: Permission denied`

  This message indicates that the user does not have the appropriate permissions or that some authentication failed on the server. It may be that you are not in the share list (see the preceding error message and explanation), or that the server does not believe you are who you say you are. Check the server's `/etc/dfs/sharetab` file.

- `mount: ...: Not a directory`

  Either the remote path or the local path is not a directory. Check the spelling in your command, and try to run `ls` on both directories.

The `mount` command hangs indefinitely if there are no `nfsd` daemons running on the NFS server. This happens when there is no `/etc/dfs/dfstab` file on the server when it enters run level 3. To clear the problem, restart the `nfsd` daemons by typing

```
/usr/lib/nfs/nfsd 4
```

Fixing Hung Programs

If programs hang while doing file-related work, your NFS server may be dead. You may see the following message on your console:

```
NFS server hostname not responding, still trying
```

This message indicates that NFS server *hostname* is down, or that there is a problem with the server or with the network.

If your machine hangs completely, check the server(s) from which you mounted the resource. If one or more are down, do not be concerned. When the server comes back up, programs resume automatically. No files are destroyed.

If you soft mount a resource and the server dies, other work should not be affected. Programs that time out trying to access soft mounted remote files will fail with `errno ETIMEDOUT`, but you should still be able to access other resources.

If all servers are running, ask someone else using these same servers if they are having trouble. If more than one machine is having problems getting service, there is a problem with the server. Log in to the server. Run `ps` to see if `nfsd` is running and accumulating CPU time (run `ps -ef` a few times, letting some time pass between each call). If not, you may be able to kill and then restart `nfsd`. If this does not work, you have to reboot the server. If `nfsd` is not running, it may be that the server has been taken to a run level that does not support file sharing. Use `who -r` to obtain the server's current run level.

If other systems seem to be up and running, check your network connection and the connection of the server.

If programs on the client are hung but the server is up, NFS requests to the server other than reads and writes are succeeding, and messages of the form

```
xdr_opaque: encode FAILED
```

are appearing on either the client or the server console, you may be requesting more data than the underlying transport provider can provide. Try remounting the file system using the `-o rsize=`*nnn*`,wsize=`*nnn* options to `mount`(1M) to restrict the request sizes the client will generate.

Fixing a Machine That Hangs Part Way Through Boot

If your machine boots normally, then hangs when it tries to mount resources automatically, most likely one or more servers are down. Use `init` to go to single user mode or to a run level that does not mount remote resources automatically. Then start the appropriate daemons in the background and use the mount command to mount each resource usually mounted automatically through the /etc/vfstab file. By mounting resources one at a time, you can determine which server is down. To restart a server that is down or hung, see the preceding section.

If you cannot mount any of your resources, most likely your network connection is bad.

Improving Access Time

If access to remote files seems unusually slow, type

```
ps -ef
```

on the server to be sure that it is not being affected adversely by a runaway daemon. If there is nothing unusual in the display, and other clients are getting good response, make sure your biod daemons are running. At the client, type

```
ps -ef | grep biod
```

Look for biod daemons in the display, then enter the command again. If the biods do not accumulate excessive CPU time, they are probably hung. If they are dead or hung, follow these steps:

1. Kill the daemon processes by typing:

   kill -9 *pid1 pid2 pid3 pid4*

2. Restart the biod daemons by typing:

   ```
   /usr/lib/nfs/biod 4
   ```

If the biods are running, check your network connection. The command netstat -i can help you determine if you are dropping packets; for more information, see the netstat(1M) manual page.

18 Setting Up Secure NFS

Introduction

NFS is a powerful and convenient way to share resources on a network of different machine architectures and operating systems. However, the same features that make sharing resources through NFS convenient also pose some security problems. An NFS server authenticates a file request by authenticating the machine making the request, but not the user. If superuser privilege is not restricted when a resource is shared, a client user could run su and impersonate the owner of a file.

Given root access and a good knowledge of network programming, anyone is capable of injecting arbitrary data into the network, and picking up any data from the network. The most dangerous attacks are those involving the injection of data, such as impersonating a user by generating the right packets, or recording conversations and replaying them later. These attacks affect data integrity. Attacks involving passive eavesdropping—merely listening to network traffic without impersonating anybody—are not as dangerous, since data integrity is not compromised. Users can protect the privacy of sensitive information by encrypting data that goes over the network.

A common approach to network security problems is to leave the solution to each application. A better approach is to implement a standard authentication system at a level that covers all applications.

System V Release 4.0 includes an authentication system at the level of Remote Procedure Call (RPC)—the mechanism on which NFS is built. This system, known as Secure RPC, greatly improves the security of network environments and provides additional security to NFS. The security features it provides to NFS are known as Secure NFS.

Because Secure RPC is at the core of Secure NFS, it is necessary to understand how authentication works in RPC to understand Secure NFS. This chapter first presents an overview of Secure RPC, then tells you how to set up Secure NFS. Following the set up procedure are a few important points you should be aware of if you plan to use Secure NFS.

An Overview of Secure RPC

The goal of Secure RPC is to build a system at least as secure as a time-shared system. The way a user is authenticated in a time-sharing system is through a login password. With DES authentication, the same is true. Users can log in on any remote machine, just as they can on a local terminal, and their login passwords are their passports to network security. In time-sharing, the trusted person is the system administrator, who has an ethical obligation not to change a password in order to impersonate someone. In secure RPC, the network administrator is trusted not to alter entries in a database that stores "public keys."

You need to be familiar with two terms to understand an RPC authentication system: *credentials* and *verifiers*. Using ID badges as an example, the credential is what identifies a person: a name, address, birth date, and so on. The verifier is the photo attached to the badge: you can be sure the badge has not been stolen by checking the photo on the badge against the person carrying it. In RPC, the client process sends both a credential and a verifier to the server with each RPC request. The server sends back only a verifier, since the client already knows the server's credentials.

RPC's authentication is open-ended, which means that a variety of authentication systems may be plugged into it. Currently, there are two such systems: UNIX and DES.

When UNIX authentication is used by a network service, the credentials contain the client's machine-name, UID, `gid`, and group-access-list, but the verifier contains nothing. Because there is no verifier, a root user could deduce appropriate credentials, using commands such as su. Another problem with UNIX authentication is that it assumes all machines on a network are UNIX machines. UNIX authentication breaks down when applied to other operating systems in a heterogeneous network.

To overcome the problems of UNIX authentication, Secure RPC uses DES authentication—a scheme that employs verifiers, yet allows Secure RPC to be general enough to be used by most operating systems.

DES Authentication

DES authentication uses the Data Encryption Standard (DES) and public key cryptography to authenticate both users and machines in the network. DES is a standard encryption mechanism; public key cryptography is a cipher system that involves two keys: one public and one private.

The security of DES authentication is based on a sender's ability to encrypt the current time, which the receiver can then decrypt and check against its own clock. The timestamp is encrypted with DES. Two things are necessary for this scheme to work: 1) the two agents must agree on the current time, and 2) the sender and receiver must be using the same encryption key.

If a network runs a time synchronization program, then the time on the client and the server is synchronized automatically. If a time synchronization program is not available, timestamps can be computed using the server's time instead of the network time. The client asks the server for the time before starting the RPC session, then computes the time difference between its own clock and the server's. This difference is used to offset the client's clock when computing timestamps. If the client and server clocks get out of sync to the point where the server begins to reject the client's requests, the DES authentication system resynchronizes with the server.

The client and server arrive at the same encryption key by generating a random *conversation key*, then using public key cryptography (an encryption scheme involving public and secret keys) to deduce a *common key*. The common key is a key that only the client and server are capable of deducing. The conversation key is used to encrypt and decrypt the client's timestamp; the common key is used to encrypt and decrypt the conversation key.

A Secure RPC Client/Server Session

Here is the series of transactions in a client/server session using Secure RPC.

Step 1 Sometime prior to a transaction, the user runs a program that generates a *public key* and a *secret key*. (Each user has a unique public key and secret key.) The public key is stored, in encrypted form, in a public database; the secret key is stored, also in encrypted form, in a private directory.

Step 2 The user logs in and runs the `keylogin` program (or the `keylogin` program may be included in `/etc/profile` so that it runs automatically whenever the user logs in). The `keylogin` program prompts the user for a secure RPC password and uses the password to decrypt the secret key. The `keylogin` program then passes the decrypted secret key to a program called the Keyserver. (The Keyserver is an RPC service with a local instance on every machine.) The Keyserver saves the decrypted secret key, and waits for the user to initiate a transaction with a server.

Step 3 When the user initiates a transaction with a server

a. the Keyserver randomly generates a *conversation key*.

b. the kernel uses the conversation key to encrypt the client's timestamp (among other things).

c. the Keyserver looks up the server's public key in the public database.

d. the Keyserver uses the client's secret key and the server's public key to create a *common key*.

e. the Keyserver encrypts the conversation key with the common key.

Step 4 The transmission including the timestamp and the conversation key is then sent to the server. The transmission includes a credential and a verifier. The credential contains three things:

- the client's machine name.
- the conversation key, encrypted with the common key.
- a "window," encrypted with the conversation key.

The window is the difference the client says should be allowed between the server's clock and the client's timestamp. If the difference between the server's clock and the timestamp is greater than the window, the server should reject the client's request. (For secure NFS, the window currently defaults to 30 minutes.)

The client's verifier contains

- the encrypted timestamp.
- an encrypted verifier of the specified window, incremented by 1.

The reason the window verifier is needed is this: Suppose somebody wants to impersonate a user and writes a program that, instead of filling in the encrypted fields of the credential and verifier, just stuffs in random bits. The server will decrypt the conversation key into some random key, and use it to try to decrypt the window and the timestamp. The result will just be random numbers. After a few thousand trials, however, there is a good chance that the random window/timestamp pair will pass the authentication system. The window verifier makes guessing the right credential much more difficult.

Step 5 When the server receives the transmission from the client

1. the Keyserver local to the server looks up the the client's public key in the public database.
2. the Keyserver uses the client's public key and the server's secret key to deduce the common key—the same common key computed by the client. (No one but the server and the client can calculate the common key, since doing so requires knowing one secret key or the other.)
3. the kernel uses the common key to decrypt the conversation key.
4. the kernel calls the Keyserver to decrypt the client's timestamp with the decrypted conversation key.

Step 6 After the server decrypts the client's timestamp, it stores four things in a credential table:

- the client's machine name.

- the conversation key.
- the window.
- the client's timestamp.

The server stores the first three things for future use. It stores the timestamp to protect against replays. The server will only accept timestamps that are chronologically greater than the last one seen, so any replayed transactions are guaranteed to be rejected.

Step 7 The server returns a verifier to the client, which includes

- the index ID, which the server records in its credential table.
- the client's timestamp minus one, encrypted by conversation key.

The reason for subtracting one from the timestamp is to insure that it is invalid and cannot be reused as a client verifier.

Step 8 The client receives the verifier and authenticates the server. The client knows that only the server could have sent the verifier, since only the server knows what timestamp the client sent.

Step 9 The client returns the index ID to the server in its second transaction and sends another encrypted timestamp.

Step 10 The server sends back the client's timestamp minus 1, encrypted by the conversation key.

With every transaction after the first, the client sends its index ID and another encrypted timestamp, and the server returns the timestamp minus 1.

NOTE Implicit in these procedures is the name of caller, who must be authenticated in some manner. The Keyserver cannot use DES authentication to do this, since it would create a deadlock. The Keyserver solves this problem by storing the secret keys by UID, and only granting requests to local root processes. The client process then executes a set-UID process, owned by root, which makes the request on the part of the client, telling the Keyserver the real UID of the client.

Administering Secure NFS

To use Secure NFS, all the machines for which you are responsible must have a domain name. A *domain* is an administrative entity, typically consisting of several machines, that joins a larger network. If you are running YP, you should also establish the YP name service for the domain.

With UNIX authentication, the name of a domain is the UID. UIDs are assigned per domain. A problem with this scheme is that UIDs clash when domains are linked across the network. Another problem with UNIX authentication has to do with superusers; with UNIX authentication, the superuser ID (uid 0) is assigned one per machine, not one per domain, which means that a domain can have multiple superusers—all with the same uid.

DES authentication corrects these problems by using netnames. A *netname* is simply a string of printable characters created by concatenating the name of the operating system, a user ID, and a domain name. For example, a UNIX system user with a user ID of 508 in the domain `eng.acme.COM` would be assigned the following netname: `unix.508@eng.acme.COM`. Because user IDs are unique within a domain, and because domain names are unique on a network, this scheme produces a unique netname for every user.

To overcome the problem of multiple super-users per domain, netnames are assigned to machines as well as to users. A machine's netname is formed much like a users—by concatenating the name of the operating system and the machine name with the domain name. A UNIX machine named `hal` in the domain `eng.acme.COM` would have the netname `unix.hal@eng.acme.COM`.

1. Assign your domain a domain name, and make the domain name known to each machine in the domain, following the instructions in the *TCP/IP Network Administrator's Guide*.

2. Either establish public keys and secret keys for your clients' users using the `newkey` command, or have each user establish his or her own public and secret keys using the `chkey` command.

   When public and secret keys have been generated, the public keys are stored in the `publickey` database, and users' secret keys are stored in `/etc/keystore`. Secret keys for root users are stored in `/etc/.rootkey`.

NOTE For information about these commands, see the newkey(1M) and the chkey(1) manual pages.

3. If you choose, put the keylogin program in etc/profile so that it runs automatically whenever a user logs in. Otherwise, make sure users know to run keylogin when they log in.
4. If you are running YP, verify that the ypbind daemon is running and that there is a ypserv running in the domain.
5. Verify that the keyserv daemon (the Keyserver) is running by typing

   ```
   ps -ef | grep keyserv
   ```

   If it isn't running, start the Keyserver by typing

   ```
   /usr/sbin/keyserv
   ```

6. Edit the /etc/dfs/dfstab file and add the secure option to the appropriate entries (those that indicate resources you want clients to mount using DES authentication).
7. On each client machine, edit /etc/vfstab to include secure as a mount option in the appropriate entries (those that indicate resources that should be mounted using DES authentication).

NOTE If a client does not mount as secure a resource that is shared as secure, everything works, but users have access as user nobody, rather than as themselves.

If you are not running YP, all users must keep their secret keys synchronized with their login passwords by invoking chkey if they change their entries in the password database.

When you reinstall, move, or upgrade a machine, remember to save /etc/keystore and /etc/.rootkey.

Important Considerations

The following are points you should be aware of if you plan to use Secure RPC:

- If a server crashes when no one is around (after a power failure for example), all of the secret keys that are stored on the system are wiped out. Now no process is able to access secure network services, or mount an NFS file system. The important processes at this time are usually root processes, so things would work if root's secret key were stored away, but nobody is around to type the password that decrypts it. If the machine has a local disk, the solution to the problem is to store root's decrypted secret key in a file, which the Keyserver can read.
- Some systems boot in single-user mode, with a `root` login shell on the console and no password prompt. Physical security is imperative in such cases.
- A problem with diskless clients is that diskless machine booting is not totally secure. It is possible for somebody to impersonate the boot-server, and boot a devious kernel that, for example, makes a record of your secret key on a remote machine. Secure NFS provides protection only after the kernel and the Keyserver are running. Before that, there is no way to authenticate the replies given by the boot server. This is not considered a serious problem, because it is highly unlikely that somebody would be able to write this compromised kernel without source code. Also, the crime is not without evidence. If you polled the network for boot-servers, you would discover the devious boot-server's location.
- Most set-UID programs are owned by root; since root's secret key is always stored at boot time, these programs will behave as they always have. If a set-UID program is owned by a user, however, it may not always work. For example, if a set-UID program is owned by `dave`, and `dave` has not logged into the machine since it booted, then the program would not be able to access secure network services.
- If you log in to a remote machine (using `login`, `rlogin`, or `telnet`) and use `keylogin` to gain access, you give away access to your account. This is because your secret key gets passed to that machine's Keyserver, which then stores it. This is only a concern if you don't trust the remote machine. If you have doubts, however, don't log in to a remote machine if it requires a password. Instead, use NFS to mount resources shared by the remote machine. As an alternative, you can use `keylogout(1)` to delete the Keyserver.

- Using secure NFS can result in some degrading of performance. However, that is the impact on network performance. Not all file operations go over the network, so the impact on total system performance is actually lower. Secure NFS is an optional feature; environments that require higher performance at the expense of security can turn it off.

19 Using the Automounter

Introduction

Resources shared through NFS can be mounted using a method called "automounting." The automount program mounts and unmounts remote directories on an as-needed basis. Whenever a user on a client running the automounter invokes a command that needs to access a remote file, the shared resource to which that file belongs is mounted automatically. When a certain amount of time has elapsed without the resource being accessed, it is automatically unmounted.

Once the automounter is invoked, an administrator does not have to set up automatic mounting with the `vfstab` file, or use the `mount` and `umount` commands at the command line. All mounting is done automatically and transparently.

NOTE Mounting some resources with `automount` does not exclude the possibility of mounting others with `mount`; in fact, in a diskless machine you *must* mount root (/) and `/usr` with `mount`.

This chapter explains how the automounter works and provides instructions for setting it up.

How the Automounter Works

Unlike mount, automount does not consult the file /etc/vfstab for a list of resources to mount automatically. Rather, it consults a series of maps.

The automounter mounts everything under the directory /tmp_mnt, and provides a symbolic link from the requested mount point to the actual mount point under /tmp_mnt. For instance, if a user wants to mount a remote directory src under /usr/src, the actual mount point will be /tmp_mnt/usr/src, and /usr/src will be a symbolic link to that location.

When automount is called, it forks a daemon to serve each mount point in the maps and makes the kernel believe that the mount has taken place. The daemon sleeps until a request is made to access the corresponding resource. In the mount state, the daemon intercepts the request, mounts the remote resource, creates a symbolic link between the requested mount point and the actual mount point under /tmp_mnt, passes the symbolic link to the kernel, and steps aside.

When a predetermined amount of time has passed with the link not being touched (generally five minutes), the daemon unmounts the resource and resumes its previous position.

Preparing the Maps

A server never knows, nor cares, whether the files it shares are accessed through `mount` or `automount`. Nothing, therefore, needs to be done to a server to run the automounter.

A client, however, needs special files for the automounter. As mentioned earlier, `automount` does not consult `/etc/vfstab`; rather, it consults a map file (or files) specified at the command line.

There are three basic kinds of automounter maps:

- a master map
- a direct map
- an indirect map

The master map is a general map that invokes other maps that contain more specific information. When you enter a command to invoke the automounter and you specify the master map, the master map then calls direct and indirect maps to get the information it needs to give to the `automount` command.

A direct map contains all the information `automount` needs to do the mount. It can be called directly by the automounter, or through a master map.

An indirect map allows you to specify alternate servers for a set of resources. It also allows you to specify resources to be mounted as a hierarchy under the same mount point. The indirect map is called only through a master map, which specifies the mount point on which all resources listed in the indirect map should be mounted.

You should create all your automounter maps in your `/etc` directory, using any supported text editor.

Conventions

When creating any of the automounter maps, follow these conventions:

- Use a backslash before characters that may confuse the automounter's parser. For example, suppose you want to mount a directory whose name includes a colon, such as `rc0:dk1`. This resource name might result in the map entry

```
/junk   -ro   vmsserver:rc0:dk1
```

The automounter will be confused by the second colon in this entry. To avoid a problem, the entry should look like this:

```
/junk -ro vmsserver:rc0\:dk1
```

- Use double quotes around a resource name to hide white space.
- If you enter a comment into a map, make sure each line of the comment is preceded by #.
- If you include a long entry in a map, use backslashes to split the line into two or more shorter lines. For example, the entry

```
/usr/frame -ro,soft redwood:/usr/frame1.3 balsa:/export/frame
```

could be written as

```
/usr/frame -ro,soft  redwood:/usr/frame1.3 \
                     balsa:/export/frame
```

Writing a Master Map

Each line in a master map has the syntax:

mount-point map [*mount-options*]

where *mount-point* is the full pathname of a directory on which resources should be mounted; *map* is the name of the map that lists the resources to be mounted and their locations; and *mount-options* is a comma-separated list of options that regulates the mounting of the entries mentioned in the *map* (unless the *map* entries list other options).

The options that can be specified for *mount-options* are the same options that can be specified with the mount command, except for fg and bg. See the mount(1M) manual page for information.

The *mount-point* can be any mount point you have created for a remote mount. Here is a sample entry in a master map:

```
/usr/man  /etc/libmap  -ro
```

This entry tells the automounter to look in the indirect map /etc/libmap and to mount everything listed there on /usr/man on the local system. This entry also tells the automounter to mount resources on /usr/man read-only; however, if libmap indicates that a resource should be mounted read-write, it will be mounted read-write.

If the master map calls a direct map, *mount-point* should be /-. This tells the automounter to mount the entries in a direct map on the mount point specified in that map (the *location* field in a direct map contains a full pathname).

Below is a sample master map:

```
#Mount-point  Map              Mount-options
/usr/reports  /etc/reportmap   -rw,intr,secure
/usr/man      /etc/libmap      -ro
/-            /etc/direct.map  -ro,intr
```

Writing a Direct Map

All entries in a direct map have the syntax

key [*mount-options*] *location*

where *key* is the full pathname of the mount point; *mount-options* is a comma-separated list of options that regulates the mounting of the resource specified in the entry; and *location* is the location of the resource, specified as *server*:*pathname*.

The *mount-options* can be any of the options that can be specified with the mount command, except for the options fg and bg. For a list of valid options, see the mount(1M) manual page.

The following is a sample entry in a direct map:

```
/usr/fun -ro,soft peach:/usr/games
```

This entry means that the remote resource /usr/games on the server named peach should be soft mounted read-only on the local mount point /usr/fun. Whenever a user tries to access a file or directory that is part of the usr/games directory tree, the automounter reads the direct map, mounts the resource from server peach onto the mount point /tmp_mnt/usr/fun on the local system, then creates a symbolic link between /tmp_mnt/usr/fun and /usr/fun. The user is unaware that the mount operation is taking place, and the resource appears to the user to be at /usr/fun.

The following is a typical direct map:

```
/usr/local \
                  /bin     -ro,soft  ivy:/export/local/sun3 \
                  /share   -ro,soft  ivy:/export/local/share \
                  /src     -ro,soft  ivy:/export/local/src
/usr/man                   -ro,soft  oak:/usr/man \
                                     rose:/usr/man \
                                     willow:/usr/man
/usr/games                 -ro,soft  peach:/usr/games
/var/spool/news            -ro,soft  pine:/var/spool/news
/usr/frame                 -ro,soft  redwood:/usr/frame1.3 \
                                     balsa:/export/frame
```

Note that, in the first entry, more than one mount point is specified, and more than one location is specified. Specifying multiple mounts and locations is discussed later in this chapter.

Writing an Indirect Map

Entries in an indirect map have the syntax

key [*mount-options*] *location*

where *key* is the name (not the full pathname) of the directory that will be used as the mount point; *mount-options* is a comma-separated list of options that regulates the mount; and *location* is the location of the resource, specified as *server*:*pathname*.

Once the key is obtained by the automounter, it is suffixed to the mount point associated with it either on the command line or in the master map.

For example, suppose one of the entries in the master map reads:

```
/usr/reports /etc/reportmap -rw,intr,secure
```

Here /etc/reportmap is the name of the indirect map that lists the remote resources to be mounted under /usr/reports.

Here is a typical indirect map:

```
#key     mount-options   location
willow                   willow:/home/willow
cypress                  cypress:/home/cypress
poplar                   poplar:/home/poplar
pine                     pine:/export/pine
apple                    apple:/export/home
ivy                      ivy:/home/ivy
peach    -rw,nosuid      peach:/export/home
```

Assume this map resides on host `oak`. If user Laura has an entry in `oak`'s password database specifying her home directory as `/home/willow/laura`, then whenever she logs into machine `oak`, the automounter will mount (as `/tmp_mnt/home/willow`) the directory `/home/willow` from machine `willow`; if one of the directories is indeed `laura`, Laura will be in her home directory.

NOTE Any option in the indirect map overrides all options specified in the master map or on the command line.

Specifying Multiple Mounts

An entry in a direct or indirect map can describe multiple mounts, where the mounts can be from different locations and with different mount options. In the sample map, the first entry (which is actually one long entry whose readability has been improved by splitting it into three lines) mounts `/usr/local/bin`, `/usr/local/share` and `/usr/local/src` from the server `ivy`, with the options "read-only" and "soft." The entry could also read:

```
/usr/local \
                /bin     -ro,soft     ivy:/export/local/sun3 \
                /share   -rw,secure   willow:/usr/local/share \
                /src     -ro,intr     oak:/home/jones/src
```

where the options are different and more than one server is used.

Multiple mounts can be hierarchical. When resources are mounted hierarchically, each resource is mounted on a subdirectory within another resource. When the root of the hierarchy is referenced, the automounter mounts the entire hierarchy. The concept of *root* here is very important. In the case of a single mount, there is no need to specify the root of the mount point, because it is assumed that the location of the mount point is at the mount root or `/`. When mounting a hierarchy, however, the automounter must have a mount point for each mount within the hierarchy. The following illustration shows a true hierarchical mounting.

```
/usr/local \
                /        -rw,intr     peach:/export/local \
                /bin     -ro,soft     ivy:/export/local/sun3 \
                /share   -rw,secure   willow:/usr/local/share \
                /src     -ro,intr     oak:/home/jones/src
```

The mount points used here for the hierarchy are `/`, `/usr/bin`, `/share`, and `src`. Note that these mount point paths are relative to the *mount* root, not the host's *file system* root. The first entry in the example above has `/` as its mount point. It is mounted at the mount root. There is no requirement that the first mount of a hierarchy be at the mount root. The automounter will issue `mkdir` commands to build a path to the first mount point if it is not at the mount root.

NOTE A true hierarchical mount can be a problem if the server for the root of the hierarchy goes down; any attempt to unmount the lower branches will fail, since the unmounting has to proceed through the mount root, which also cannot be unmounted while its server is down.

Specifying Multiple Locations

A mount entry in a direct or indirect map can include more than one location in its *location* field. This means that the mounting can be done from any of the locations specified.

Specifying multiple locations makes sense only when you are mounting a resource read-only, since you generally want to have control over the locations of files you write or modify. An example of a read-only resource you might mount from a variety of locations is on-line documentation. In a large network, the current set of on-line manual pages may be available from more than one server. It doesn't matter which server you mount them from, as long as the server is up and running and sharing its files. If manual pages reside in a directory `/usr/man` on three different hosts, called `oak`, `rose`, and `willow`, you can mount from any location by specifying the following in the direct map:

```
/usr/man -ro,soft oak:/usr/man rose:/usr/man \
     willow:/usr/man
```

This could also be expressed as a comma-separated list of servers, followed by a colon and the pathname (as long as the pathname is the same on all servers):

```
/usr/man -ro,soft oak,rose,willow:/usr/man
```

From the list of servers, the automounter first selects those that are on the local network and queries or "pings" them. The first server to respond is selected, and an attempt is made to mount from it.

If the server goes down while the mount is in effect, the resource becomes unavailable. An option here is to wait five minutes until the auto-unmount takes place and try again; next time around, the automounter will choose one of the available servers. (Another option is to use the `umount` command, inform the automounter of the change in the mount table, and retry the mount; see "Updating the Mount Table" later in this chapter for more information.)

Specifying Subdirectories

Until now we have used the form *server*:*pathname* to indicate a location in a map entry; you can also specify a subdirectory in the *location* field, using the syntax *server*:*pathname*:*directory*.

Assume you have a master map on host oak that contains the following entry:

```
/home    /etc/auto.home    -rw, intr, secure
```

Here /etc/auto.home is the name of an indirect map that contains the entries to be mounted under /home.

Below is the auto.home map:

```
#key       mount-options   location
cypress                    cypress:/home/cypress
poplar                     poplar:/home/poplar
pine                       pine:/export/pine
apple                      apple:/export/home
ivy                        ivy:/home/ivy
peach      -rw,nosuid      peach:/export/home
john                       willow:/home/willow:john
mary                       willow:/home/willow:mary
joe                        willow:/home/willow:joe
```

Look at the first entry in the map, and assume user Adam has his home directory on host cypress. If Adam has an entry in host oak's password database specifying his home directory as /home/cypress/adam, then he can log in to oak and the automounter will mount (as /tmp_mnt/home/cypress) the directory /home/cypress residing on cypress. If one of the directories is adam, then Adam will be in his home directory. Because of the options specified in oak's master map, Adam's home directory is mounted read/write, interruptable, and secure.

Now look at the last three entries in the indirect map. Note that these entries refer to the same resource on the same host (/home/willow). They also specify subdirectories in the *location* field (john, mary, and joe).

Now when a user logs in to host oak and requests access to the home directory john on host willow, the automounter mounts willow:/home/willow. It then places a symbolic link between /tmp_mnt/home/willow/john and /home/john.

If user Mary then tries to access her home directory from host oak, the automounter sees that willow:/home/willow is already mounted, so all it has to do is return the link between /tmp_mnt/home/willow/mary and /home/mary.

In general, it's a good idea to provide a *subdirectory* entry in the *location* field when different map entries refer to the same resource shared by the same server.

Using Substitutions

If you have a map with a lot of subdirectories specified, as in the following indirect map

```
#key    mount-options   location
john                    willow:/home/willow:john
mary                    willow:/home/willow:mary
joe                     willow:/home/willow:joe
able                    pine:/export/home:able
baker                   peach:/export/home:baker
          [. . .]
```

you should consider using string substitutions. The ampersand character (&) can be used to substitute the key wherever it appears. Using the ampersand, the above map would look like this:

```
#key    mount-options    location
john                     willow:/home/willow:&
mary                     willow:/home/willow:&
joe                      willow:/home/willow:&
able                     pine:/export/home:&
baker                    peach:/export/home:&
        [. . .]
```

If the name of the server is the same as the key itself, for instance:

```
#key    mount-options    location
willow                   willow:/home/willow
peach                    peach:/home/peach
pine                     pine:/home/pine
oak                      oak:/home/oak
poplar                   poplar:/home/poplar
        [. . .]
```

the use of the ampersand would result in:

```
#key    mount-options    location
willow                   &:/home/&
peach                    &:/home/&
pine                     &:/home/&
oak                      &:/home/&
poplar                   &:/home/&
        [. . .]
```

If all entries in a map have the same format, you can use the catch-all substitute character, the asterisk (*), as in the following example:

```
#key    mount-options  location
oak                    &:/export/&
poplar                 &:/export/&
*                      &:/home/&
```

Once the automounter reads the catch-all key, it does not continue to read the map.

Using Environment Variables

You can use the value of an environmental variable by prefixing a dollar sign ($) to its name. Braces can also be used to delimit the name of the variable from appended letters or digits.

The environmental variables can be inherited from the environment or can be defined explicitly with the -D option on the command line. For example, if you want each client to mount client-specific files in the network in a replicated format, you could create a specific map for each client according to its name, so that the relevant line for host oak would be:

```
/mystuff   cypress,ivy,balsa:/export/hostfiles/oak
```

and for willow:

```
/mystuff   cypress,ivy,balsa:/export/hostfiles/willow
```

This scheme is viable within a small network, but maintaining this kind of host-specific map across a large network usually isn't feasible. The solution in this case would be to invoke the automounter with a command line similar to the following:

```
automount -D HOST='uname' ......
```

and have the entry in the direct map read:

```
/mystuff   cypress,ivy,balsa:/export/hostfiles/$HOST
```

Now each host would find its own files in the mystuff directory, and the task of centrally administering and distributing the maps becomes easier.

Invoking the Automounter

Once the maps are written, you should make sure that there are no equivalent entries in /etc/vfstab, and that all the entries in the maps refer to NFS shared resources.

The syntax to invoke the automounter is:

```
automount [-mnTv] [-Dname=vvalue] [-fmaster-file] [-Mmount-directory] \
     [-tsub-options] [directory map [-mount-options]] . . .
```

See automount(1M) for a complete description of options. The sub-options are the same as those for NFS mount, with the exception of bg (background) and fg (foreground), which do not apply.

The default mount point for all mounts is /tmp_mnt. Like the other names, this is an arbitrary name. It can be changed when you enter the automount command with the -M option. For example,

```
automount -M /auto ...
```

causes all mounts to happen under the directory /auto, which the automounter creates if it doesn't already exist.

The automounter can be invoked in one of the following ways:

1. You can specify all arguments to the automounter without reference to the master map:

   ```
   automount /net -hosts /home /etc/indirect_map \
        -rw,intr,secure /- /etc/direct_map -ro,intr
   ```

2. You can specify all arguments in the master map and instruct the automounter to look in it for instructions:

   ```
   automount -f /etc/master_map
   ```

3. You can specify more mount points and maps in addition to those mentioned in the master map:

   ```
   automount -f /etc/master_map /src /etc/auto.src -ro,soft
   ```

4. You can nullify one of the entries in the master map (particularly useful if you are using a map that you cannot modify but does not meet the needs of your machine):

```
automount -f /usr/lib/master_map /home -null
```

5. You can override an entry in the master map by specifying a different indirect map on the command line, as follows:

   ```
   automount -f /usr/lib/master_map /home \
        /myown/indirect_map -rw,intr
   ```

   This command tells the automounter to mount /home according to instructions in /myown/*indirect_map*, not according to instructions in the indirect map specified in the master map.

Updating the Mount Table

If you use `umount` to unmount explicitly one of the automounted resources, have the automounter re-read the `/etc/mnttab` file. This can be accomplished by sending signal 1 to the automount daemon, as shown below.

```
        kill -1 PID
```

where *PID* is the automount process ID.

Modifying the Maps

You can modify the automounter maps at any time; however, the automounter looks at the master and indirect maps only when it is invoked. To make changes take effect, stop NFS operation and start it again (by exiting and re-entering run level 3).

You do not have to exit and re-enter run level 3 to make changes to a direct map take effect. Changes to a direct map take effect the next time the automounter has to mount the modified entry.

Troubleshooting

The following are error messages you may see if the automounter fails.

- *mapname*`: Not found`

  The required map cannot be located. This message is produced only when the `-v` (verbose) option is given. Check the spelling and pathname of the map name.

- `dir` *mountpoint* `must start with '/'`

  The automounter mount point must be given as full pathname. Check the spelling and pathname of the mount point.

- *mountpoint*`: Not a directory`

  The *mountpoint* exists but it is not a directory. Check the spelling and pathname of the mount point.

- `hierarchical mountpoint:` *mountpoint*

  The automounter will not allow itself to be mounted within an automounted directory. You will have to think of another strategy.

- `WARNING:` *mountpoint* `not empty!`

  The mount point is not an empty directory. This message is produced only when the `-v` (verbose) option is given, and it is only a warning. All it means is that the previous contents of *mountpoint* will not be accessible.

- `Can't mount` *mountpoint*`:` *reason*

  The automounter cannot mount itself at *mountpoint*. The *reason* should be self-explanatory.

- *hostname*`:`*filesystem* `already mounted on` *mountpoint*

  The automounter is attempting to mount a resource on a mount point, but the resource is already mounted on that mount point. This happens if an entry in `/etc/vfstab` is duplicated in an automounter map (either by accident or because the output of `mount -p` was redirected to `vfstab`). Delete one of the redundant entries.

- `WARNING:` *hostname*`:`*filesystem* `already mounted on` *mountpoint*

  The automounter is mounting itself on top of an existing mount point.

- `couldn't create` *directory*`:` *reason*

  The system could not create a directory. The *reason* should be self-explanatory.

- `bad entry in map` *mapname* `"`*map entry*`"`
- `map` *mapname*`, key` *map key*`: bad`

  The map entry is malformed, and the automounter cannot interpret it. Recheck the entry; perhaps there are characters in it that need escaping.

- *hostname*`: exports:` *rpc_err*

  There is an error when the automounter tries to get a share list from *hostname.* This indicates a server or network problem.

- `host` *hostname* `not responding`
- *hostname*`:`*filesystem* `server not responding`
- `Mount of` *hostname*`:`*filesystem* `on` *mountpoint*`:` *reason*

  You will see these error messages after the automounter attempts to mount from *hostname* but gets no response or fails. This may indicate a server or network problem.

- *mountpoint* `-` *pathname* `from` *hostname*`: absolute symbolic link`

  When mounting a resource, the automounter has detected that *mountpoint* is an absolute symbolic link (beginning with `/`). The content of the link is *pathname.* This may have undesired consequences on the client; for example, the content of the link may be `/usr`.

- `Cannot create socket for broadcast rpc:` *rpc_err*
- `Many_cast select problem:` *rpc_err*
- `Cannot send broadcast packet:` *rpc_err*
- `Cannot receive reply to many_cast:` *rpc_err*

  All these error messages indicate problems attempting to "ping" servers for a replicated file system. This may indicate a network problem.

- `trymany: servers not responding:` *reason*

  No server in a replicated list is responding. This may indicate a network problem.

- `Remount` *hostname*`:`*filesystem* `on` *mountpoint*`: server not responding`

  An attempted remount after an unmount failed. This indicates a server problem.

- `NFS server (pid`*n*`@`*mountpoint*`) not responding still trying`

  An NFS request made to the automount daemon with PID *n* serving *mountpoint* has timed out. The automounter may be overloaded temporarily, or dead. Wait a few minutes; if the condition persists, the easiest solution is to reboot the client. If you don't want to reboot, exit all processes that make use of automounted resources (or change to a non-automounted resource, in the case of a shell), kill the current automount process, and restart it again from the command line. If this fails, you must reboot.

20. THE NETWORK LOCK MANAGER

20 The Network Lock Manager

An Overview of the Network Lock Manager

NFS takes advantage of a network locking facility, accomplished via a user-level daemon called the Network Lock Manager. The Lock Manager supports the UNIX System V style of advisory and mandatory file and record locking—provided via `lockf()` and `fcntl()`.

Locking prevents multiple processes from modifying the same file at the same time, and allows cooperating processes to synchronize access to shared files. The user interfaces with the network locking service by way of the standard `fcntl()` system-call interface, and rarely requires any detailed knowledge of how it works. The kernel maps user calls to the `fcntl()` system call or the `lockf()` library call into RPC-based messages to the local lock manager. The fact that the file system may be spread across multiple machines is really not a complication—until a crash occurs.

All computers crash from time to time, and in an NFS environment, where multiple machines can have access to the same file at the same time, the process of recovering from a crash is necessarily more complex than in a non-network environment. First of all, locking is inherently stateful. If a server crashes, clients with locked files must be able to recover their locks. If a client crashes, its servers must have the sense to release the locks held by processes running on the client. Second, to preserve NFS's overall transparency, the recovery of lost locks must not require the intervention of the applications themselves. This is accomplished as follows:

- Basic file access operations, such as read and write, use a stateless protocol (the NFS protocol). All interactions between NFS servers and clients are atomic—the server doesn't remember anything about its clients from one interaction to the next. In the case of a server crash, client applications will simply sleep until the server comes back up and their NFS operations can complete.
- Stateful services (those that require the server to maintain client information from one transaction to the next), such as the locking service, are not part of the NFS per se. They are separate services that use the status monitor to ensure that their implicit network state information remains consistent with the real state of the network. There are two specific state-related problems involved in providing the locking in a network context:

1. If the client has crashed, the lock can be held forever by the server.
2. If the server has crashed, it loses its state (including all its lock information) when it recovers.

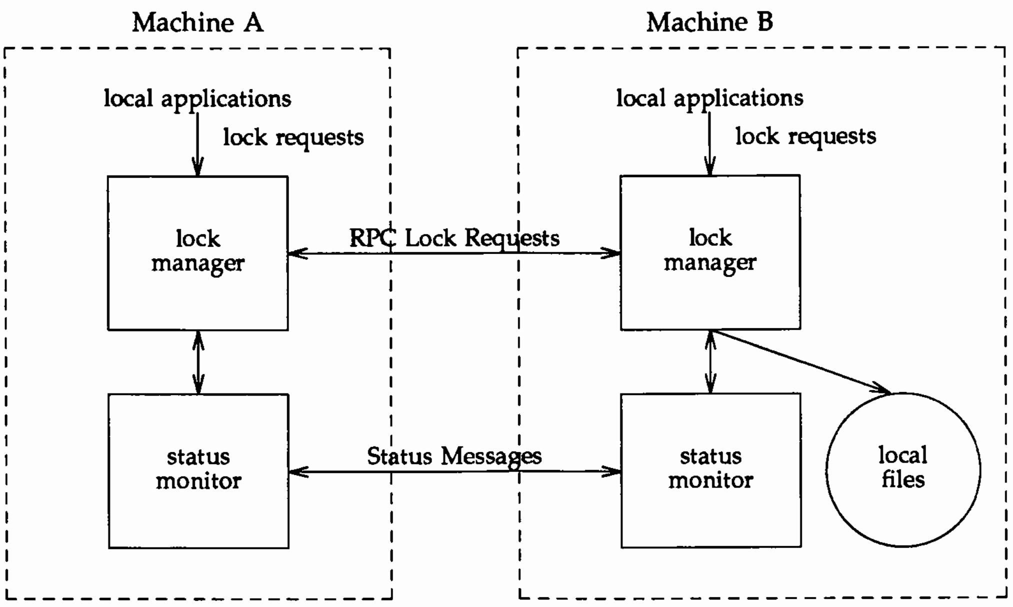
The Network Lock Manager solves both of these problems by cooperating with the Network Status Monitor to ensure that it is notified of relevant machine crashes. Its own protocol then allows it to recover the lock information it needs when crashed machines recover.

The lock manager [lockd(1M)] and the status monitor [statd(1M)] are both network-service daemons—they run at user level, but they are essential to the kernel's ability to provide fundamental network services, and they are therefore run on all network machines. Like other network-service daemons—which provide, for example, remote-login services (**rlogind**)—they are best seen as extensions to the kernel, which, for reasons of space, efficiency, and organization, are implemented as daemons. Application programs that need a network service can either call the appropriate daemon directly with RPC/XDR, or use a system call to call the kernel. In this later case, the kernel will use RPC to call the daemon. The network daemons communicate among themselves with RPC. (See "The Locking Protocol" later in this section for some details about the lock manager protocol.)

It should be noted that the daemon-based approach to network services allows for tailoring by users who need customized services. It is possible, for example, for users to alter the lock manager to provide locking in a different style.

The following figure depicts the overall architecture of the locking service.

Figure 20-1: Architecture of the Locking Service over NFS



At each server site, a lock manager process accepts lock requests, made on behalf of client processes by a remote lock manager, or on behalf of local processes by the kernel. The client and server lock managers communicate with RPC calls. Upon receiving a remote lock request for a machine that it doesn't already hold a lock on, the lock manager registers its interest in that machine with the local status monitor, and waits for that monitor to notify it that the machine is up. The monitor continues to watch the status of registered machines, and notifies the lock manager if one of them is rebooted (after a crash). If the lock request is for a local file, the lock manager tries to satisfy it, and communicates back to the application along with the appropriate RPC path.

The crash recovery procedure is very simple. If the failure of a client is detected, the server releases the failed client's locks, on the assumption that the client application will request locks again as needed. If the recovery (and, by implication, the crash) of a server is detected, the client lock manger retransmits all the lock requests previously granted by the recovered server. This retransmitted information is used by the server to reconstruct its locking state.

The locking service, then, is essentially stateless. Or to be more precise, its state information is carefully circumscribed within a pair of system daemons that are set up for automatic, application-transparent crash recovery. If a server crashes, and thus loses its state, it expects that its clients will be notified of the crash and that they will send it the information it needs to reconstruct its state. The key in this approach is the status monitor, which the lock manager uses to detect both client and server failures.

For more information about the lock manager daemon, see the `lockd`(1M) manual page.

The Locking Protocol

There are four basic kernel-to-Lock Manager requests:

KLM_LOCK
  Lock the specified record.

KLM_UNLOCK
  Unlock the specified record.

KLM_TEST
  Test if the specified record is locked.

KLM_CANCEL
  Cancel an outstanding lock request.

Despite the fact that the network lock manager adheres to the `lockf()` and `fcntl()` semantics, there are a few subtle points about its behavior that deserve mention. These arise directly from the nature of the network:

- The first and most important point to be made about the lock manager's behavior has to do with crashes. When an NFS client goes down, the lock managers on all of its servers are notified by their status monitors, and they simply release its locks, on the assumption that it will request them again when it wants them. When a server crashes, however, matters are different: Clients will wait for the server to come back up. When it does, the server's lock manager will give the client lock managers a grace period to submit lock reclaim requests; during this period, the server will accept only reclaim requests. The client status monitors will notify their respective lock managers when the server recovers. The default grace period is 45 seconds.

- It is possible that, after a server crash, a client will not be able to recover a lock that it had on a file on that server. This can happen for the simple reason that another process may have beaten the recovering application process to the lock. In this case the SIGLOST signal will be sent to the process (the default action for this signal is to kill the application).
- The local lock manager does not reply to the kernel lock request until the server lock manager has gotten back to it. Further, if the lock request is on a server new to the local lock manager, the lock manager registers its interest in that server with the local status monitor and waits for its reply. Thus, if either the status monitor or the server's lock manager are unavailable, the reply to a lock request for remote data is delayed until it becomes available.

For more information about UNIX System V locking, see the `fcntl`(2) and `lockf`(3C) manual pages.

The Network Status Monitor

The lock manager relies heavily on a service called the Network Status Monitor to maintain the inherently stateful locking service within the stateless NFS environment. However, the status monitor, implemented as `statd`(1M), is very general and can also be used to support other kinds of stateful network services and applications. Normally, crash recovery is one of the most difficult aspects of network application development, and requires a major design and installation effort. The status monitor makes it more or less routine.

The status monitor works by providing a general framework for collecting network status information. Implemented as a daemon that runs on all network machines, it provides a simple protocol that allows applications to monitor easily the status of other machines. Its use improves overall robustness, and avoids situations in which applications running on different machines (or even on the same machine) come to disagree about the status of a site—a potentially dangerous situation that can lead to inconsistencies in many applications.

Applications using the status monitor do so by registering with it the machines that they are interested in. The monitor then tracks the status of those machines, and when one of them crashes (actually, when one of them recovers from a crash), it notifies the interested applications to that effect. Then they take whatever actions are necessary to reestablish a consistent state.

This approach provides the following advantages:

- Only applications that use stateful services must pay the overhead—in time and in code—of dealing with the status monitor.
- The implementation of stateful network applications is eased, since the status monitor shields application developers from the complexity of the network.

For more information about the status monitor, see the `statd(1M)` manual page.

APPENDIX D: USING THE NFS sysadm INTERFACE

D Using the NFS sysadm Interface

Introduction

UNIX System V Release 4.0 provides you with an interface, called sysadm, that lets you set up and administer NFS through a series of menus. The sysadm interface not only lets you enter basic NFS configuration information, but it also acts as a tutorial by introducing and explaining key NFS concepts. Once you access a sysadm menu, help screens provide you with background information and explanations regarding menu selections. Access the help screens by using the "HELP" function key; use the "CANCEL" function key to exit the help mode. Continue making menu selections until you complete the particular task.

NOTE For instruction on moving through the interface, selecting menu options, and so on, see the tutorial in the *System Administrator's Guide*.

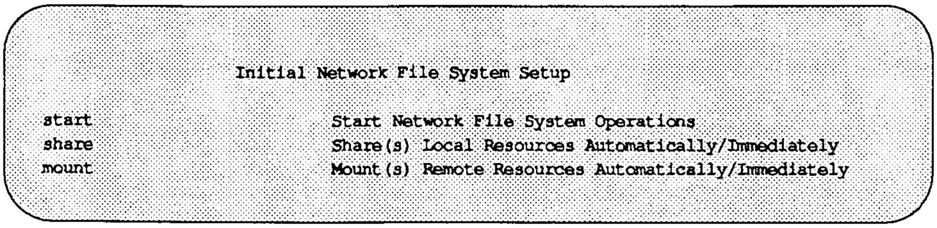
Here are the NFS procedures you can perform through the sysadm menu interface.

Procedure 1	Set Up Network File System To perform initial NFS setup.
Procedure 2	Start/Stop Network File System To start and stop NFS and check if it is currently running.
Procedure 3	Local Resource Sharing To manage the local resources you make available to other machines.
Procedure 4	Remote Resource Mounting To manage remote resources made available to your machine.

Procedure 1: Set Up Network File System

This procedure is used to set up NFS on your machine. When the procedure is done, you will have completed everything needed to run NFS on your system. This procedure assumes NFS software is installed on your system, as well as all the software utilities on which NFS depends. For information about installation, see the Release 4.0 *Release Notes*.

1. Log in as `root`.
2. Type `sysadm network_services` .
3. Select `remote_files`; then select `nfs_setup`, and then `nfs` to bring you to the following screen:



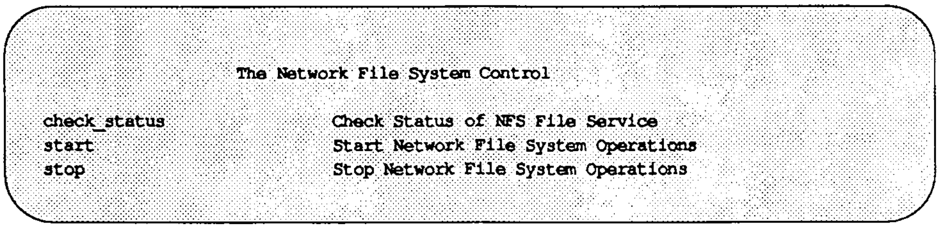
```
                 Initial Network File System Setup

start                         Start Network File System Operations
share                         Share(s) Local Resources Automatically/Immediately
mount                         Mount(s) Remote Resources Automatically/Immediately
```

You should execute each of the tasks in the order listed. Continue making interactive menu selections until the job is done. Remember, the "HELP" function key will provide you with help messages along the way.

Procedure 2: Start/Stop Network File System

This procedure is used to start and stop NFS. It's also used to determine if NFS is running.

1. Type `sysadm network_services` .
2. Select `remote_files`; then select `specific_ops`, then `nfs`, and then `control`. You are now at the following screen:



```
                         The Network File System Control

check_status             Check Status of NFS File Service
start                    Start Network File System Operations
stop                     Stop Network File System Operations
```

Select `start` to start NFS, or `stop` to stop NFS.

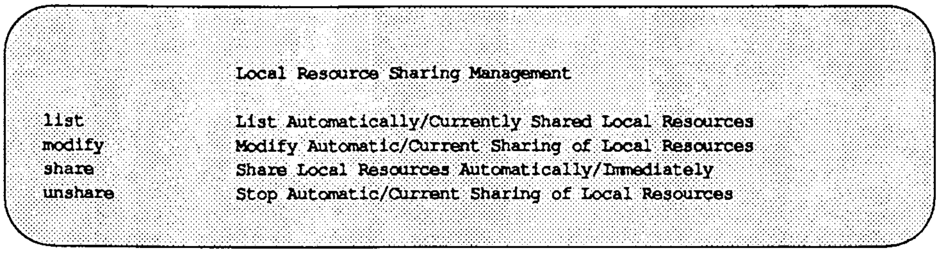
Select `check_status` to determine whether or not NFS file service is running.

Remember, the "HELP" function key will provide you with help messages along the way.

Procedure 3: Local Resource Sharing

This procedure allows you to make your local resources available or unavailable (share/unshare) to remote systems, via NFS. You can arrange this to happen automatically when NFS is started, or immediately during a work session. You can also modify the options by which your local resources are shared. Finally, this procedure enables you to list your local resources that are currently shared via NFS.

1. Type `sysadm network_services` .
2. Select `remote_files`; then select `local_resources`. You are now at the following screen:



```
                    Local Resource Sharing Management

list                List Automatically/Currently Shared Local Resources
modify              Modify Automatic/Current Sharing of Local Resources
share               Share Local Resources Automatically/Immediately
unshare             Stop Automatic/Current Sharing of Local Resources
```

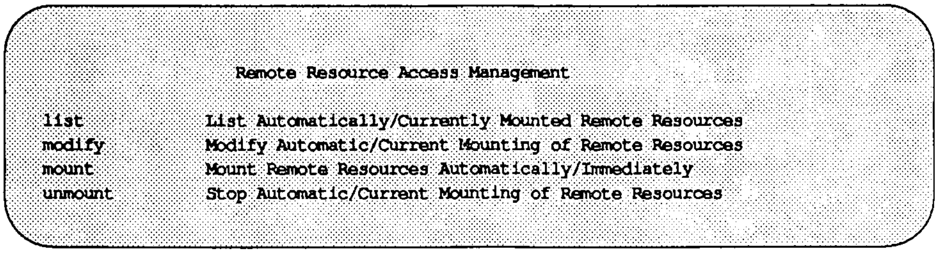
Select `list`, then `nfs` to list the local resources currently shared by NFS. Select `modify`, then `nfs` to modify sharing permissions of local resources via NFS. Select `share`, then `nfs` to share local resources via NFS. Select `unshare`, then `nfs` to unshare local resources currently shared via NFS.

Remember, the "HELP" function key will provide you with help messages along the way. Also see the chapters "Sharing and Mounting Resources Explicitly" and "Obtaining Information" for more information about these tasks.

Procedure 4: Remote Resource Mounting

This procedure allows you to make remote resources available or unavailable (mount/unmount) to your local computer, via NFS. With this procedure, you can specify resources to be mounted or unmounted automatically, whenever NFS operation stops and starts, or you can mount and unmount a resource immediately during a work session. You can also modify the options by which remote resources are mounted on your local computer. Finally, this procedure enables you to list the resources of remote systems that are currently available, via NFS, to your machine.

1. Type `sysadm network_services` .
2. Select `remote_files`, then select `remote_resources`. You are now at the following screen:



```
                    Remote Resource Access Management

list           List Automatically/Currently Mounted Remote Resources
modify         Modify Automatic/Current Mounting of Remote Resources
mount          Mount Remote Resources Automatically/Immediately
unmount        Stop Automatic/Current Mounting of Remote Resources
```

Select `list`, then `nfs` to list remote resources mounted via NFS. Select `modify`, then `nfs` to modify mount permissions of remote resources. Select `mount`, then `nfs` to mount remote resources. Select `unmount`, then `nfs` to terminate mounting of remote resources currently shared via NFS.

Remember, the "HELP" function key will provide you with help messages along the way. Also see the chapters "Sharing and Mounting Resources Explicitly" and "Obtaining Information" for details about these tasks.

Glossary

binding	The process by which a client locates the server that shares the information desired, and then sets up communication with that server.
caller	A process that uses RPC to have another process execute a procedure call.
client	A machine that mounts resources that have been shared by a server.
credentials	Information that is used to prove that something is as it claims to be—for example, an identification badge. (See also *verifiers*).
daemon	A program that runs autonomously, performing actions that facilitate more complex operations; for example, the mail service is run by several daemons, all of which work more or less without human oversight.
Data Encryption Standard (DES)	A standard cryptography algorithm used to ensure data security.
file handle	A key that a client gets from a server to facilitate all further requests between that client and server.
group identification number (GID)	A number that refers to a specific group of users on a system. Each user can belong to one or more groups.
hierarchy	A part or all of a file structure. The term *hierarchy* refers to both the directories of the file structure, and the files within the structure.
host	An individual machine.
hung	When a process has been stopped abnormally, with no way to restart that process.
map	A file that contains a listing of mount points and their corresponding resources. The maps are used by the automounter program to locate where a file structure should be mounted when it is needed.

mount	The action a client performs to access files in a server's shared directories. When a client mounts a resource, it does not copy that resource, but rather transparently accesses the resource as if it were local.
mount point	The location within the directory tree through which a machine accesses a mounted resource. The mount point for a resource is usually an empty directory.
Network File System (NFS)	A service that enables machines to share file resources across a network.
process identification number (PID)	A number that identifies a process to the operating system. Every process has a PID by which it is referenced.
public key cryptography	A cipher system that involves a published public key and an encoded secret key.
Remote Procedure Call (RPC)	Procedures that provide the means by which one process (the *caller*) can have another process (the *server*) execute a procedure call as if the caller had done so itself locally.
resource	A file system, a portion of a file system, a directory, or a single file, shared by a server across a network.
run level	A user mode, also called a *run state.*
server	A machine that shares file systems or portions of file systems, allowing remote machines to mount those resources.
server process	A process that receives directives from a caller process to execute procedures locally.
share	The action a server machine performs to allow some or all of its file system to become available to other hosts.

user identification number (UID)	A number that identifies a user to the operating system. Every user has a unique number that identifies that user to the operating system.
verifiers	Information that is used to prove that credentials are valid.

Index

REMOTE SERVICES USER'S GUIDE

REMOTE SERVICES USER'S GUIDE

Contents

21. INTRODUCTION TO RS

21 Introduction

About This Guide

UNIX System V Release 4 includes various user programs that enable you to perform operations on remote hosts using the TCP/IP networking software. This guide covers many of the most frequently used.

This guide is not intended to be a replacement for the remote services manual pages. It does not cover all user level network commands, nor does it discuss all applications of the commands that are covered.

Audience

This guide is directed to users who are unfamiliar with using the remote commands available with TCP/IP.

Organization

The organization of this guide is as follows:

- "Introduction," which gives you a bird's eye view of the contents of this guide, presents a brief explanation of what a remote command is, and introduces some basic terminology.
- "Copying Files Remotely," which explains how to copy files from one machine to another with the `rcp` command.
- "Executing Commands Remotely," which explains how to execute commands in a machine different from the one you are currently logged in to, using the `rsh` command.
- "Logging In to a Remote Machine," which explains how to login to another UNIX machine, using `rlogin`, and how to login to another machine that does not run the UNIX operating system, using `telnet`.
- "Transferring Files Between Machines," which explains how to transfer files both interactively and non-interactively between different machine, using the `ftp` and the `tftp` commands.
- "Obtaining Information," which explains how to obtain information about other users using the `finger` command, and how to determine if other machines are alive or not using the `ping` command.

Before You Begin

Remote commands are used with the TCP/IP networking software, which is part of the System V Release 4.0. TCP/IP must be running for any of the commands to work.

If you have any trouble with these commands, or if TCP/IP is not running and you wish to use these commands, talk to your system administrator.

An Introduction to Remote Commands

A remote command is a command which runs on a different machine than the one you are using. Remote commands allow you to access machines of different architectures, or even machines which are not running the same operating system.

User commands such as

`rlogin`	(remote login)
`rsh`	(remote shell)
`rcp`	(remote copy)
`ftp`	(file transfer program)

let you log in, create and use a shell, get information, and copy files on a remote machine. The `finger` command is useful to display information on any user you specify. You can use the `telnet` command to log in to other machines, whether or not they run the UNIX operating system.

Terminology

In reading the description of some of the commands, you may encounter terms which are unfamiliar to you. Most of the new terms used in this guide are explained here. If you find a term with which you are unfamiliar and which is not listed here, ask your system administrator.

abort	Discontinuing a process in the middle without waiting for the normal exit. Aborting is normally achieved by sending an interrupt signal to the program you are running.
daemon	A program that handles jobs automatically. Many remote commands get information from a remote machine by exchanging data with *daemons* running on the remote machine. An example of a daemon is the mail service, which processes mail automatically and routes it to the intended recipient.
local	The machine you are currently logged in to. It is contrasted with the term *remote* machine (see below).
remote	The machine to which you are connected across the network. You interact with the remote machine by using the commands shown in this guide.
shell	The program that you use to interface with your machine. A local shell is the shell you are running on the local machine, and a remote shell is the shell that runs on the machine to which you are connected. The Bourne Shell normally generates a dollar sign ($) prompt to show that it is ready to accept a command.
suspend	To halt temporarily whatever program you are running. You can return to a suspended program at any time and resume exactly where you left off.

Conventions

In this guide, prompts in screens are shown in the form *hostname*$, to make clear whether the shell is a local or a remote shell. You can set your terminal to include the host name in the prompt by adding the following line to your `$HOME/.profile` file:

```
        PS1="`hostname`$PS1"
```

22. COPYING FILES BETWEEN MACHINES

22. COPYING FILES BETWEEN MACHINES

22 Copying Files Between Machines

Introduction

The `rcp` program lets you copy files from your machine to another one, and vice versa. The basic syntax of `rcp` is:

```
rcp source destination
```

where the *source* is where the file is coming from, and *destination* is where it is going. The specific form that the two arguments take for different types of transfers are shown in the following sections.

Copying from Another Machine to Your Machine Using rcp

To copy a file from another machine to your machine with `rcp`, use the following syntax:

```
    rcp machinename:file directory
```

where *machinename* is the name of the machine you want to copy from; *file* is the file you want to copy; and *directory* is where you want to put the file on your system.

For example, to copy a file called `/home/charon/new.toy` from the machine called `pluto` to the directory called `/home/medici/toys` on your machine, venus, type

```
    rcp pluto:/home/charon/new.toy /home/medici/toys
```

You can use normal shorthand for directories (such as $HOME for your home directory when using the Bourne shell, `.` for the current directory and `..` for the parent directory).

When you want to call the file by a different name on your own machine, specify a destination *filename* at the end of the destination directory on your machine. For example, you could copy the file `new.toy` from machine `pluto` to your home directory and rename it `my.toy` by typing

```
    rcp pluto:/home/charon/new.toy $HOME/my.toy
```

Copying from Your Machine to Another Machine Using rcp

To `rcp` a file from your machine onto another machine, reverse the syntax described in the preceding section:

```
        rcp file machinename:directory
```

where *file* is the file on your machine you want to copy; *machinename* is name of the machine you want to copy to; and *directory* is the place you want to send the file to.

For example, to copy a file called `/home/medici/old.toy` from your machine to the directory called `/home/charon/trash` on the machine `pluto` type

```
        rcp /home/medici/old.toy pluto:/home/charon/trash
```

When you want to call the file by a different name on the other machine, specify a destination file at the end of the destination directory on that machine. For example, typing

```
        rcp /home/medici/old.toy pluto:/home/charon/trash/toy
```

will copy the file `old.toy` from `medici`'s home directory to the file named `toy` in the directory `/home/charon/trash` in the machine `pluto`.

Copying Directories with rcp

To copy a directory and its contents from another machine to your machine, or vice versa, use `rcp` with the `-r` option. Then, follow the steps for copying files; replace the filenames with the appropriate directory names.

NOTE Copying directories with `rcp` doesn't preserve ownership settings, nor does it necessarily preserve permissions.

To copy a directory and its contents from another machine to your machine, the syntax is

`rcp -r` *machinename*`:`*directory local_directory*

where *machinename* is the name of the remote machine; *directory* is the directory on that machine that you want to copy; and *local_directory* is the directory on your machine you want to copy to.

To copy a directory and its contents from your machine to another machine, the syntax is

`rcp -r` *local_directory machinename*`:`*directory*

where *local_directory* is the directory on your machine you want to copy, and *directory* is the place on the other machine you want to copy to.

Error Messages

An error message you commonly get when trying to do a remote copy is

```
....Permission denied
```

This error message may indicate that

- you do not have read permission on the file you want to copy.
- you do not have write permission on the directory you want to copy to.
- you do not have permission to access files in the remote machine because your machine's name is not in the remote machine's list of trusted hosts.

If you receive the message

```
Login incorrect
```

you do not have permission to access files in the remote machine because your name is not in that machine's password database.

In all cases when you receive an error message, consult with your system administrator.

23. EXECUTING COMMANDS REMOTELY

23 Executing Commands Remotely

Introduction

The `rsh` command allows you to execute a single command on another machine without having to log in formally (`rsh` stands for remote shell, or an interpreter capable of executing commands on another machine). `rsh` can save time when you know you only want to do one thing on the remote machine.

To execute a command on another machine, type `rsh` followed by the machine's name and the command. For example, if you want to see the contents of the directory `/home/fresno/crops` on the machine `fresno`, type

```
        rsh fresno ls -C /home/fresno/crops
```

When you execute a command on another machine using `rsh`, `rsh` doesn't log in; it talks to a daemon that spawns a shell for you and executes the command on the other machine. The type of shell spawned depends on the configuration of the entry for you in the remote machine's password database. If the shell spawned is the Bourne shell, your `.profile` file in the remote machine will be read; if it is the C shell, your `.cshrc` file will be read, if present, and `rsh` will use any pertinent aliases that you have defined on the other machine when executing the command.

Like `rlogin` and `rcp`, `rsh` uses the other machine's password database and the files `/etc/hosts.equiv` and `.rhosts` to determine whether you have unchallenged access privileges.

rsh and the Expansion of Shell Metacharacters

Like in the case of `rcp`, any shell metacharacters that are not escaped or quoted result in their expansion at the local level, not at the level of the remote machine.

This applies also to the redirection characters, >, <, and |. For instance, if you were to enter on machine oak the command

```
rsh willow ls /etc > /tmp/list
```

the output of the `ls` command on machine `willow` would be redirected to a file `/tmp/list` on machine oak; but if you enter the command

```
rsh willow ls /etc '>' /tmp/list
```

the output would now be redirected to a file `/tmp/list` on machine `willow`, because the redirection would not happen now at the local level.

Calling rsh with No Commands

If you call `rsh` using the syntax

```
rsh machinename
```

that is, with no arguments after the name of the remote machine, `rsh` will behave exactly as if you had entered

```
rlogin machinename
```

and you will be logged in at the remote machine (assuming you have permission).

Calling rsh by a Different Name

The command rsh can be called under a different name, by making a symbolic link between the file /usr/bin/rsh and a file called by the name of the remote host.

For example, to create a symbolic link between rsh and a remote host called willow, you would enter the command

```
        ln -s /usr/bin/rsh /usr/hosts/willow
```

Now, provided the directory /usr/hosts is in your search path, you can enter the command

```
        willow
```

on your machine to log in to machine willow.

If you want to obtain a listing of the directory /etc on machine willow, you can enter

```
        willow ls /etc
```

You can repeat the linking process for all the machines that you frequently access remotely. Note that making the symbolic links in the directory /usr/hosts is a convention; you can make them in any directory, as long as you have permission to create files in the directory and it is in your search path.

24. LOGGING IN TO REMOTE MACHINES

24 Logging In to Remote Machines

Logging In to Another UNIX Machine with rlogin

The `rlogin` command logs you in to other UNIX machines on a network.

Logging In with rlogin

To log in to a UNIX machine on a network, type `rlogin` and the *machine name* of the other machine. If your machine's name is in the other machine's `/etc/hosts.equiv` file, or in the `.rhosts` file in your remote home directory, then the other machine trusts your machine name and won't require you to type your password. Otherwise, a password prompt will appear. Type your password for that machine followed by **Return**. If you have an entry in the password database of the other machine, and you entered the correct password, you will be logged in the other machine as if you had just physically logged in to it.

```
venus$ rlogin jupiter
Password: (Here you type your password.)
Last login: Mon Oct 20 00:30:52 from venus
jupiter$ pwd
/home/medici
jupiter$ exit
Connection closed.
venus$
```

Aborting an rlogin Connection

To abort an `rlogin` connection, type a tilde character followed by a period (`~.`) at the beginning of a line. The login connection to the other machine aborts, and you find yourself back at your original machine.

NOTE

Usually you abort an `rlogin` connection only when you can't terminate the connection using `exit` or `logout` at the end of the work session.

When you log in to a series of machines, accessing each machine through another machine, and you use `~.` to abort the connection to any of the machines in the series, you return to the machine where you started; all the intermediate connections are severed.

```
venus$ rlogin comet
Last login: Thu Nov 21 05:04:03 from venus
comet% ~. (Sometimes ~ doesn't echo.)
Closed connection.
venus$
```

To disconnect to an intermediate `rlogin`, use two tildes followed by period (`~~.`), as shown in the example below:

```
venus% rlogin comet
comet% rlogin jupiter
jupiter% ~~. (Sometimes ~~ doesn't echo.)
comet%
```

Who Can rlogin?

A remote user who has no entry in the password database of a given machine is automatically denied permission to log in to it.

If a user is in the password database, the user will be asked for a password. If the password matches the one stored in the password database, the user will be granted permission to log in to the machine.

If the machine is listed in `/etc/hosts.equiv` on the remote host, or if the user has an `.rhosts` file located in his or her remote home directory with the machine name listed in it, the user can log in without first supplying a password.

If you cannot `rlogin` to a remote machine because you are asked for a password which the machine identifies as incorrect, see your system administrator.

rlogin to a Machine As Someone Else

From time to time you may want to log in as someone else, so that you can fully manipulate files on the remote machine. One example of this would be when you are working on someone else's machine (and using their username) and you want to log in to your own machine as yourself. The `-l` option to `rlogin` allows you to do this. The format is as follows:

```
rlogin machinename -l username
```

However, a number of restrictions apply to the `-l` option; for information, see the `rlogin`(1) manual page.

Suspending an rlogin Connection

If you are using a job control shell (`jsh` or `ksh`), you can suspend an `rlogin` connection, then return to it later. To do so, type the tilde character (~) followed by Ctrl-Z. The `rlogin` connection becomes a stopped process, and you are put back into the machine you logged in from. To reactivate the connection, type `fg`, or `%` followed by the job number of the stopped process (the default job number for `%` is the job you most recently stopped or put in the background).

```
venus$ rlogin animation
Last login: Thu Nov 21 07:07:07 from venus
animation$ ~    (Sometimes ^Z doesn't echo on the screen.)

Stopped
venus$ pwd
/home/medici
venus$ fg
rlogin animation    (Type Return here to get the command prompt.)

animation$ logout
Connection closed.
venus%
```

As is the case with aborting `rlogin` with `~~.`, using *two* tildes and a Ctrl-Z will suspend you to an intermediate `rlogin`. For example, if from `oak` you `rlogin` to `willow` and from there to `cypress`, entering `~.` will bring you back to `oak`, but entering `~~.` will bring you back to `willow`.

Logging In to a Machine Running Another Operating System with telnet

Because you can log in from one UNIX system machine to another using `rlogin`, you need to use `telnet` only when you want to log in to a machine running a different operating system.

Logging In with telnet

Imagine that you want to log in to machine `tops20`, running the TOPS20™ operating system. To log in to `tops20`, type `telnet`, followed by its machine name. `telnet` notifies you of the connection with the other machine, then identifies your escape character. Now you log in to the machine as you ordinarily would.

```
venus$ telnet tops20
Trying...
Connected to tops20.
Escape character is '^]'.


Yoyodyne Corp., TOPS-20 Monitor 6.1 (6762)-4
@LOG MEDICI

...
```

NOTE If you attempt to log in to a machine that isn't a part of your network, `telnet` displays a notification and a prompt. Exit from `telnet` by typing `quit`, or the abbreviation `q`.

Suspending a telnet Connection

If you are using a job control shell (`jsh` or `ksh`), you can suspend a `telnet` connection, then return to it later. To do so, type the standard escape character (usually Ctrl-]), followed by `z` at the `telnet>` prompt. The `telnet` program becomes a background process. To reactivate the connection, type `fg`, or `%` followed by the job number of the background process (the default job number for `%` is the job you most recently put in the background).

```
venus% telnet tops20
Trying...
Connected to tops20.
Escape character is '^]'.


Yoyodyne Corp., TOPS-20 Monitor 6.1 (6762)-4
@LOG MEDICI

...
@    (Type Ctrl-] to get telnet> prompt.)
telnet> z

Stopped
venus% fg
telnet tops20    (Type Return twice to get command prompt of other system.)

@logout
Connection closed by foreign host.
venus%
```

Aborting a telnet Connection

Just as with `rlogin`, you should abort a `telnet` connection only when you can't terminate the connection using `exit` or `logout` at the end of the work session.

If you have to abort a `telnet` connection, type the `telnet` *escape character* (usually Ctrl-], followed by `quit` at the `telnet>` prompt. The login connection to the other machine aborts, and you find yourself back at your original machine.

NOTE When you log in to a series of machines, accessing each machine through another machine, and you abort the connection to any of the machines in the series, you return to the machine where you originally started.

```
venus$ telnet tops20
Trying...
Connected to tops20.
Escape character is '^]'.


Yoyodyne Corp., TOPS-20 Monitor 6.1 (6762)-4
@LOG MEDICI

...
@    (Type Ctrl-] to get telnet> prompt.)
telnet> quit
venus$
```

25. TRANSFERRING FILES BETWEEN MACHINES

25. TRANSFERRING FILES BETWEEN MACHINES

25 Transferring Files Between Machines

Transferring Files with ftp

The ftp program is used to copy files to and from machines on a network. ftp is somewhat different from rcp in that it is not necessary to be a user on the remote machine, nor does the remote machine need to be running the same operating system. This command is also useful when you are trying to transfer files with unknown filenames, as ftp allows you to list directory contents on remote machines.

When you start ftp, you are placed in an interactive session with the daemon on the remote machine. The daemon is the part of the ftp program on the remote machine that handles all that needs to be done on that end. Once you are connected, the daemon reports that the connection is established, and then asks for a login. If you have an entry in the password database on the remote machine, you can just press Return to take the default answer, which is your own username. When you enter the correct password, you will then be given access to files on that machine. A sample of such a login appears as,

```
venus$ ftp dinger
Connected to dinger.
220 dinger FTP server ready.
Name (dinger:stein): Return

331 Password required for stein.
Password: (Here you type your password.)
230 User stein logged in.
ftp>
```

As you can see, the password does not echo on the screen when it is entered. Now the situation may arise where you want to transfer files between a machine on which you do not have an entry in the password database. Files on such a machine can still be accessed if the machine is set up for "anonymous" ftp. To set up a machine for anonymous ftp, create logins for ftp and anonymous in your /etc/passwd file. Then make sure you have the following directories and files on your system:

File or Directory	Permissions	Owner
`/home/ftp`	`dr-x--x--x`	`ftp`
`/home/ftp/bin`	`d--x--x--x`	`root`
`/home/ftp/bin/ls`	`---x--x--x`	`root`
`/home/ftp/dev`	`d--x--x--x`	`root`
`/home/ftp/dev/tcp`	`crw-rw-rw-`	`root`
`/home/ftp/etc`	`d--x--x--x`	`root`
`/home/ftp/etc/group`	`-r--r--r--`	`root`
`/home/ftp/etc/netconfig`	`-r--r--r--`	`root`
`/home/ftp/etc/passwd`	`-r--r--r--`	`root`
`/home/ftp/pub`	`drwxrwxrwx`	`ftp`

A session using anonymous ftp might look like this:

```
venus$ ftp berg
Connected to berg.
220 berg FTP server ready.
Name (berg:stein): anonymous
331 Guest login ok, send ident as password.
Password:   (Here you type some identification string.)
User anonymous logged in.
ftp>
```

Notice the request for `send ident as password`. This is saying that there is no specific password, but that you should send some sort of identification as a password (your name, for example). After connection has been established, you are then given the `ftp` prompt that tells you that `ftp` is ready to accept transfer commands.

Getting a Listing of Files on the Remote Machine

Once you are connected to a remote `ftp` daemon, you can get a listing of the files on the remote machine by using the command `ls`. All of the files accessible to you in that directory will then be listed. It is possible to move from one directory to another on the remote machine by means of the `cd` command, but it should be noted that unless you have access to those files, you will not be able to transfer them.

Copying Files Using get and put

The two commands that are used most with `ftp` are `get` and `put`. These commands get a copy of a file from the remote machine, or put a copy onto the remote machine, respectively. To use either command, enter the command followed by *filename*, which is the file to be copied. The `ftp` program will report that the transfer has begun, and then when the transfer is complete, along with diagnostic data on how long the transfer took.

```
ftp> get lab1.results
200 PORT command successful.
150 ASCII data connection for lab1.results (129.144.60.88,1163).
226 ASCII Transfer complete.
local: lab1.results remote: lab1.results
1162 bytes received in 0.08 seconds (14 Kbytes/s)
ftp>
ftp> put lab5.data
200 PORT command successful.
150 ASCII data connection for lab5.data (129.144.60.88,1165).
226 Transfer complete.
local: lab5.data remote: lab5.data
1162 bytes sent in 0.04 seconds (28 Kbytes/s)
ftp>
```

Copying Multiple Files Using mget and mput

You can "get" and "put" more than one file at a time. This is done by using the commands `mget` and `mput`, along with using metacharacters (for example, `*` and `?`). The metacharacter `*` will match anything, while `?` will match any one character. As with `get` and `put`, `ftp` will report when transfer begins. Before each file is transferred, you are asked whether or not you wish to transfer it. At this point you answer `y` and the file is transferred, or `n` and the file is skipped. After all matching files have been transferred, you are given the `ftp` prompt again.

```
ftp> mput lab*
mput lab1.results? y
200 PORT command successful.
150 ASCII data connection for lab1.results (129.144.60.88,1180).
226 Transfer complete.
local: lab1.results remote: lab1.results
31 bytes sent in 0.02 seconds (1.5 Kbytes/s)
mput lab2.data? n
mput lab3.results? y
200 PORT command successful.
150 ASCII data connection for lab3.results (129.144.60.88,1181).
226 Transfer complete.
local: lab3.results remote: lab3.results
75 bytes sent in 1e-06 seconds (7.3e+04 Kbytes/s)
ftp>
ftp> mget report?.final
mget report1.final? y
200 PORT command successful.
150 ASCII data connection for report1.final (129.144.60.88,1195).
226 ASCII Transfer complete.
local: report1.final remote: report1.final
2605 bytes received in .44 seconds (5.8 Kbytes/s)
mget report2.final? n
ftp>
```

Quitting an ftp Session

When you are finished with `ftp`, you can quit by entering the command `quit` at the prompt. The connection to the remote daemon will be dropped, and you will be returned to your local shell.

Aborting ftp While Transferring a File

If you are transferring files to or from a remote machine and it goes down, you need to abort the transfer. To abort `ftp` when transferring a file, press the interrupt key—usually Break. You are notified that the transfer was aborted, and then given an `ftp` prompt again.

What Happens If There Is No Daemon Present?

Sometimes there is no `ftp` daemon running on the remote machine. This can happen if the daemon dies for any reason, or the machine never started one in the first place. If there is no daemon, you can still use `ftp`, but the session is non-interactive. When this situation arises, `ftp` behaves like `tftp` (see the following section).

Transferring Files Non-Interactively Using tftp

The `tftp` program is very much like `ftp`, except that it is not an interactive process. This means that `tftp` does not require that you connect to the remote machine. Rather, when you begin a `tftp` session, you are able to issue commands that directly copy files to and from the remote machine, as long as you have an entry in the password database on the remote machine. However, since connection is not maintained between file transfers, you cannot get any directory information from the remote machine.

To use `tftp`, enter the command

```
tftp
```

When you see the prompt `tftp>`, you are ready to begin transferring files.

Getting Files Using get

In order to copy a file from the remote machine, you must use the command `get`. The syntax is

> `get` *machinename*`:`*file file2 ... fileN*

where *machinename* is the machine you wish to get files from, and *file* is the name of the file you wish to get. More than one file can be placed on the command line, with spaces separating the file names.

Putting Files Using put

In order to copy a file to the remote machine, you must use the command `put`. The syntax for `put` is

> `put` *machinename*`:`*file file2 ... fileN* `[`*remote_directory*`]`

where *machinename* is the machine you wish to transfer files to, *file* is the filename or a space-separated list of filenames that you wish to transfer, and *remote_directory* is an optional argument showing the specific directory on the remote machine into which you wish the files to be placed.

Quitting a tftp Session

When you are finished with `ftp`, you can quit by entering the command `quit` at the prompt.

26. OBTAINING INFORMATION

26 Obtaining Information

Displaying User Information with finger

The `finger` command displays information about any user you specify. `finger` does not give you information about other machines. It tells you only about other users. In fact, `finger` is so user-oriented that it accepts people's real names, as well as their usernames, as arguments.

This is what `finger` tells you:

- the user's login name
- his or her real name
- his or her home directory and login shell
- the last time he or she logged in to the machine from which you are issuing the command
- the last time he or she received mail, and the last time he or she read it
- the name of his or her terminal(s), and how long it's been idle.

Here is a slightly simplified example of a two typical `finger` requests. Your output may vary somewhat.

```
venus$ finger moby@sea
[sea]
Login name: moby                             In real life: Ishmael Wong
Directory: /home/shipwreck/moby       Shell: /usr/bin/sh
On since Nov 14 06:33:41 on console   4 days 14 hours Idle Time
New mail received Wed Nov 18 20:34:02 1987;
  unread since Wed Nov 18 16:20:24 1987
venus$ finger Henry Stamper
Login name: hank                             In real life: Henry Stamper, Jr
Directory: /home/oregon/hank          Shell: /usr/bin/sh
Last login Wed Oct 21 16:16 on ttyp0 from cairo
No unread mail
```

The `finger` command is useful to make sure that the user you are looking for is still active.

Determining If a Machine Is Alive on the Network Using ping

From time to time you will find that a remote machine is not answering your requests. This may indicate network-wide problems or simply that the host is down or disconnected from the network. The `ping` command offers the simplest way to find out if a host on your network is down. Its basic syntax is:

`/usr/sbin/ping` *host* [*timeout*]

where *host* is the name of the machine in question. The optional *timeout* argument indicates the time in seconds for `ping` to keep trying to reach the machine—20 seconds by default. The `ping(1)` manual page describes additional details.

If you type, for instance,

```
ping elvis
```

you will receive the following message, if host `elvis` is up:

```
elvis is alive
```

indicating that `elvis` has responded to your `ping`. If, however, host `elvis` is down or disconnected from the network, you will receive the following message after the specified (or default) timeout has elapsed:

```
no answer from elvis
```

Index

REFERENCE MANUAL

REFERENCE MANUAL

Table of Contents

1. Commands

3. Functions

4. File Formats

7. Special Files

Permuted Index

fusage disk	access profiler	fusage(1M)
ethers Ethernet	address mapping operations	ethers(3N)
control arp	address resolution display and	arp(1M)
ARP	Address Resolution Protocol	ARP(7)
rarpd DARPA Reverse	Address Resolution Protocol server	rarpd(1M)
domain ethers Ethernet	address to hostname database or	ethers(4)
rfadmin Remote File Sharing domain	administration	rfadmin(1M)
fumount forced unmount of an	advertised resource	fumount(1M)
services Internet services and	aliases	services(4)
control	arp address resolution display and	arp(1M)
	ARP Address Resolution Protocol	ARP(7)
resources rmnttry	attempt to mount queued remote	rmnttry(1M)
systems automount	automatically mount NFS file	automount(1M)
file systems	automount automatically mount NFS	automount(1M)
systems share make local resource	available for mounting by remote	share(1M)
local systems dfshares list	available resources from remote or	dfshares(1M)
systems dfshares list	available resources from remote	dfshares(1M)
share make local resources	available	share(1M)
hosts host name data	base	hosts(4)
netmasks network mask data	base	netmasks(4)
networks network name data	base	networks(4)
protocols protocol name data	base	protocols(4)
comsat, in.comsat	biff server	comsat(1M)
	biod NFS daemon	biod(1M)
request rumount	cancel queued remote resource	rumount(1M)
password rfpasswd	change Remote File Sharing host	rfpasswd(1M)
chkey	change user encryption key	chkey(1)
	chkey change user encryption key	chkey(1)
rexec return stream to a remote	command	rexec(3N)
dfstab file containing	commands for sharing resources	dfstab(4)
	comsat, in.comsat biff server	comsat(1M)
routines resolv.conf	configuration file for name server	resolv.conf(4)
TCP/IP strcf STREAMS	Configuration File for STREAMS	strcf(4)
parameters ifconfig	configure network interface	ifconfig(1M)
resources dfstab file	containing commands for sharing	dfstab(4)
arp address resolution display and	control	arp(1M)
ICMP Internet	Control Message Protocol	ICMP(7)
TCP Internet Transmission	Control Protocol	TCP(7)
table htable	convert DoD Internet format host	htable(1M)
rcp remote file	copy	rcp(1)
database newkey	create a new key in the publickey	newkey(1M)
biod NFS	daemon	biod(1M)
inetd Internet services	daemon	inetd(1M)
lockd network lock	daemon	lockd(1M)
nfsd NFS	daemon	nfsd(1M)
rfudaemon Remote File Sharing	daemon process	rfudaemon(1M)
routed network routing	daemon	routed(1M)

Protocol server rarpd	DARPA Reverse Address Resolution	rarpd(1M)
telnetd	DARPA TELNET protocol server	telnetd(1M)
Protocol server tftpd	DARPA Trivial File Transfer	tftpd(1M)
tnamed, in.tnamed	DARPA trivial name server	tnamed(1M)
hosts host name	data base	hosts(4)
netmasks network mask	data base	netmasks(4)
networks network name	data base	networks(4)
protocols protocol name	data base	protocols(4)
netrc file for ftp remote login	data	netrc(4)
inetd.conf Internet servers	database	inetd.conf(4)
create a new key in the publickey	database newkey	newkey(1M)
ethers Ethernet address to hostname	database or domain	ethers(4)
publickey public key	database	publickey(4)
UDP Internet User	Datagram Protocol	UDP(7)
rdate set system	date from a remote host	rdate(1M)
keylogin	decrypt and store secret key	keylogin(1)
information	dfmounts display mounted resource	dfmounts(1M)
from remote or local systems	dfshares list available resources	dfshares(1M)
sharing resources	dfstab file containing commands for	dfstab(4)
whois Internet user name	directory service	whois(1)
fusage	disk access profiler	fusage(1M)
arp address resolution	display and control	arp(1M)
finger	display information about users	finger(1)
information dfmounts	display mounted resource	dfmounts(1M)
information rmntstat	display mounted resource	rmntstat(1M)
fstypes file that registers	distributed file system packages	fstypes(4)
domain and network names	dname print Remote File Sharing	dname(1M)
/res_mkquery, res_send, res_init,	dn_comp, dn_expand resolver/	resolver(3N)
/res_send, res_init, dn_comp,	dn_expand resolver routines	resolver(3N)
a host gettable get	DoD Internet format host table from	gettable(1M)
htable convert	DoD Internet format host table	htable(1M)
rfadmin Remote File Sharing	domain administration	rfadmin(1M)
dname print Remote File Sharing	domain and network names	dname(1M)
address to hostname database or	domain ethers Ethernet	ethers(4)
named, in.named Internet	domain name server	named(1M)
hosts ping send ICMP	ECHO_REQUEST packets to network	ping(1M)
chkey change user	encryption key	chkey(1)
rfstop stop the Remote File Sharing	environment	rfstop(1M)
ethers	Ethernet address mapping operations	ethers(3N)
database or domain ethers	Ethernet address to hostname	ethers(4)
operations	ethers Ethernet address mapping	ethers(3N)
database or domain	ethers Ethernet address to hostname	ethers(4)
rexecd remote	execution server	rexecd(1M)
inet Internet protocol	family	inet(7)
sharing resources dfstab	file containing commands for	dfstab(4)
rcp remote	file copy	rcp(1)
netrc	file for ftp remote login data	netrc(4)

resolv.conf configuration	file for name server routines	resolv.conf(4)
strcf STREAMS Configuration	File for STREAMS TCP/IP	strcf(4)
File Sharing name server master	file rfmaster Remote	rfmaster(4)
rfudaemon Remote	File Sharing daemon process	rfudaemon(1M)
rfadmin Remote	File Sharing domain administration	rfadmin(1M)
names dname print Remote	File Sharing domain and network	dname(1M)
rfstop stop the Remote	File Sharing environment	rfstop(1M)
rfpasswd change Remote	File Sharing host password	rfpasswd(1M)
file rfmaster Remote	File Sharing name server master	rfmaster(4)
nsquery Remote	File Sharing name server query	nsquery(1M)
script rfuadmin Remote	File Sharing notification shell	rfuadmin(1M)
rumountall mount, unmount Remote	File Sharing resources rmountall,	rmountall(1M)
rfstart start Remote	File Sharing	rfstart(1M)
idload Remote	File Sharing user and group mapping	idload(1M)
file that registers distributed	file system packages fstypes	fstypes(4)
sharetab shared	file system table	sharetab(4)
automount automatically mount NFS	file systems	automount(1M)
file system packages fstypes	file that registers distributed	fstypes(4)
ftp	file transfer program	ftp(1)
tftp trivial	file transfer program	tftp(1)
ftpd	file transfer protocol server	ftpd(1M)
tftpd DARPA Trivial	File Transfer Protocol server	tftpd(1M)
users	finger display information about	finger(1)
information server	fingerd, in.fingerd remote user	fingerd(1M)
resource fumount	forced unmount of an advertised	fumount(1M)
gettable get DoD Internet	format host table from a host	gettable(1M)
htable convert DoD Internet	format host table	htable(1M)
distributed file system packages	fstypes file that registers	fstypes(4)
	ftp file transfer program	ftp(1)
netrc file for	ftp remote login data	netrc(4)
	ftpd file transfer protocol server	ftpd(1M)
advertised resource	fumount forced unmount of an	fumount(1M)
	fusage disk access profiler	fusage(1M)
Protocol network interfaces if	general properties of Internet	if(7)
host table from a host	gettable get DoD Internet format	gettable(1M)
idload Remote File Sharing user and	group mapping	idload(1M)
Internet format host table from a	host gettable get DoD	gettable(1M)
hosts	host name data base	hosts(4)
rfpasswd change Remote File Sharing	host password	rfpasswd(1M)
rdate set system date from a remote	host	rdate(1M)
ruptime show	host status of local machines	ruptime(1)
gettable get DoD Internet format	host table from a host	gettable(1M)
htable convert DoD Internet format	host table	htable(1M)
ethers Ethernet address to	hostname database or domain	ethers(4)
hosts.equiv, .rhosts trusted	hosts by system and by user	hosts.equiv(4)
	hosts host name data base	hosts(4)
ECHO_REQUEST packets to network	hosts ping send ICMP	ping(1M)

by system and by user	hosts.equiv, .rhosts trusted hosts	hosts.equiv(4)
host table	htable convert DoD Internet format	htable(1M)
network hosts ping send	ICMP ECHO_REQUEST packets to	ping(1M)
Protocol	ICMP Internet Control Message	ICMP(7)
group mapping	idload Remote File Sharing user and	idload(1M)
interface parameters	ifconfig configure network	ifconfig(1M)
comsat,	in.comsat biff server	comsat(1M)
	inet Internet protocol family	inet(7)
	inetd Internet services daemon	inetd(1M)
database	inetd.conf Internet servers	inetd.conf(4)
server fingerd,	in.fingerd remote user information	fingerd(1M)
finger display	information about users	finger(1)
dfmounts display mounted resource	information	dfmounts(1M)
rmntstat display mounted resource	information	rmntstat(1M)
fingerd, in.fingerd remote user	information server	fingerd(1M)
server named,	in.named Internet domain name	named(1M)
rwhod,	in.rwhod system status server	rwhod(1M)
talkd,	in.talkd server for talk program	talkd(1M)
nslookup query name servers	interactively	nslookup(1M)
lo software loopback network	interface	lo(7)
ifconfig configure network	interface parameters	ifconfig(1M)
the TELNET protocol telnet user	interface to a remote system using	telnet(1)
of Internet Protocol network	interfaces if general properties	if(7)
ICMP	Internet Control Message Protocol	ICMP(7)
named, in.named	Internet domain name server	named(1M)
host gettable get DoD	Internet format host table from a	gettable(1M)
htable convert DoD	Internet format host table	htable(1M)
inet	Internet protocol family	inet(7)
IP	Internet Protocol	IP(7)
if general properties of	Internet Protocol network/	if(7)
inetd.conf	Internet servers database	inetd.conf(4)
services	Internet services and aliases	services(4)
inetd	Internet services daemon	inetd(1M)
Protocol TCP	Internet Transmission Control	TCP(7)
UDP	Internet User Datagram Protocol	UDP(7)
service whois	Internet user name directory	whois(1)
tnamed,	in.tnamed DARPA trivial name server	tnamed(1M)
	IP Internet Protocol	IP(7)
chkey change user encryption	key	chkey(1)
publickey public	key database	publickey(4)
newkey create a new	key in the publickey database	newkey(1M)
keylogin decrypt and store secret	key	keylogin(1)
key	keylogin decrypt and store secret	keylogin(1)
for storing public and private	keys keyserv server	keyserv(1M)
and private keys	keyserv server for storing public	keyserv(1M)
slink streams	linker	slink(1M)
remote or local systems dfshares	list available resources from	dfshares(1M)

interface	lo software loopback network	lo(7)
ruptime show host status of	local machines	ruptime(1)
rwho who's logged in on	local machines	rwho(1)
mounting by remote/ share make	local resource available for	share(1M)
mounting by remote/ unshare make	local resource unavailable for	unshare(1M)
share make	local resources available	share(1M)
available resources from remote or	local systems dfshares list	dfshares(1M)
lockd network	lock daemon	lockd(1M)
	lockd network lock daemon	lockd(1M)
rwho who's	logged in on local machines	rwho(1)
netrc file for ftp remote	login data	netrc(4)
rlogin remote	login	rlogin(1)
rlogind remote	login server	rlogind(1M)
lo software	loopback network interface	lo(7)
ruptime show host status of local	machines	ruptime(1)
rwho who's logged in on local	machines	rwho(1)
route manually	manipulate the routing tables	route(1M)
tables route	manually manipulate the routing	route(1M)
Remote File Sharing user and group	mapping idload	idload(1M)
ethers Ethernet address	mapping operations	ethers(3N)
netmasks network	mask data base	netmasks(4)
Remote File Sharing name server	master file rfmaster	rfmaster(4)
ICMP Internet Control	Message Protocol	ICMP(7)
statd network status	monitor	statd(1M)
	mount mount remote resources	mount(1M)
automount automatically	mount NFS file systems	automount(1M)
rmnttry attempt to	mount queued remote resources	rmnttry(1M)
mount	mount remote NFS resources	mount(1M)
mount	mount remote resources	mount(1M)
mountd NFS	mount request server	mountd(1M)
resources rmountall, rumountall	mount, unmount Remote File Sharing	rmountall(1M)
	mountd NFS mount request server	mountd(1M)
dfmounts display	mounted resource information	dfmounts(1M)
rmntstat display	mounted resource information	rmntstat(1M)
make local resource available for	mounting by remote systems share	share(1M)
make local resource available for	mounting by remote systems share	share(1M)
make local resource unavailable for	mounting by remote systems unshare	unshare(1M)
rmount queue remote resource	mounts	rmount(1M)
shareall, unshareall share, unshare	multiple resources	shareall(1M)
hosts host	name data base	hosts(4)
networks network	name data base	networks(4)
protocols protocol	name data base	protocols(4)
whois Internet user	name directory service	whois(1)
rfmaster Remote File Sharing	name server master file	rfmaster(4)
named, in.named Internet domain	name server	named(1M)
nsquery Remote File Sharing	name server query	nsquery(1M)
resolv.conf configuration file for	name server routines	resolv.conf(4)

tnamed, in.tnamed DARPA trivial	name server	tnamed(1M)
nslookup query	name servers interactively	nslookup(1M)
name server	named, in.named Internet domain	named(1M)
File Sharing domain and network	names dname print Remote	dname(1M)
	netmasks network mask data base	netmasks(4)
data	netrc file for ftp remote login	netrc(4)
	netstat show network status	netstat(1M)
send ICMP ECHO_REQUEST packets to	network hosts ping	ping(1M)
lo software loopback	network interface	lo(7)
ifconfig configure	network interface parameters	ifconfig(1M)
properties of Internet Protocol	network interfaces if general	if(7)
lockd	network lock daemon	lockd(1M)
netmasks	network mask data base	netmasks(4)
networks	network name data base	networks(4)
Remote File Sharing domain and	network names dname print	dname(1M)
routed	network routing daemon	routed(1M)
system supporting for packet	network routing routing	routing(4)
statd	network status monitor	statd(1M)
netstat show	network status	netstat(1M)
	networks network name data base	networks(4)
publickey database	newkey create a new key in the	newkey(1M)
biod	NFS daemon	biod(1M)
nfsd	NFS daemon	nfsd(1M)
automount automatically mount	NFS file systems	automount(1M)
mountd	NFS mount request server	mountd(1M)
mount mount remote	NFS resources	mount(1M)
	nfsd NFS daemon	nfsd(1M)
rfuadmin Remote File Sharing	notification shell script	rfuadmin(1M)
interactively	nslookup query name servers	nslookup(1M)
server query	nsquery Remote File Sharing name	nsquery(1M)
ethers Ethernet address mapping	operations	ethers(3N)
registers distributed file system	packages fstypes file that	fstypes(4)
routing system supporting for	packet network routing	routing(4)
ping send ICMP ECHO_REQUEST	packets to network hosts	ping(1M)
configure network interface	parameters ifconfig	ifconfig(1M)
change Remote File Sharing host	password rfpasswd	rfpasswd(1M)
to network hosts	ping send ICMP ECHO_REQUEST packets	ping(1M)
and network names dname	print Remote File Sharing domain	dname(1M)
server for storing public and	private keys keyserv	keyserv(1M)
Remote File Sharing daemon	process rfudaemon	rfudaemon(1M)
fusage disk access	profiler	fusage(1M)
ftp file transfer	program	ftp(1)
talkd, in.talkd server for talk	program	talkd(1M)
tftp trivial file transfer	program	tftp(1)
network interfaces if general	properties of Internet Protocol	if(7)
ARP Address Resolution	Protocol	ARP(7)
inet Internet	protocol family	inet(7)

ICMP Internet Control Message	Protocol	ICMP(7)
IP Internet	Protocol	IP(7)
protocols	protocol name data base	protocols(4)
if general properties of Internet	Protocol network interfaces	if(7)
ftpd file transfer	protocol server	ftpd(1M)
DARPA Reverse Address Resolution	Protocol server rarpd	rarpd(1M)
telnetd DARPA TELNET	protocol server	telnetd(1M)
tftpd DARPA Trivial File Transfer	Protocol server	tftpd(1M)
TCP Internet Transmission Control	Protocol	TCP(7)
to a remote system using the TELNET	protocol telnet user interface	telnet(1)
trpt transliterate	protocol trace	trpt(1M)
UDP Internet User Datagram	Protocol	UDP(7)
	protocols protocol name data base	protocols(4)
keyserv server for storing	public and private keys	keyserv(1M)
publickey	public key database	publickey(4)
newkey create a new key in the	publickey database	newkey(1M)
	publickey public key database	publickey(4)
nslookup	query name servers interactively	nslookup(1M)
Remote File Sharing name server	query nsquery	nsquery(1M)
rmount	queue remote resource mounts	rmount(1M)
rumount cancel	queued remote resource request	rumount(1M)
rmnttry attempt to mount	queued remote resources	rmnttry(1M)
Resolution Protocol server	rarpd DARPA Reverse Address	rarpd(1M)
	rcp remote file copy	rcp(1)
host	rdate set system date from a remote	rdate(1M)
packages fstypes file that	registers distributed file system	fstypes(4)
rexec return stream to a	remote command	rexec(3N)
rexecd	remote execution server	rexecd(1M)
rcp	remote file copy	rcp(1)
rfudaemon	Remote File Sharing daemon process	rfudaemon(1M)
administration rfadmin	Remote File Sharing domain	rfadmin(1M)
network names dname print	Remote File Sharing domain and	dname(1M)
rfstop stop the	Remote File Sharing environment	rfstop(1M)
rfpasswd change	Remote File Sharing host password	rfpasswd(1M)
master file rfmaster	Remote File Sharing name server	rfmaster(4)
query nsquery	Remote File Sharing name server	nsquery(1M)
shell script rfuadmin	Remote File Sharing notification	rfuadmin(1M)
/rumountall mount, unmount	Remote File Sharing resources	rmountall(1M)
rfstart start	Remote File Sharing	rfstart(1M)
mapping idload	Remote File Sharing user and group	idload(1M)
rdate set system date from a	remote host	rdate(1M)
netrc file for ftp	remote login data	netrc(4)
rlogin	remote login	rlogin(1)
rlogind	remote login server	rlogind(1M)
mount mount	remote NFS resources	mount(1M)
list available resources from	remote or local systems dfshares	dfshares(1M)
rmount queue	remote resource mounts	rmount(1M)

rumount cancel queued	remote resource request	rumount(1M)
mount mount	remote resources	mount(1M)
rmnttry attempt to mount queued	remote resources	rmnttry(1M)
rsh	remote shell	rsh(1)
rshd	remote shell server	rshd(1M)
telnet user interface to a	remote system using the TELNET/	telnet(1)
list available resources from	remote systems dfshares	dfshares(1M)
resource available for mounting by	remote systems share make local	share(1M)
unavailable for mounting by	remote systems /make local resource	unshare(1M)
fingerd, in.fingerd	remote user information server	fingerd(1M)
cancel queued remote resource	request rumount	rumount(1M)
mountd NFS mount	request server	mountd(1M)
resolver, res_mkquery, res_send,	res_init, dn_comp, dn_expand/	resolver(3N)
dn_comp, dn_expand/ resolver,	res_mkquery, res_send, res_init,	resolver(3N)
arp address	resolution display and control	arp(1M)
ARP Address	Resolution Protocol	ARP(7)
rarpd DARPA Reverse Address	Resolution Protocol server	rarpd(1M)
name server routines	resolv.conf configuration file for	resolv.conf(4)
res_init, dn_comp, dn_expand/	resolver, res_mkquery, res_send,	resolver(3N)
res_init, dn_comp, dn_expand	resolver routines /res_send,	resolver(3N)
remote systems share make local	resource available for mounting by	share(1M)
forced unmount of an advertised	resource fumount	fumount(1M)
dfmounts display mounted	resource information	dfmounts(1M)
rmntstat display mounted	resource information	rmntstat(1M)
rmount queue remote	resource mounts	rmount(1M)
rumount cancel queued remote	resource request	rumount(1M)
by remote/ unshare make local	resource unavailable for mounting	unshare(1M)
share make local	resources available	share(1M)
containing commands for sharing	resources dfstab file	dfstab(4)
systems dfshares list available	resources from remote or local	dfshares(1M)
mount mount remote	resources	mount(1M)
attempt to mount queued remote	resources rmnttry	rmnttry(1M)
mount, unmount Remote File Sharing	resources rmountall, rumountall	rmountall(1M)
unshareall share, unshare multiple	resources shareall,	shareall(1M)
dn_expand/ resolver, res_mkquery,	res_send, res_init, dn_comp,	resolver(3N)
rexec	return stream to a remote command	rexec(3N)
server rarpd DARPA	Reverse Address Resolution Protocol	rarpd(1M)
command	rexec return stream to a remote	rexec(3N)
	rexecd remote execution server	rexecd(1M)
administration	rfadmin Remote File Sharing domain	rfadmin(1M)
server master file	rfmaster Remote File Sharing name	rfmaster(4)
host password	rfpasswd change Remote File Sharing	rfpasswd(1M)
	rfstart start Remote File Sharing	rfstart(1M)
environment	rfstop stop the Remote File Sharing	rfstop(1M)
notification shell script	rfuadmin Remote File Sharing	rfuadmin(1M)
daemon process	rfudaemon Remote File Sharing	rfudaemon(1M)
by user hosts.equiv,	.rhosts trusted hosts by system and	hosts.equiv(4)

aliases	services Internet services and	services(4)
rdate	set system date from a remote host	rdate(1M)
for mounting by remote systems	share make local resource available	share(1M)
shareall, unshareall	share, unshare multiple resources	shareall(1M)
multiple resources	shareall, unshareall share, unshare	shareall(1M)
sharetab	shared file system table	sharetab(4)
	sharetab shared file system table	sharetab(4)
rfudaemon Remote File	Sharing daemon process	rfudaemon(1M)
rfadmin Remote File	Sharing domain administration	rfadmin(1M)
dname print Remote File	Sharing domain and network names	dname(1M)
rfstop stop the Remote File	Sharing environment	rfstop(1M)
rfpasswd change Remote File	Sharing host password	rfpasswd(1M)
rfmaster Remote File	Sharing name server master file	rfmaster(4)
nsquery Remote File	Sharing name server query	nsquery(1M)
rfuadmin Remote File	Sharing notification shell script	rfuadmin(1M)
dfstab file containing commands for	sharing resources	dfstab(4)
mount, unmount Remote File	Sharing resources /rumountall	rmountall(1M)
rfstart start Remote File	Sharing	rfstart(1M)
idload Remote File	Sharing user and group mapping	idload(1M)
rsh remote	shell	rsh(1)
Remote File Sharing notification	shell script rfuadmin	rfuadmin(1M)
rshd remote	shell server	rshd(1M)
ruptime	show host status of local machines	ruptime(1)
netstat	show network status	netstat(1M)
	slink streams linker	slink(1M)
lo	software loopback network interface	lo(7)
rfstart	start Remote File Sharing	rfstart(1M)
	statd network status monitor	statd(1M)
statd network	status monitor	statd(1M)
netstat show network	status	netstat(1M)
ruptime show host	status of local machines	ruptime(1)
rwhod, in.rwhod system	status server	rwhod(1M)
environment rfstop	stop the Remote File Sharing	rfstop(1M)
keylogin decrypt and	store secret key	keylogin(1)
keyserv server for	storing public and private keys	keyserv(1M)
for STREAMS TCP/IP	strcf STREAMS Configuration File	strcf(4)
rexec return	stream to a remote command	rexec(3N)
STREAMS TCP/IP strcf	STREAMS Configuration File for	strcf(4)
slink	streams linker	slink(1M)
STREAMS Configuration File for	STREAMS TCP/IP strcf	strcf(4)
routing routing system	supporting for packet network	routing(4)
.rhosts trusted hosts by	system and by user hosts.equiv,	hosts.equiv(4)
rdate set	system date from a remote host	rdate(1M)
that registers distributed file	system packages fstypes file	fstypes(4)
rwhod, in.rwhod	system status server	rwhod(1M)
network routing routing	system supporting for packet	routing(4)
sharetab shared file	system table	sharetab(4)

telnet user interface to a remote	system using the TELNET protocol	telnet(1)
automatically mount NFS file	systems automount	automount(1M)
available resources from remote	systems dfshares list	dfshares(1M)
available for mounting by remote	systems share make local resource	share(1M)
unavailable for mounting by remote	systems /make local resource	unshare(1M)
get DoD Internet format host	table from a host gettable	gettable(1M)
convert DoD Internet format host	table htable	htable(1M)
sharetab shared file system	table	sharetab(4)
manually manipulate the routing	tables route	route(1M)
talkd, in.talkd server for	talk program	talkd(1M)
talk	talk to another user	talk(1)
program	talkd, in.talkd server for talk	talkd(1M)
Protocol	TCP Internet Transmission Control	TCP(7)
Configuration File for STREAMS	TCP/IP strcf STREAMS	strcf(4)
telnetd DARPA	TELNET protocol server	telnetd(1M)
to a remote system using the	TELNET protocol /user interface	telnet(1)
system using the TELNET protocol	telnet user interface to a remote	telnet(1)
server	telnetd DARPA TELNET protocol	telnetd(1M)
	tftp trivial file transfer program	tftp(1)
Protocol server	tftpd DARPA Trivial File Transfer	tftpd(1M)
name server	tnamed, in.tnamed DARPA trivial	tnamed(1M)
trpt transliterate protocol	trace	trpt(1M)
ftp file	transfer program	ftp(1)
tftp trivial file	transfer program	tftp(1)
ftpd file	transfer protocol server	ftpd(1M)
tftpd DARPA Trivial File	Transfer Protocol server	tftpd(1M)
trpt	transliterate protocol trace	trpt(1M)
TCP Internet	Transmission Control Protocol	TCP(7)
tftp	trivial file transfer program	tftp(1)
server tftpd DARPA	Trivial File Transfer Protocol	tftpd(1M)
tnamed, in.tnamed DARPA	trivial name server	tnamed(1M)
	trpt transliterate protocol trace	trpt(1M)
hosts.equiv, .rhosts	trusted hosts by system and by user	hosts.equiv(4)
	UDP Internet User Datagram Protocol	UDP(7)
unshare make local resource	unavailable for mounting by remote/	unshare(1M)
fumount forced	unmount of an advertised resource	fumount(1M)
rmountall, rumountall mount,	unmount Remote File Sharing/	rmountall(1M)
unavailable for mounting by remote/	unshare make local resource	unshare(1M)
shareall, unshareall share,	unshare multiple resources	shareall(1M)
resources shareall,	unshareall share, unshare multiple	shareall(1M)
idload Remote File Sharing	user and group mapping	idload(1M)
UDP Internet	User Datagram Protocol	UDP(7)
chkey change	user encryption key	chkey(1)
trusted hosts by system and by	user hosts.equiv, .rhosts	hosts.equiv(4)
fingerd, in.fingerd remote	user information server	fingerd(1M)
using the TELNET protocol telnet	user interface to a remote system	telnet(1)
whois Internet	user name directory service	whois(1)

NAME

chkey – change user encryption key

SYNOPSIS

`chkey`

DESCRIPTION

The `chkey` command prompts for a password and uses it to encrypt a new user encryption key. The encrypted key is stored in the `publickey`(4) database.

This command should be executed only on the master server for the `publickey`(4) database.

SEE ALSO

`keylogin`(1), `keylogout`(1), `publickey`(4), `keyserv`(1M), `newkey`(1).

NAME

`finger` – display information about local and remote users

SYNOPSIS

`finger` [`-bfhilmpqsw`] *username*...

`finger` [`-l`] *username@hostname*... (`TC/IP`)

DESCRIPTION

By default, the `finger` command displays information about each , logged-in user, including login name, full name, terminal name (prepended with a '*' if write-permission is denied), idle time, login time, and location if known.

Idle time is minutes if it is a single integer, hours and minutes if a ':' is present, or days and hours if a `d` is present.

When one or more *username* arguments are given, more detailed information is given for each *username* specified, whether they are logged in or not. *username* must be that of a local user, and may be a first or last name, or an account name. When `finger` is used to find users on a remote device, the user and the name of the remote device are specified in the form *username@hostname*. Information is presented in a multi-line format, and includes, in addition to the information mentioned above:

> the user's home directory and login shell
>
> time the user logged in if currently logged in, or the time the user last logged in if not, as well as the terminal or host from which the user logged in and, if a terminal.
>
> last time the user received mail, and the last time the user read their mail
>
> any plan contained in the file `.plan` in the user's home directory
>
> and any project on which the user is working described in the file `.project` (also in the user's home directory)

The following options are available:

`-b` Suppress printing the user's home directory and shell in a long format printout.

`-f` Suppress printing the header that is normally printed in a non-long format printout.

`-h` Suppress printing of the `.project` file in a long format printout.

`-i` Force "idle" output format, which is similar to short format except that only the login name, terminal, login time, and idle time are printed.

`-l` Force long output format.

`-m` Match arguments only on user name (not first or last name).

`-p` Suppress printing of the `.plan` file in a long format printout.

`-q` Force quick output format, which is similar to short format except that only the login name, terminal, and login time are printed.

-s Force short output format.

-w Suppress printing the full name in a short format printout.

Within the TCP/IP network, the -l option can be used remotely.

FILES

/var/adm/utmp	who is logged in
/etc/passwd	for users' names
/var/adm/lastlog	last login times
~/.plan	plans
~/.project	projects

SEE ALSO

passwd(1), **who**(1), **whois**(1)

NOTES

Only the first line of the ~/.project file is printed.

NAME

`ftp` – file transfer program

SYNOPSIS

`ftp` [`-dgintv`] [*hostname*]

DESCRIPTION

The `ftp` command is the user interface to the ARPANET standard File Transfer Protocol (FTP). `ftp` transfers files to and from a remote network site.

The client host with which `ftp` is to communicate may be specified on the command line. If this is done, `ftp` immediately attempts to establish a connection to an FTP server on that host; otherwise, `ftp` enters its command interpreter and awaits instructions from the user. When `ftp` is awaiting commands from the user, it displays the prompt `ftp>`.

The following options may be specified at the command line, or to the command interpreter:

`-d` Enable debugging.

`-g` Disable filename globbing.

`-i` Turn off interactive prompting during multiple file transfers.

`-n` Do not attempt auto-login upon initial connection. If auto-login is not disabled, `ftp` checks the `.netrc` file in the user's home directory for an entry describing an account on the remote machine. If no entry exists, `ftp` will prompt for the login name of the account on the remote machine (the default is the login name on the local machine), and, if necessary, prompts for a password and an account with which to login.

`-t` Enable packet tracing (unimplemented).

`-v` Show all responses from the remote server, as well as report on data transfer statistics. This is turned on by default if `ftp` is running interactively with its input coming from the user's terminal.

The following commands can be specified to the command interpreter:

`!` [*command*]
: Run *command* as a shell command on the local machine. If no *command* is given, invoke an interactive shell.

`$` *macro-name* [*args*]
: Execute the macro *macro-name* that was defined with the `macdef` command. Arguments are passed to the macro unglobbed.

`account` [*passwd*]
: Supply a supplemental password required by a remote system for access to resources once a login has been successfully completed. If no argument is included, the user will be prompted for an account password in a non-echoing input mode.

append *local-file* [*remote-file*]
: Append a local file to a file on the remote machine. If *remote-file* is not specified, the local file name is used, subject to alteration by any `ntrans` or `nmap` settings. File transfer uses the current settings for representation type, file structure, and transfer mode.

ascii Set the representation type to network ASCII. This is the default type.

bell Sound a bell after each file transfer command is completed.

binary
: Set the representation type to image.

bye Terminate the FTP session with the remote server and exit `ftp`. An EOF will also terminate the session and exit.

case Toggle remote computer file name case mapping during `mget` commands. When `case` is on (default is off), remote computer file names with all letters in upper case are written in the local directory with the letters mapped to lower case.

cd *remote-directory*
: Change the working directory on the remote machine to *remote-directory*.

cdup Change the remote machine working directory to the parent of the current remote machine working directory.

close Terminate the FTP session with the remote server, and return to the command interpreter. Any defined macros are erased.

cr Toggle RETURN stripping during network ASCII type file retrieval. Records are denoted by a RETURN/LINEFEED sequence during network ASCII type file transfer. When `cr` is on (the default), RETURN characters are stripped from this sequence to conform with the UNIX system single LINEFEED record delimiter. Records on non-UNIX-system remote hosts may contain single LINEFEED characters; when an network ASCII type transfer is made, these LINEFEED characters may be distinguished from a record delimiter only when `cr` is off.

delete *remote-file*
: Delete the file *remote-file* on the remote machine.

debug
: Toggle debugging mode. When debugging is on, `ftp` prints each command sent to the remote machine, preceded by the string -->.

dir [*remote-directory*] [*local-file*]
: Print a listing of the directory contents in the directory, *remote-directory*, and, optionally, placing the output in *local-file*. If no directory is specified, the current working directory on the remote machine is used. If no local file is specified, or *local-file* is –, output is sent to the terminal.

disconnect
: A synonym for `close`.

`form` [*format-name*]
: Set the carriage control format subtype of the representation type to *format-name*. The only valid *format-name* is `non-print`, which corresponds to the default non-print subtype.

`get` *remote-file* [*local-file*]
: Retrieve the *remote-file* and store it on the local machine. If the local file name is not specified, it is given the same name it has on the remote machine, subject to alteration by the current `case`, `ntrans`, and `nmap` settings. The current settings for representation type, file structure, and transfer mode are used while transferring the file.

`glob` Toggle filename expansion, or globbing, for `mdelete`, `mget` and `mput`. If globbing is turned off, filenames are taken literally.

Globbing for `mput` is done as in `sh`(1). For `mdelete` and `mget`, each remote file name is expanded separately on the remote machine, and the lists are not merged.

Expansion of a directory name is likely to be radically different from expansion of the name of an ordinary file: the exact result depends on the remote operating system and FTP server, and can be previewed by doing `mls` *remote-files* –.

`mget` and `mput` are not meant to transfer entire directory subtrees of files. You can do this by transferring a `tar`(1) archive of the subtree (using a representation type of image as set by the `binary` command).

`hash` Toggle hash-sign (#) printing for each data block transferred. The size of a data block is 8192 bytes.

`help` [*command*]
: Print an informative message about the meaning of *command*. If no argument is given, `ftp` prints a list of the known commands.

`lcd` [*directory*]
: Change the working directory on the local machine. If no *directory* is specified, the user's home directory is used.

`ls` [*remote-directory*] [*local-file*]
: Print an abbreviated listing of the contents of a directory on the remote machine. If *remote-directory* is left unspecified, the current working directory is used. If no local file is specified, or if *local-file* is –, the output is sent to the terminal.

`macdef` *macro-name*
: Define a macro. Subsequent lines are stored as the macro *macro-name*; a null line (consecutive NEWLINE characters in a file or RETURN characters from the terminal) terminates macro input mode. There is a limit of 16 macros and 4096 total characters in all defined macros. Macros remain defined until a `close` command is executed.

The macro processor interprets `$` and `\` as special characters. A `$` followed by a number (or numbers) is replaced by the corresponding argument on the macro invocation command line. A `$` followed by an `i` signals that macro processor that the executing macro is to be looped. On the first pass `$i` is replaced by the first argument on the macro invocation command line, on the second pass it is replaced by the second argument, and so on. A \ followed by any character is replaced by that character. Use the \ to prevent special treatment of the `$`.

`mdelete` [*remote-files*]
Delete the *remote-files* on the remote machine.

`mdir` *remote-files local-file*
Like `dir`, except multiple remote files may be specified. If interactive prompting is on, `ftp` will prompt the user to verify that the last argument is indeed the target local file for receiving `mdir` output.

`mget` *remote-files*
Expand the *remote-files* on the remote machine and do a `get` for each file name thus produced. See `glob` for details on the filename expansion. Resulting file names will then be processed according to `case`, `ntrans`, and `nmap` settings. Files are transferred into the local working directory, which can be changed with `lcd` *directory*; new local directories can be created with `! mkdir` *directory*.

`mkdir` *directory-name*
Make a directory on the remote machine.

`mls` *remote-files local-file*
Like `ls`(1), except multiple remote files may be specified. If interactive prompting is on, `ftp` will prompt the user to verify that the last argument is indeed the target local file for receiving `mls` output.

`mode` [*mode-name*]
Set the transfer mode to *mode-name*. The only valid *mode-name* is `stream`, which corresponds to the default stream mode. This implementation only supports `stream`, and requires that it be specified.

`mput` *local-files*
Expand wild cards in the list of local files given as arguments and do a `put` for each file in the resulting list. See `glob` for details of filename expansion. Resulting file names will then be processed according to `ntrans` and `nmap` settings.

`nmap` [*inpattern outpattern*]
Set or unset the filename mapping mechanism. If no arguments are specified, the filename mapping mechanism is unset. If arguments are specified, remote filenames are mapped during `mput` commands and `put` commands issued without a specified remote target filename. If arguments are specified, local filenames are mapped during `mget` commands and `get` commands issued without a specified local target filename.

This command is useful when connecting to a non-UNIX-system remote host with different file naming conventions or practices. The mapping follows the pattern set by *inpattern* and *outpattern*. *inpattern* is a template for incoming filenames (which may have already been processed according to the `ntrans` and `case` settings). Variable templating is accomplished by including the sequences `$1`, `$2`, ..., `$9` in *inpattern*. Use \ to prevent this special treatment of the `$` character. All other characters are treated literally, and are used to determine the **nmap** *inpattern* variable values.

For example, given *inpattern* `$1.$2` and the remote file name `mydata.data`, `$1` would have the value `mydata`, and `$2` would have the value `data`.

The *outpattern* determines the resulting mapped filename. The sequences `$1`, `$2`, ..., `$9` are replaced by any value resulting from the *inpattern* template. The sequence `$0` is replaced by the original filename. Additionally, the sequence [*seq1* , *seq2*] is replaced by *seq1* if *seq1* is not a null string; otherwise it is replaced by *seq2*.

For example, the command `nmap $1.$2.$3 [$1,$2].[$2,file]` would yield the output filename `myfile.data` for input filenames `myfile.data` and `myfile.data.old`, `myfile.file` for the input filename `myfile`, and `myfile.myfile` for the input filename `myfile`. SPACE characters may be included in *outpattern*, as in the example `nmap $1 | sed "s/  *$//" > $1`. Use the \ character to prevent special treatment of the `$`, `[`, `]`, and `,`, characters.

`ntrans` [*inchars* [*outchars*]]

Set or unset the filename character translation mechanism. If no arguments are specified, the filename character translation mechanism is unset. If arguments are specified, characters in remote filenames are translated during `mput` commands and `put` commands issued without a specified remote target filename, and characters in local filenames are translated during `mget` commands and `get` commands issued without a specified local target filename.

This command is useful when connecting to a non-UNIX-system remote host with different file naming conventions or practices. Characters in a filename matching a character in *inchars* are replaced with the corresponding character in *outchars*. If the character's position in *inchars* is longer than the length of *outchars*, the character is deleted from the file name.

`open` *host* [*port*]

Establish a connection to the specified *host* FTP server. An optional port number may be supplied, in which case, `ftp` will attempt to contact an FTP server at that port. If the *auto-login* option is on (default setting), `ftp` will also attempt to automatically log the user in to the FTP server.

`prompt`

Toggle interactive prompting. Interactive prompting occurs during multiple file transfers to allow the user to selectively retrieve or store files. By default, prompting is turned on. If prompting is turned off, any `mget` or `mput` will transfer all files, and any `mdelete` will delete all files.

`proxy` *ftp-command*

Execute an FTP command on a secondary control connection. This command allows simultaneous connection to two remote FTP servers for transferring files between the two servers. The first `proxy` command should be an `open`, to establish the secondary control connection. Enter the command `proxy ?` to see other FTP commands executable on the secondary connection.

The following commands behave differently when prefaced by `proxy`: `open` will not define new macros during the auto-login process, `close` will not erase existing macro definitions, `get` and `mget` transfer files from the host on the primary control connection to the host on the secondary control connection, and `put`, `mputd`, and `append` transfer files from the host on the secondary control connection to the host on the primary control connection.

Third party file transfers depend upon support of the PASV command by the server on the secondary control connection.

`put` *local-file* [*remote-file*]
: Store a local file on the remote machine. If *remote-file* is left unspecified, the local file name is used after processing according to any `ntrans` or `nmap` settings in naming the remote file. File transfer uses the current settings for representation type, file structure, and transfer mode.

`pwd` Print the name of the current working directory on the remote machine.

`quit` A synonym for `bye`.

`quote` *arg1 arg2* ...
: Send the arguments specified, verbatim, to the remote FTP server. A single FTP reply code is expected in return. (The `remotehelp` command displays a list of valid arguments.)

  `quote` should be used only by experienced users who are familiar with the FTP protocol.

`recv` *remote-file* [*local-file*]
: A synonym for `get`.

`remotehelp` [*command-name*]
: Request help from the remote FTP server. If a *command-name* is specified it is supplied to the server as well.

`rename` *from to*
: Rename the file *from* on the remote machine to have the name *to*.

`reset` Clear reply queue. This command re-synchronizes command/reply sequencing with the remote FTP server. Resynchronization may be necessary following a violation of the FTP protocol by the remote server.

`rmdir` *directory-name*
: Delete a directory on the remote machine.

`runique`
: Toggle storing of files on the local system with unique filenames. If a file already exists with a name equal to the target local filename for a `get` or `mget` command, a `.1` is appended to the name. If the resulting name matches another existing file, a `.2` is appended to the original name. If this process continues up to `.99`, an error message is printed, and the transfer does not take place. The generated unique filename will be

reported. `runique` will not affect local files generated from a shell command. The default value is off.

`send` *local-file* [*remote-file*]
: A synonym for `put`.

`sendport`
: Toggle the use of `PORT` commands. By default, `ftp` will attempt to use a `PORT` command when establishing a connection for each data transfer. The use of `PORT` commands can prevent delays when performing multiple file transfers. If the `PORT` command fails, `ftp` will use the default data port. When the use of `PORT` commands is disabled, no attempt will be made to use `PORT` commands for each data transfer. This is useful when connected to certain FTP implementations that ignore `PORT` commands but incorrectly indicate they have been accepted.

`status`
: Show the current status of `ftp`.

`struct` [*struct-name*]
: Set the file structure to *struct-name*. The only valid *struct-name* is `file`, which corresponds to the default file structure. The implementation only supports `file`, and requires that it be specified.

`sunique`
: Toggle storing of files on remote machine under unique file names. The remote FTP server must support the `STOU` command for successful completion. The remote server will report the unique name. Default value is off.

`tenex` Set the representation type to that needed to talk to TENEX machines.

`trace` Toggle packet tracing (unimplemented).

`type` [*type-name*]
: Set the representation type to *type-name*. The valid *type-name*s are `ascii` for network ASCII, `binary` or `image` for image, and `tenex` for local byte size with a byte size of 8 (used to talk to TENEX machines). If no type is specified, the current type is printed. The default type is network ASCII.

`user` *user-name* [*password*] [*account*]
: Identify yourself to the remote FTP server. If the password is not specified and the server requires it, `ftp` will prompt the user for it (after disabling local echo). If an account field is not specified, and the FTP server requires it, the user will be prompted for it. If an account field is specified, an account command will be relayed to the remote server after the login sequence is completed if the remote server did not require it for logging in. Unless `ftp` is invoked with auto-login disabled, this process is done automatically on initial connection to the FTP server.

`verbose`
: Toggle verbose mode. In verbose mode, all responses from the FTP server are displayed to the user. In addition, if verbose mode is on, when a file transfer completes, statistics regarding the efficiency of the transfer are

reported. By default, verbose mode is on if `ftp`'s commands are coming from a terminal, and off otherwise.

? [*command*]
: A synonym for `help`.

Command arguments which have embedded spaces may be quoted with quote (") marks.

If any command argument which is not indicated as being optional is not specified, `ftp` will prompt for that argument.

ABORTING A FILE TRANSFER

To abort a file transfer, use the terminal interrupt key. Sending transfers will be immediately halted. Receiving transfers will be halted by sending an FTP protocol **ABOR** command to the remote server, and discarding any further data received. The speed at which this is accomplished depends upon the remote server's support for **ABOR** processing. If the remote server does not support the **ABOR** command, an `ftp>` prompt will not appear until the remote server has completed sending the requested file.

The terminal interrupt key sequence will be ignored when `ftp` has completed any local processing and is awaiting a reply from the remote server. A long delay in this mode may result from the **ABOR** processing described above, or from unexpected behavior by the remote server, including violations of the ftp protocol. If the delay results from unexpected remote server behavior, the local `ftp` program must be killed by hand.

FILE NAMING CONVENTIONS

Local files specified as arguments to `ftp` commands are processed according to the following rules.

1) If the file name – is specified, the standard input (for reading) or standard output (for writing) is used.

2) If the first character of the file name is |, the remainder of the argument is interpreted as a shell command. `ftp` then forks a shell, using popen(3S) with the argument supplied, and reads (writes) from the standard output (standard input) of that shell. If the shell command includes SPACE characters, the argument must be quoted; for example `"| ls -lt"`. A particularly useful example of this mechanism is: `"dir | more"`.

3) Failing the above checks, if globbing is enabled, local file names are expanded according to the rules used in the sh(1); see the `glob` command. If the `ftp` command expects a single local file (for example, `put`), only the first filename generated by the globbing operation is used.

4) For `mget` commands and `get` commands with unspecified local file names, the local filename is the remote filename, which may be altered by a `case`, `ntrans`, or `nmap` setting. The resulting filename may then be altered if `runique` is on.

5) For `mput` commands and `put` commands with unspecified remote file names, the remote filename is the local filename, which may be altered by a `ntrans` or `nmap` setting. The resulting filename may then be altered by the remote server if `sunique` is on.

FILE TRANSFER PARAMETERS

The FTP specification specifies many parameters which may affect a file transfer.

The representation type may be one of network ASCII, EBCDIC, image, or local byte size with a specified byte size (for PDP-10's and PDP-20's mostly). The network ASCII and EBCDIC types have a further subtype which specifies whether vertical format control (NEWLINE characters, form feeds, etc.) are to be passed through (non-print), provided in TELNET format (TELNET format controls), or provided in ASA (FORTRAN) (carriage control (ASA)) format. `ftp` supports the network ASCII (subtype non-print only) and image types, plus local byte size with a byte size of 8 for communicating with TENEX machines.

The file structure may be one of `file` (no record structure), `record`, or `page`. `ftp` supports only the default value, which is `file`.

The transfer mode may be one of `stream`, `block`, or `compressed`. `ftp` supports only the default value, which is `stream`.

SEE ALSO

`ls`(1), `rcp`(1), `tar`(1), `sh`(1), `ftpd`(1M), `popen`(3S), `netrc`(4).

NOTES

Correct execution of many commands depends upon proper behavior by the remote server.

An error in the treatment of carriage returns in the 4.2 BSD code handling transfers with a representation type of network ASCII has been corrected. This correction may result in incorrect transfers of binary files to and from 4.2 BSD servers using a representation type of network ASCII. Avoid this problem by using the image type.

NAME

keylogin – decrypt and store secret key

SYNOPSIS

`keylogin`

DESCRIPTION

The **keylogin** command prompts for a password, and uses it to decrypt the user's secret key stored in the **publickey**(4) database. Once decrypted, the user's key is stored by the local key server process, **keyserv**(1M), to be used by any secure network service, such as NFS.

SEE ALSO

chkey(1), **keylogout**(1), **publickey**(4), **keyserv**(1M), **newkey**(1).

NAME

`rcp` – remote file copy

SYNOPSIS

`rcp` [`-p`] *filename1 filename2*
`rcp` [`-pr`] *filename...directory*

DESCRIPTION

The `rcp` command copies files between machines. Each *filename* or *directory* argument is either a remote file name of the form:

hostname:path

or a local file name (containing no : characters, or a / before any : characters).

If a *filename* is not a full path name, it is interpreted relative to your home directory on *hostname*. A *path* on a remote host may be quoted (using \, ", or ') so that the metacharacters are interpreted remotely.

`rcp` does not prompt for passwords; your current local user name must exist on *hostname* and allow remote command execution by `rsh`(1).

`rcp` handles third party copies, where neither source nor target files are on the current machine. Hostnames may also take the form

username@hostname:filename

to use *username* rather than your current local user name as the user name on the remote host. `rcp` also supports Internet domain addressing of the remote host, so that:

username@host.domain:filename

specifies the username to be used, the hostname, and the domain in which that host resides. Filenames that are not full path names will be interpreted relative to the home directory of the user named *username*, on the remote host.

The destination hostname may also take the form *hostname.username:filename* to support destination machines that are running older versions of `rcp`.

The following options are available:

`-p` Attempt to give each copy the same modification times, access times, and modes as the original file.

`-r` Copy each subtree rooted at *filename*; in this case the destination must be a directory.

FILES

`$HOME/.profile`

SEE ALSO

`ftp`(1), `rlogin`(1), `rsh`(1), `hosts.equiv`(4).

NOTES

`rcp` is meant to copy between different hosts; attempting to `rcp` a file onto itself, as with:

```
rcp tmp/file myhost:/tmp/file
```

results in a severely corrupted file.

`rcp` does not detect all cases where the target of a copy might be a file in cases where only a directory should be legal.

`rcp` can become confused by output generated by commands in a `$HOME/.profile` on the remote host.

`rcp` requires that the source host have permission to execute commands on the remote host when doing third-party copies.

If you forget to quote metacharacters intended for the remote host you get an incomprehensible error message.

NAME

`rlogin` – remote login

SYNOPSIS

`rlogin` [`-L`] [`-8`] [`-e` *c*] [`-l` *username*] *hostname*

DESCRIPTION

`rlogin` establishes a remote login session from your terminal to the remote machine named *hostname*.

Hostnames are listed in the *hosts* database, which may be contained in the `/etc/hosts` file, the Internet domain name server, or in both. Each host has one official name (the first name in the database entry), and optionally one or more nicknames. Either official hostnames or nicknames may be specified in *hostname*.

Each remote machine may have a file named `/etc/hosts.equiv` containing a list of trusted hostnames with which it shares usernames. Users with the same username on both the local and remote machine may `rlogin` from the machines listed in the remote machine's `/etc/hosts.equiv` file without supplying a password. Individual users may set up a similar private equivalence list with the file `.rhosts` in their home directories. Each line in this file contains two names: a *hostname* and a *username* separated by a space. An entry in a remote user's `.rhosts` file permits the user named *username* who is logged into *hostname* to log in to the remote machine as the remote user without supplying a password. If the name of the local host is not found in the `/etc/hosts.equiv` file on the remote machine, and the local username and hostname are not found in the remote user's `.rhosts` file, then the remote machine will prompt for a password. Hostnames listed in `/etc/hosts.equiv` and `.rhosts` files must be the official hostnames listed in the hosts database; nicknames may not be used in either of these files.

To counter security problems, the `.rhosts` file must be owned by either the remote user or by root.

The remote terminal type is the same as your local terminal type (as given in your environment `TERM` variable). The terminal or window size is also copied to the remote system if the server supports the option, and changes in size are reflected as well. All echoing takes place at the remote site, so that (except for delays) the remote login is transparent. Flow control using `CTRL-S` and `CTRL-Q` and flushing of input and output on interrupts are handled properly.

The following options are available:

`-L` Allow the `rlogin` session to be run in litout mode.

`-8` Pass eight-bit data across the net instead of seven-bit data.

`-e` *c* Specify a different escape character, *c*, for the line used to disconnect from the remote host.

`-l` *username*
Specify a different *username* for the remote login. If you do not use this option, the remote username used is the same as your local username.

Escape Sequences

Lines that you type which start with the tilde character are escape sequences (the escape character can be changed using the **-e** options):

`~.`	Disconnect from the remote host — this is not the same as a logout, because the local host breaks the connection with no warning to the remote end.
susp	Suspend the login session (only if you are using a shell with Job Control). **susp** is your suspend character, usually see `tty`(1).

FILES

`/etc/passwd`	
`/usr/hosts/*`	for *hostname* version of the command
`/etc/hosts.equiv`	list of trusted hostnames with shared usernames
`$HOME/.rhosts`	private list of trusted hostname/username combinations

SEE ALSO

`rsh`(1), `stty`(1), `tty`(1), `named`(1M), `hosts`(4), `hosts.equiv`(4).

NOTES

When a system is listed in `hosts.equiv`, its security must be as good as local security. One insecure system listed in `hosts.equiv` can compromise the security of the entire system.

If you use a windowing terminal and you intend to run `layers`(1) on the remote system, then you must invoke `rlogin` with the `-8` option.

This implementation can only use the TCP network service.

NAME

`rsh` – remote shell

SYNOPSIS

`rsh` [`-n`] [`-l` *username*] *hostname command*

`rsh` *hostname* [`-n`] [`-l` *username*] *command*

hostname [`-n`] [`-l` *username*] *command*

DESCRIPTION

`rsh` connects to the specified *hostname* and executes the specified *command*. `rsh` copies its standard input to the remote command, the standard output of the remote command to its standard output, and the standard error of the remote command to its standard error. Interrupt, quit and terminate signals are propagated to the remote command; `rsh` normally terminates when the remote command does.

If you omit *command*, instead of executing a single command, `rsh` logs you in on the remote host using `rlogin`(1). Shell metacharacters which are not quoted are interpreted on the local machine, while quoted metacharacters are interpreted on the remote machine. See EXAMPLES.

Hostnames are given in the *hosts* database, which may be contained in the `/etc/hosts` file, the Internet domain name database, or both. Each host has one official name (the first name in the database entry) and optionally one or more nicknames. Official hostnames or nicknames may be given as *hostname*.

If the name of the file from which `rsh` is executed is anything other than `rsh`, `rsh` takes this name as its *hostname* argument. This allows you to create a symbolic link to `rsh` in the name of a host which, when executed, will invoke a remote shell on that host. By creating a directory and populating it wih symbolic links in the names of commonly used hosts, then including the directory in your shell's search path, you can run `rsh` by typing *hostname* to your shell.

Each remote machine may have a file named `/etc/hosts.equiv` containing a list of trusted hostnames with which it shares usernames. Users with the same username on both the local and remote machine may `rsh` from the machines listed in the remote machine's `/etc/hosts` file. Individual users may set up a similar private equivalence list with the file `.rhosts` in their home directories. Each line in this file contains two names: a *hostname* and a *username* separated by a space. The entry permits the user named *username* who is logged into *hostname* to use `rsh` to access the remote machine as the remote user. If the name of the local host is not found in the `/etc/hosts.equiv` file on the remote machine, and the local username and hostname are not found in the remote user's `.rhosts` file, then the access is denied. The hostnames listed in the `/etc/hosts.equiv` and `.rhosts` files must be the official hostnames listed in the `hosts` database; nicknames may not be used in either of these files.

`rsh` will not prompt for a password if access is denied on the remote machine unless the *command* argument is omitted.

OPTIONS

`-l` *username*
: Use *username* as the remote username instead of your local username. In the absence of this option, the remote username is the same as your local username.

`-n`
: Redirect the input of `rsh` to `/dev/null`. You sometimes need this option to avoid unfortunate interactions between `rsh` and the shell which invokes it. For example, if you are running `rsh` and invoke a `rsh` in the background without redirecting its input away from the terminal, it will block even if no reads are posted by the remote command. The -n option will prevent this.

The type of remote shell (`sh`, `rsh`, or other) is determined by the user's entry in the file `/etc/passwd` on the remote system.

EXAMPLES

The command:

```
rsh lizard cat lizard.file >> example.file
```

appends the remote file `lizard.file` from the machine called "lizard" to the file called `example.file` on the machine called "example," while the command:

```
rsh lizard cat lizard.file ">>" lizard.file2
```

appends the file `lizard.file` on the machine called "lizard" to the file `another.lizard.file` which also resides on the machine called "lizard."

FILES

`/etc/hosts`
`/etc/passwd`

SEE ALSO

`rlogin`(1), `vi`(1), `named`(1M), `hosts`(4), `hosts.equiv`(4).

NOTES

When a system is listed in `hosts.equiv`, its security must be as good as local security. One insecure system listed in `hosts.equiv` can compromise the security of the entire system.

You cannot run an interactive command [such as `vi`(1)]; use `rlogin` if you wish to do so.

Stop signals stop the local `rsh` process only; this is arguably wrong, but currently hard to fix for reasons too complicated to explain here.

The current local environment is not passed to the remote shell.

Sometimes the -n option is needed for reasons that are less than obvious. For example, the command:

```
rsh somehost dd if=/dev/nrmt0 bs=20b | tar xvpBf -
```

will put your shell into a strange state. Evidently, what happens is that the `tar` terminates before the `rsh`. The `rsh` then tries to write into the "broken pipe" and, instead of terminating neatly, proceeds to compete with your shell for its standard input. Invoking `rsh` with the -n option avoids such incidents.

This bug occurs only when `rsh` is at the beginning of a pipeline and is not reading standard input. Do not use the -n if `rsh` actually needs to read standard input. For example,

```
tar cf - . | rsh sundial dd of=/dev/rmt0 obs=20b
```

does not produce the bug. If you were to use the -n in a case like this, `rsh` would incorrectly read from `/dev/null` instead of from the pipe.

NAME

`ruptime` – show host status of local machines

SYNOPSIS

`ruptime` [`-alrtu`]

DESCRIPTION

`ruptime` gives a status line like `uptime` for each machine on the local network; these are formed from packets broadcast by each host on the network once a minute.

Machines for which no status report has been received for 5 minutes are shown as being down.

Normally, the listing is sorted by host name, but this order can be changed by specifying one of the options listed below.

The following options are available:

`-a`	Count even those users who have been idle for an hour or more.
`-l`	Sort the display by load average.
`-r`	Reverse the sorting order.
`-t`	Sort the display by up time.
`-u`	Sort the display by number of users.

FILES

`/var/spool/rwho/whod.*`	data files

SEE ALSO

`rwho`(1), `rwhod`(1M).

NAME

`rusers` – who's logged in on local machines

SYNOPSIS

`rusers` [`-ahilu`] *host* ...

DESCRIPTION

The `rusers` command produces output similar to **who**(1), but for remote machines. The listing is in the order that responses are received, but this order can be changed by specifying one of the options listed below.

The default is to print out the names of the users logged in. When the `-l` flag is given, additional information is printed for each user, including idle time, when user logged in, and tty.

A remote host will only respond if it is running the `rusersd`(1M) daemon, which may be started up from `inetd`(1M) or `listen`(1M).

The following options are available:

`-a`	Give a report for a machine even if no users are logged on.
`-h`	Sort alphabetically by host name.
`-i`	Sort by idle time.
`-l`	Give a longer listing in the style of **who**(1).
`-u`	Sort by number of users.

SEE ALSO

`inetd`(1M), `listen`(1M), **pmadm**(1M), `rusersd`(1M), **sacadm**(1M), **who**(1).

NAME

rwho – who's logged in on local machines

SYNOPSIS

rwho [-a]

DESCRIPTION

The **rwho** command produces output similar to **who**(1), but for all machines on your network. If no report has been received from a machine for 5 minutes, **rwho** assumes the machine is down, and does not report users last known to be logged into that machine.

If a user has not typed to the system for a minute or more, **rwho** reports this idle time. If a user has not typed to the system for an hour or more, the user is omitted from the output of **rwho** unless the **-a** flag is given.

The **-a** option reports all users whether or not they have typed to the system in the past hour.

FILES

`/var/spool/rwho/whod.*`	information about other machines

SEE ALSO

`finger`(1), `ruptime`(1), **who**(1), **rwhod**(1M).

NOTES

Does not work through gateways.

This is unwieldy when the number of machines on the local net is large.

The **rwho** service daemon, **rwhod**(1M), must be enabled for this command to return useful results.

NAME

`talk` – talk to another user

SYNOPSIS

`talk` *username* [*ttyname*]

DESCRIPTION

`talk` is a visual communication program that copies lines from your terminal to that of a user on the same or on another host. *username* is that user's login name.

The program is architecture dependent; it works only between machines of the same architecture.

If you want to talk to a user who is logged in more than once, the *ttyname* argument may be used to indicate the appropriate terminal name.

When first called, `talk` sends the message:

```
Message from TalkDaemon@ her_machine at time ...
talk: connection requested by your_name@your_machine
talk: respond with: talk your_name@your_machine
```

to the user you wish to talk to. At this point, the recipient of the message should reply by typing:

```
talkyour_name@your_machine
```

It does not matter from which machine the recipient replies, as long as the login name is the same. Once communication is established, the two parties may type simultaneously, with their output appearing in separate windows. Typing Ctrl-L redraws the screen, while your erase, kill, and word kill characters will work in `talk` as normal. To exit, just type your interrupt character; `talk` then moves the cursor to the bottom of the screen and restores the terminal.

Permission to talk may be denied or granted by use of the `mesg`(1) command. At the outset talking is allowed. Certain commands, such as `pr`(1), disallow messages in order to prevent messy output.

FILES

`/etc/hosts`	to find the recipient's machine
`/var/adm/utmp`	to find the recipient's tty

SEE ALSO

`mail`(1), `mesg`(1), `pr`(1), `who`(1), `write`(1), `talkd`(1M).

NAME

telnet – user interface to a remote system using the TELNET protocol

SYNOPSIS

telnet [*host* [*port*]]

DESCRIPTION

telnet communicates with another host using the TELNET protocol. If telnet is invoked without arguments, it enters command mode, indicated by its prompt telnet>. In this mode, it accepts and executes the commands listed below. If it is invoked with arguments, it performs an open command (see "Telnet Commands" below) with those arguments.

Once a connection has been opened, telnet enters input mode. In this mode, text typed is sent to the remote host. The input mode entered will be either character at a time or line by line depending on what the remote system supports.

In character at a time mode, most text typed is immediately sent to the remote host for processing.

In line by line mode, all text is echoed locally, and (normally) only completed lines are sent to the remote host. The local echo character (initially ^E) may be used to turn off and on the local echo (this would mostly be used to enter passwords without the password being echoed).

In either mode, if the *localchars* toggle is TRUE (the default in line mode; see below), the user's quit, intr, and flush characters are trapped locally, and sent as TELNET protocol sequences to the remote side. There are options (see toggle, autoflush, and toggle, autosynch) which cause this action to flush subsequent output to the terminal (until the remote host acknowledges the TELNET sequence) and flush previous terminal input (in the case of quit and intr).

While connected to a remote host, telnet command mode may be entered by typing the telnet escape character (initially ^]). When in command mode, the normal terminal editing conventions are available.

USAGE

Telnet Commands

The following commands are available. Only enough of each command to uniquely identify it need be typed (this is also true for arguments to the mode, set, toggle, and display commands).

open *host* [*port*]
: Open a connection to the named host. If no port number is specified, telnet will attempt to contact a TELNET server at the default port. The host specification may be either a host name [see hosts(4)] or an Internet address specified in the dot notation [see inet(7)].

close Close any open TELNET session and exit telnet. An EOF (in command mode) will also close a session and exit.

quit Same as close, above.

`z` Suspend `telnet`. This command only works when the user is using a shell that supports job control, such as `sh`(1).

`mode` *type*
: *type* is either `line` (for line by line mode) or *character* (for character at a time mode). The remote host is asked for permission to go into the requested mode. If the remote host is capable of entering that mode, the requested mode will be entered.

`status`
: Show the current status of `telnet`. This includes the peer one is connected to, as well as the current mode.

`display` [*argument...*]
: Display all, or some, of the `set` and `toggle` values (see `toggle`, *arguments*).

`?` [*command*]
: Get help. With no arguments, `telnet` print a help summary. If a command is specified, `telnet` will print the help information for just that command.

`send` *arguments*
: Send one or more special character sequences to the remote host. The following are the arguments which may be specified (more than one argument may be specified at a time):

  `escape`
  : Send the current `telnet` escape character (initially `^]`).

  `synch` Send the TELNET `SYNCH` sequence. This sequence discards all previously typed (but not yet read) input on the remote system. This sequence is sent as TCP urgent data (and may not work if the remote system is a 4.2 BSD system — if it does not work, a lower case r may be echoed on the terminal).

  `brk` Send the TELNET `BRK` (Break) sequence, which may have significance to the remote system.

  `ip` Send the TELNET `IP` (Interrupt Process) sequence, which aborts the currently running process on the remote system.

  `ao` Sends the TELNET `AO` (Abort Output) sequence, which flushes all output `from` the remote system to the user's terminal.

  `ayt` Sends the TELNET `AYT` (Are You There) sequence, to which the remote system may or may not choose to respond.

  `ec` Sends the TELNET `EC` (Erase Character) sequence, which erases the last character entered.

  `el` Sends the TELNET `EL` (Erase Line) sequence, which should cause the remote system to erase the line currently being entered.

  `ga` Sends the TELNET `GA` (Go Ahead) sequence, which likely has no significance to the remote system.

nop Sends the TELNET NOP (No Operation) sequence.

? Prints out help information for the send command.

set *argument value*
Set any one of a number of telnet variables to a specific value. The special value off turns off the function associated with the variable. The values of variables may be interrogated with the display command. The variables which may be specified are:

echo This is the value (initially ^E) which, when in line by line mode, toggles between doing local echoing of entered characters (for normal processing), and suppressing echoing of entered characters (for example, entering a password).

escape
This is the telnet escape character (initially ^]) which enters telnet command mode (when connected to a remote system).

interrupt
If telnet is in localchars mode (see toggle localchars) and the interrupt character is typed, a TELNET IP sequence (see send and ip) is sent to the remote host. The initial value for the interrupt character is taken to be the terminal's intr character.

quit If telnet is in localchars mode (see toggle localchars) and the quit character is typed, a TELNET BRK sequence (see send, brk) is sent to the remote host. The initial value for the quit character is taken to be the terminal's quit character.

flushoutput
If telnet is in localchars mode (see toggle localchars) and the flushoutput character is typed, a TELNET AO sequence (see send, ao) is sent to the remote host. The initial value for the flush character is taken to be the terminal's flush character.

erase If telnet is in localchars mode (see toggle localchars), and if telnet is operating in character at a time mode, then when this character is typed, a TELNET EC sequence (see send, ec) is sent to the remote system. The initial value for the erase character is taken to be the terminal's erase character.

kill If telnet is in localchars mode (see toggle localchars), and if telnet is operating in character at a time mode, then when this character is typed, a TELNET EL sequence (see send, el) is sent to the remote system. The initial value for the kill character is taken to be the terminal's kill character.

eof If telnet is operating in line by line mode, entering this character as the first character on a line sends this character to the remote system. The initial value of the eof character is taken to be the terminal's eof character.

toggle *arguments...*

Toggle (between TRUE and FALSE) various flags that control how telnet responds to events. More than one argument may be specified. The state of these flags may be interrogated with the display command. Valid arguments are:

autoflush
: If autoflush and localchars are both TRUE, then when the ao, intr, or quit characters are recognized (and transformed into TELNET sequences; see set for details), telnet refuses to display any data on the user's terminal until the remote system acknowledges (using a TELNET Timing Mark option) that it has processed those TELNET sequences. The initial value for this toggle is TRUE if the terminal user had not done an stty noflsh, otherwise FALSE [see stty(1)].

autosynch
: If autosynch and localchars are both TRUE, then when either the intr or *quit* characters are typed (see set for descriptions of the intr and quit characters), the resulting TELNET sequence sent is followed by the TELNET SYNCH sequence. This procedure should cause the remote system to begin throwing away all previously typed input until both of the TELNET sequences have been read and acted upon. The initial value of this toggle is FALSE.

crmod Toggle RETURN mode. When this mode is enabled, most RETURN characters received from the remote host will be mapped into a RETURN followed by a line feed. This mode does not affect those characters typed by the user, only those received from the remote host. This mode is not very useful unless the remote host only sends RETURN, but never LINEFEED. The initial value for this toggle is FALSE.

debug Toggle socket level debugging (useful only to the super-user). The initial value for this toggle is FALSE .

localchars
: If this is TRUE , then the flush, interrupt, quit, erase, and kill characters (see set) are recognized locally, and transformed into appropriate TELNET control sequences (respectively ao, ip, brk, ec, and el; see send). The initial value for this toggle is TRUE in line by line mode, and FALSE in character at a time mode.

netdata
: Toggle the display of all network data (in hexadecimal format). The initial value for this toggle is FALSE.

options
: Toggle the display of some internal telnet protocol processing

(having to do with TELNET options). The initial value for this toggle is FALSE.

? Display the legal `toggle` commands.

SEE ALSO

`rlogin`(1), `sh`(1), `stty`(1), `hosts`(4), `inet`(7).

NOTES

Do not attempt to run `layers`(1) while using `telnet`.

There is no adequate way for dealing with flow control.

On some remote systems, echo has to be turned off manually when in line by line mode.

There is enough settable state to justify a `.telnetrc` file.

In line by line mode, the terminal's EOF character is only recognized (and sent to the remote system) when it is the first character on a line.

NAME

`tftp` – trivial file transfer program

SYNOPSIS

`tftp` [*host*]

DESCRIPTION

`tftp` is the user interface to the Internet TFTP (Trivial File Transfer Protocol), which allows users to transfer files to and from a remote machine. The remote *host* may be specified on the command line, in which case `tftp` uses *host* as the default host for future transfers (see the connect command below).

USAGE

Commands

Once `tftp` is running, it issues the prompt `tftp>` and recognizes the following commands:

connect *host-name* [*port*]

Set the *host* (and optionally *port*) for transfers. The TFTP protocol, unlike the FTP protocol, does not maintain connections between transfers; thus, the connect command does not actually create a connection, but merely remembers what host is to be used for transfers. You do not have to use the connect command; the remote host can be specified as part of the get or put commands.

mode *transfer-mode*

Set the mode for transfers; *transfer-mode* may be one of `ascii` or `binary`. The default is `ascii`.

put *filename*
put *localfile remotefile*
put *filename1 filename2 . . . filenameN remote-directory*

Transfer a file, or a set of files, to the specified remote file or directory. The destination can be in one of two forms: a filename on the remote host if the host has already been specified, or a string of the form

host:*filename*

to specify both a host and filename at the same time. If the latter form is used, the specified host becomes the default for future transfers. If the remote-directory form is used, the remote host is assumed to be running the UNIX system.

get *filename*
get *remotename localname*
get *filename1 filename2 filename3 . . . filenameN*

Get a file or set of files (three or more) from the specified remote *sources*. *source* can be in one of two forms: a filename on the remote host if the host has already been specified, or a string of the form

host:*filename*

to specify both a host and filename at the same time. If the latter form is used, the last host specified becomes the default for future transfers.

`quit` Exit `tftp`. An EOF also exits.

`verbose`
Toggle verbose mode.

`trace` Toggle packet tracing.

`status`
Show current status.

`rexmt` *retransmission-timeout*
Set the per-packet retransmission timeout, in seconds.

`timeout` *total-transmission-timeout*
Set the total transmission timeout, in seconds.

`ascii` Shorthand for `mode ascii`.

`binary`
Shorthand for `mode binary`.

`? [` *command-name* `... ]`
Print help information.

NOTES

Because there is no user-login or validation within the TFTP protocol, many remote sites restrict file access in various ways. Approved methods for file access are specific to each site, and therefore cannot be documented here.

When using the `get` command to transfer multiple files from a remote host, three or more files must be specified. The command returns an error message if only two files are specified.

NAME

whois – Internet user name directory service

SYNOPSIS

whois [-h *host*] *identifier*

DESCRIPTION

whois searches for an Internet directory entry for an *identifier* which is either a name (such as "Smith") or a handle (such as "SRI-NIC"). To force a name-only search, precede the name with a period; to force a handle-only search, precede the handle with an exclamation point.

To search for a group or organization entry, precede the argument with * (an asterisk). The entire membership list of the group will be displayed with the record.

You may of course use an exclamation point and asterisk, or a period and asterisk together.

EXAMPLES

The command

```
whois Smith
```

looks for the name or handle SMITH.

The command

```
whois !SRI-NIC
```

looks for the handle SRI-NIC only.

The command

```
whois .Smith, John
```

looks for the name JOHN SMITH only.

Adding . . . to the name or handle argument will match anything from that point; that is, `ZU ...` will match ZUL, ZUM, and so on.

NAME

arp – address resolution display and control

SYNOPSIS

arp *hostname*

arp -a [*unix* [*kmem*]]

arp -d *hostname*

arp -s *hostname ether_address* [temp] [pub] [trail]

arp -f *filename*

DESCRIPTION

The **arp** program displays and modifies the Internet-to-Ethernet address translation tables used by the address resolution protocol [arp(7)].

With no flags, the program displays the current ARP entry for *hostname*. The host may be specified by name or by number, using Internet dot notation.

The following options are available:

-a Display all of the current ARP entries by reading the table from the file *kmem* (default /dev/kmem) based on the kernel file *unix* (default /stand/unix).

-d Delete an entry for the host called *hostname*. This option may only be used by the super-user.

-s Create an ARP entry for the host called *hostname* with the Ethernet address *ether_address*. The Ethernet address is given as six hexadecimal bytes separated by colons. The entry will be permanent unless the word temp is given in the command. If the word pub is given, the entry will be published, for instance, this system will respond to ARP requests for *hostname* even though the hostname is not its own. The word trail indicates that trailer encapsulations may be sent to this host.

-f Read the file named *filename* and set multiple entries in the ARP tables. Entries in the file should be of the form

hostname ether_address [temp] [pub] [trail]

with argument meanings as given above.

SEE ALSO

ifconfig(1M), arp(7).

NAME

automount – automatically mount NFS file systems

SYNOPSIS

automount [–nTv] [–D *name=value*] [–M *mount-directory*]
[–t *sub-options*] [*directory map* [–*mount-options*]] . . .

DESCRIPTION

automount is a daemon that automatically and transparently mounts an NFS file system as needed. It monitors attempts to access directories that are associated with an automount map, along with any directories or files that reside under them. When a file is to be accessed, the daemon mounts the appropriate NFS file system. You can assign a map to a directory using an entry in a direct automount map, or by specifying an indirect map on the command line.

automount uses a map to locate an appropriate NFS file server, exported file system, and mount options. It then mounts the file system in a temporary location, and replaces the file system entry for the directory or subdirectory with a symbolic link to the temporary location. If the file system is not accessed within an appropriate interval (five minutes by default), the daemon unmounts the file system and removes the symbolic link. If the indicated directory has not already been created, the daemon creates it, and then removes it upon exiting.

Since the name-to-location binding is dynamic, updates to an automount map are transparent to the user. This obviates the need to pre-mount shared file systems for applications that have hard coded references to files.

If you specify the dummy directory /–, automount treats the *map* argument that follows as the name of a direct map. In a direct map, each entry associates the full pathname of a mount point with a remote file system to mount.

If the directory argument is a pathname, the *map* argument points to a file called an indirect map. An indirect map contains a list of the subdirectories contained within the indicated directory. With an indirect map, it is these subdirectories that are mounted automatically. The *map* argument must be a full pathname.

The *–mount-options* argument, when supplied, is a comma-separated list of mount(1M) options, preceded by a hyphen (–). If mount options are specified in the indicated map, however, those in the map take precedence.

The following options are available:

–n Disable dynamic mounts. With this option, references through the automount daemon only succeed when the target file system has been previously mounted. This can be used to prevent NFS servers from cross-mounting each other.

–T Trace. Expand each NFS call and display it on the standard output.

–v Verbose. Log status messages to the console.

–D *name=value*
Assign *value* to the indicated automount (environment) variable.

`-M` *mount-directory*
: Mount temporary file systems in the named directory, instead of `/tmp_mnt`.

`-t` *sub-options*
: Specify *sub-options* as a comma-separated list that contains any combination of the following:

  `l` *duration*
  : Specify a *duration*, in seconds, that a file system is to remain mounted when not in use. The default is 5 minutes.

  `m` *interval*
  : Specify an *interval*, in seconds, between attempts to mount a file system. The default is 30 seconds.

  `w` *interval*
  : Specify an *interval*, in seconds, between attempts to unmount file systems that have exceeded their cached times. The default is 1 minute.

ENVIRONMENT

Environment variables can be used within an `automount` map. For instance, if `$HOME` appeared within a map, `automount` would expand it to its current value for the `HOME` variable.

If a reference needs to be protected from affixed characters, enclose the variable name within braces.

USAGE

Direct/Indirect Map Entry Format

A simple map entry (mapping) takes the form:

`directory` [*–mount-options*] *location* ...

where `directory` is the full pathname of the directory to mount when used in a direct map, or the basename of a subdirectory in an indirect map. *mount-options* is a comma-separated list of `mount` options, and *location* specifies a remote file system from which the directory may be mounted. In the simple case, *location* takes the form:

host`:`*pathname*

Multiple *location* fields can be specified, in which case `automount` sends multiple `mount` requests; `automount` mounts the file system from the first host that replies to the `mount` request. This request is first made to the local net or subnet. If there is no response, any connected server may respond.

If *location* is specified in the form:

host`:`*path*`:`*subdir*

host is the name of the host from which to mount the file system, *path* is the pathname of the directory to mount, and *subdir*, when supplied, is the name of a subdirectory to which the symbolic link is made. This can be used to prevent duplicate mounts when multiple directories in the same remote file system may be accessed. With a map for `/home` such as:

```
able  homeboy:/home/homeboy:able
baker homeboy:/home/homeboy:baker
```

and a user attempting to access a file in /home/able, automount mounts homeboy:/home/homeboy, but creates a symbolic link called /home/able to the able subdirectory in the temporarily mounted file system. If a user immediately tries to access a file in /home/baker, automount needs only to create a symbolic link that points to the baker subdirectory; /home/homeboy is already mounted. With the following map:

```
able  homeboy:/home/homeboy/able
baker homeboy:/home/homeboy/baker
```

automount would have to mount the file system twice.

A mapping can be continued across input lines by escaping the NEWLINE with a backslash. Comments begin with a # and end at the subsequent NEWLINE.

Directory Pattern Matching

The & character is expanded to the value of the directory field for the entry in which it occurs. In this case:

```
able  homeboy:/home/homeboy:&
```

the & expands to able.

The * character, when supplied as the directory field, is recognized as the catch-all entry. Such an entry resolves to any entry not previously matched. For instance, if the following entry appeared in the indirect map for /home:

```
*     &:/home/&
```

this would allow automatic mounts in /home of any remote file system whose location could be specified as:

hostname:/home/*hostname*

Hierarchical Mappings

A hierarchical mapping takes the form:

directory [/ [*subdirectory*]] [-*mount-options*] *location*...
    [/ [*subdirectory*] [-*mount-options*] *location*...]...

The initial /[*subdirectory*] is optional for the first location list and mandatory for all subsequent lists. The optional *subdirectory* is taken as a filename relative to the directory. If *subdirectory* is omitted in the first occurrence, the / refers to the directory itself.

Given the direct map entry:

```
/arch/src   \
/           -ro,intr  arch:/arch/src          alt:/arch/src      \
/1.0        -ro,intr  alt:/arch/src/1.0       arch:/arch/src/1.0  \
/1.0/man    -ro,intr  arch:/arch/src/1.0/man  alt:/arch/src/1.0/man
```

automount would automatically mount /arch/src, /arch/src/1.0 and /arch/src/1.0/man, as needed, from either arch or alt, whichever host responded first.

Direct Maps

A direct map contains mappings for any number of directories. Each directory listed in the map is automatically mounted as needed. The direct map as a whole is not associated with any single directory.

Indirect Maps

An indirect map allows you to specify mappings for the subdirectories you wish to mount under the `directory` indicated on the command line. It also obscures local subdirectories for which no mapping is specified. In an indirect map, each `directory` field consists of the basename of a subdirectory to be mounted as needed.

Included Maps

The contents of another map can be included within a map with an entry of the form

+mapname

where *mapname* is a filename.

Special Maps

The `-null` map is the only special map currently available. The `-null` map, when indicated on the command line, cancels a previous map for the directory indicated.

FILES

`/tmp_mnt`	parent directory for dynamically mounted file systems

SEE ALSO

`df`(1M), `mount`(1M), `passwd`(4).

NOTES

When it receives signal number 1, `automount` rereads the `/etc/mnttab` file to update its internal record of currently-mounted file systems. If a file system mounted with `automount` is unmounted by a `umount` command, `automount` should be forced to reread the file.

Shell filename expansion does not apply to objects not currently mounted.

Since `automount` is single-threaded, any request that is delayed by a slow or non-responding NFS server will delay all subsequent automatic mount requests until it completes.

Programs that read `/etc/mnttab` and then touch files that reside under automatic mount points will introduce further entries to the file.

NAME

biod – NFS daemon

SYNOPSIS

biod [*nservers*]

DESCRIPTION

biod starts *nservers* asynchronous block I/O daemons. This command is used on an NFS client to buffer read-ahead and write-behind. Four is the usual number for *nservers*.

The biod daemons are automatically invoked in run level 3.

SEE ALSO

mountd(1M), nfsd(1M), sharetab(4).

NAME

bootparamd – boot parameter server

SYNOPSIS

bootparamd [-d]

DESCRIPTION

bootparamd is a server process that provides information to diskless clients necessary for booting. It obtains its information from the /etc/bootparams file.

bootparamd can be invoked either by inetd(1M) or by the user.

The -d option displays the debugging information.

FILES

/etc/bootparams

SEE ALSO

inetd(1M)

NAME

`comsat`, `in.comsat` – biff server

SYNOPSIS

`in.comsat`

DESCRIPTION

`comsat` is the server process which listens for reports of incoming mail and notifies users who have requested to be told when mail arrives. It is invoked as needed by `inetd`(1M), and times out if inactive for a few minutes.

`comsat` listens on a datagram port associated with the **`biff`** service specification [see **`services`**(4)] for one line messages of the form

user@*mailbox-offset*

If the *user* specified is logged in to the system and the associated terminal has the owner execute bit turned on (by a **`biff y`**), the *offset* is used as a seek offset into the appropriate mailbox file and the first 7 lines or 560 characters of the message are printed on the user's terminal. Lines which appear to be part of the message header other than the `From`, `To`, `Date`, or `Subject` lines are not printed when displaying the message.

FILES

`/var/utmp`	who's logged on and on what terminals

SEE ALSO

`services`(4), `inetd`(1M).

NOTES

The message header filtering is prone to error.

NAME

`dfmounts` – display mounted resource information

SYNOPSIS

`dfmounts` [`-F` *fstype*] [`-h`] [`-o` *specific_options*] [*restriction* ...]

DESCRIPTION

`dfmounts` shows the local resources shared through a distributed file system *fstype* along with a list of clients that have the resource mounted. If *restriction* is not specified, `dfmounts` displays remote resources mounted on the local system. *Specific_options* as well as the availability and semantics of *restriction* are specific to particular distributed file system types.

If `dfmounts` is entered without arguments, all remote resources currently mounted on the local system are displayed, regardless of file system type.

The output of `dfmounts` consists of an optional header line (suppressed with the `-h` flag) followed by a list of lines containing whitespace-separated fields. For each resource, the fields are:

resource server pathname clients

where

resource	Specifies the resource name that must be given to the `mount`(1M) command.
server	Specifies the system from which the resource was mounted.
pathname	Specifies the pathname that must be given to the `share`(1M) command.
clients	Lists the systems, comma-separated, by which the resource was mounted. Clients are listed in the form *domain.*, *domain.system*, or *system*, depending on the file system type.

A field may be null. Each null field is indicated by a hyphen (–) unless the remainder of the fields on the line are also null. In this case, it may be omitted.

Fields with whitespace are enclosed in quotation marks (" ").

FILES

`/etc/dfs/fstypes`

SEE ALSO

`dfshares`(1M), `mount`(1M), `share`(1M), `unshare`(1M).

NAME

`dfmounts` – display mounted NFS resource information

SYNOPSIS

`dfmounts` [`-F nfs`] [`-h`] [*server* ...]

DESCRIPTION

`dfmounts` shows the local resources shared through Network File System, along with a list of clients that have mounted the resource. The `-F` flag may be omitted if NFS is the only file system type listed in the file `/etc/dfs/fstypes`.

The *server* option displays information about the resources mounted from each server, where *server* can be any system on the network. If no server is specified, then *server* is assumed to be the local system.

`dfmounts` without options displays all remote resources mounted on the local system, regardless of file system type.

The output of `dfmounts` consists of an optional header line (suppressed with the `-h` flag) followed by a list of lines containing whitespace-separated fields. For each resource, the fields are:

resource server pathname clients ...

where

resource	Specifies the resource name that must be given to the `mount`(1M) command.
server	Specifies the system from which the resource was mounted.
pathname	Specifies the pathname that must be given to the `share`(1M) command.
clients	A comma-separated list of systems that have mounted the resource.

FILES

`/etc/dfs/fstypes`

SEE ALSO

`mount`(1M), `share`(1M), `unshare`(1M).

NAME

dfmounts – display mounted RFS resource information

SYNOPSIS

dfmounts [–F rfs] [–h] [resource_name ...]

DESCRIPTION

dfmounts shows the local resources shared through Remote File Sharing, along with a list of clients that have mounted the resource. The –F flag may be omitted if rfs is the first file system type listed in the file /etc/dfs/fstypes.

The output of *dfmounts* consists of an optional header line (suppressed with the –h flag) followed by a list of lines containing whitespace-separated fields. For each resource, the fields are:

resource server pathname clients ...

where

resource	Specifies the resource name that must be given to the mount(1M) command.
server	Specifies the system from which the resource was mounted.
pathname	Specifies the full pathname that must be given to the share(1M) command.
clients	A comma-separated list of systems that have mounted the resource.

A field may be null. Each null field is indicated by a hyphen (–) unless the remainder of the fields on the line are also null. In this case, it may be omitted.

FILES

/etc/dfs/fstypes

SEE ALSO

dfmounts(1M), share(1M), unshare(1M), fumount(1M), mount(1M)

NAME

`dfshares` – list available resources from remote or local systems

SYNOPSIS

`dfshares` [`-F` *fstype*] [`-h`] [`-o` *specific_options*] [*server* ...]

DESCRIPTION

`dfshares` provides information about resources available to the host through a distributed file system of type *fstype*. *Specific_options* as well as the semantics of *server* are specific to particular distributed file systems.

If `dfshares` is entered without arguments, all resources currently shared on the local system are displayed, regardless of file system type.

The output of `dfshares` consists of an optional header line (suppressed with the `-h` flag) followed by a list of lines containing whitespace-separated fields. For each resource, the fields are:

resource server access transport description ...

where

resource	Specifies the resource name that must be given to the `mount`(1M) command.
server	Specifies the name of the system that is making the resource available.
access	Specifies the access permissions granted to the client systems, either `ro` (for read-only) or `rw` (for read/write). If **dfshares** cannot determine access permissions, a hyphen (–) is displayed.
transport	Specifies the transport provider over which the *resource* is shared.
description	Describes the resource.

A field may be null. Each null field is indicated by a hyphen (–) unless the remainder of the fields on the line are also null. In this case, it may be omitted.

FILES

`/etc/dfs/fstypes`

SEE ALSO

`dfmounts`(1M), `mount`(1M), `share`(1M), `unshare`(1M).

NAME

`dfshares` – list available NFS resources from remote systems

SYNOPSIS

`dfshares [-F nfs] [-h]` [*server* ...]

DESCRIPTION

`dfshares` provides information about resources available to the host through Network File System. The `-F` flag may be omitted if NFS is the first file system type listed in the file `/etc/dfs/fstypes`.

The query may be restricted to the output of resources available from one or more servers.

The *server* option displays information about the resources shared by each server, where *server* can be any system on the network. If no server is specified, then *server* is assumed to be the local system.

`dfshares` without arguments displays all resources shared on the local system, regardless of file system type.

The output of `dfshares` consists of an optional header line (suppressed with the `-h` flag) followed by a list of lines containing whitespace-separated fields. For each resource, the fields are:

resource server access transport

where

resource	Specifies the resource name that must be given to the mount(1M) command.
server	Specifies the system that is making the resource available.
access	Specifies the access permissions granted to the client systems; however, `dfshares` cannot determine this information for an NFS resource and populates the field with a hyphen (-).
transport	Specifies the transport provider over which the *resource* is shared; however, `dfshares` cannot determine this information for an NFS resource and populates the field with a hyphen (-).

FILES

`/etc/dfs/fstypes`

SEE ALSO

share(1M), unshare(1M), mount(1M).

NAME

`dfshares` – list available RFS resources from remote systems

SYNOPSIS

`dfshares [-F rfs] [-h]` [*server* ...]

DESCRIPTION

`dfshares` provides information about resources available to the host through Remote File Sharing. The `-F` flag may be omitted if RFS is the first file system type listed in the file `/etc/dfs/fstypes`.

The query may be restricted to the output of resources available from one or more servers. If no *server* is specified, all resources in the host's domain are displayed. A *server* may be given in the following form:

system	Specifies a system in the host's domain.
domain.	Specifies all systems in *domain*.
domain.system	Specifies *system* in *domain*.

The output of `dfshares` consists of an optional header line (suppressed with the `-h` flag) followed by a list of lines containing whitespace-separated fields. For each resource, the fields are:

resource server access transport description ...

where

resource	Specifies the resource name that must be given to the `mount`(1M) command.
server	Specifies the system that is making the resource available.
access	Specifies the access permissions granted to the client systems, either `ro` (for read-only) or `rw` (for read and write).
transport	Specifies the transport provider over which the *resource* is shared.
description	Describes the resource.

A field may be null. Each null field is indicated by a hyphen (–) unless the remainder of the fields on the line are also null. In this case, it may be omitted.

ERRORS

If your host machine cannot contact the domain name server, or the argument specified is syntactically incorrect, an error message is sent to standard error.

FILES

`/etc/dfs/fstypes`

SEE ALSO

`share`(1M), `unshare`(1M), `mount`(1M)

NAME

dname – print Remote File Sharing domain and network names

SYNOPSIS

dname [–D *domain*] [–N *netspeclist*] [–dna]

DESCRIPTION

dname prints or defines a host's Remote File Sharing domain name or the network(s) used by Remote File Sharing as transport provider(s). When used with **d**, **n**, or **a** options, **dname** can be run by any user to print the domain name, transport provider name(s), or both. Only a user with root permission can use the **–D** *domain* option to set the domain name for the host or **–N** *netspeclist* to set the network specification used for Remote File Sharing. *netspeclist* is a comma-separated list of transport providers (*tp1,tp2,...*). The value of each transport provider is the network device name, relative to the */dev* directory. For example, the STARLAN NETWORK uses **starlan**.

domain must consist of no more than 14 characters, consisting of any combination of letters (upper and lower case), digits, hyphens (–), and underscores (_).

When **dname** is used to change a domain name, the host's password is removed. The administrator will be prompted for a new password the next time Remote File Sharing is started [**rfstart**(1M)].

If **dname** is used with no options, it will default to **dname –d**.

NOTES

You cannot use the **–N** or **–D** options while Remote File Sharing is running.

SEE ALSO

rfstart(1M).

NAME

`fingerd`, `in.fingerd` – remote user information server

SYNOPSIS

`in.fingerd`

DESCRIPTION

`fingerd` implements the server side of the Name/Finger protocol, specified in RFC 742. The Name/Finger protocol provides a remote interface to programs which display information on system status and individual users. The protocol imposes little structure on the format of the exchange between client and server. The client provides a single command line to the finger server which returns a printable reply.

`fingerd` waits for connections on TCP port 79. Once connected it reads a single command line terminated by a <RETURN-LINE-FEED> which is passed to `finger`(1). `fingerd` closes its connections as soon as the output is finished.

If the line is null (only a RETURN-LINEFEED is sent) then `finger` returns a default report that lists all users logged into the system at that moment.

If a user name is specified (for instance, `eric`<RETURN-LINE-FEED>) then the response lists more extended information for only that particular user, whether logged in or not. Allowable names in the command line include both login names and user names. If a name is ambiguous, all possible derivations are returned.

FILES

`/var/utmp`	who is logged in
`/etc/passwd`	for users' names
`/var/adm/lastlog`	last login times
`$HOME/.plan`	plans
`$HOME/.project`	projects

SEE ALSO

`finger`(1)

Harrenstien, Ken, *NAME/FINGER*, RFC 742, Network Information Center, SRI International, Menlo Park, Calif., December 1977.

NOTES

Connecting directly to the server from a TIP or an equally narrow-minded TELNET-protocol user program can result in meaningless attempts at option negotiation being sent to the server, which will foul up the command line interpretation. `fingerd` should be taught to filter out IAC's and perhaps even respond negatively (IAC *will not)* to all option commands received.

NAME

ftpd – file transfer protocol server

SYNOPSIS

`in.ftpd` [`-dl`] [`-t`*timeout*] *host.socket*

DESCRIPTION

`ftpd` is the Internet File Transfer Protocol (FTP) server process. The server is invoked by the Internet daemon `inetd`(1M) each time a connection to the FTP service [see **services**(4)] is made, with the connection available as descriptor 0 and the host and socket the connection originated from (in hex and decimal respectively) as argument.

Inactive connections are timed out after 90 seconds.

The following options are available:

`-t`*timeout*
: Set the inactivity timeout period to *timeout*, in seconds. The FTP server will timeout an inactive session after 15 minutes.

Requests

The FTP server currently supports the following FTP requests; case is not distinguished.

Request	Description
ABOR	abort previous command
ACCT	specify account (ignored)
ALLO	allocate storage (vacuously)
APPE	append to a file
CDUP	change to parent of current working directory
CWD	change working directory
DELE	delete a file
HELP	give help information
LIST	give list files in a directory (`ls -lg`)
MKD	make a directory
MODE	specify data transfer *mode*
NLST	give name list of files in directory (`ls`)
NOOP	do nothing
PASS	specify password
PASV	prepare for server-to-server transfer
PORT	specify data connection port
PWD	print the current working directory

QUIT	terminate session
RETR	retrieve a file
RMD	remove a directory
RNFR	specify rename-from file name
RNTO	specify rename-to file name
STOR	store a file
STOU	store a file with a unique name
STRU	specify data transfer *structure*
TYPE	specify data transfer *type*
USER	specify user name
XCUP	change to parent of current working directory
XCWD	change working directory
XMKD	make a directory
XPWD	print the current working directory
XRMD	remove a directory

The remaining FTP requests specified in RFC 959 are recognized, but not implemented.

The FTP server will abort an active file transfer only when the ABOR command is preceded by a Telnet Interrupt Process (IP) signal and a Telnet Synch signal in the command Telnet stream, as described in RFC 959.

`ftpd` interprets file names according to the globbing conventions used by sh(1). This allows users to utilize the metacharacters: `* ? [ ] { } ~`

`ftpd` authenticates users according to three rules.

1) The user name must be in the password data base, `/etc/passwd`, and not have a null password. In this case a password must be provided by the client before any file operations may be performed.

2) If the user name appears in the file `/etc/ftpusers`, `ftp` access is denied.

3) `ftp` access is denied unless the user's shell (from `/etc/passwd`) is listed in the file `/etc/shells`, or the user's shell is one of the following:
```
/bin/sh
/bin/ksh
/bin/csh
/usr/bin/sh
/usr/bin/ksh
/usr/bin/csh
```

4) If the user name is anonymous or ftp, an anonymous FTP account must be present in the password file (user ftp). In this case the user is allowed to log in by specifying any password (by convention this is given as the client host's name).

In the last case, `ftpd` takes special measures to restrict the client's access privileges. The server performs a `chroot`(2) command to the home directory of the ftp user. In order that system security is not breached, it is recommended that the ftp subtree be constructed with care; the following rules are recommended.

home_directory
: Make the home directory owned by `ftp` and unwritable by anyone.

home_directory/`usr/bin`
: Make this directory owned by the super-user and unwritable by anyone. The program `ls`(1) must be present to support the list commands. This program should have mode 111.

home_directory/`etc`
: Make this directory owned by the super-user and unwritable by anyone. Copies of the files `passwd`(4), `group`(4), and `netconfig` must be present for the `ls` command to work properly. These files should be mode 444.

home_directory/`pub`
: Make this directory mode 777 and owned by `ftp`. Users should then place files which are to be accessible via the anonymous account in this directory.

home_directory/`dev`
: Make this directory owned by the super-user and unwritable by anyone. Change directories to this directory and do the following:

```
FTP="`grep ^ftp: /etc/passwd | cut -d: -f6`"
MAJORMINOR="`ls -l /dev/tcp | nawk '{ gsub(/,/, ""); print $5, $6}'`
mknod $FTP/dev/tcp c $MAJORMINOR
chmod 666 $FTP/dev/tcp
```

SEE ALSO

`ftp`(1), `getsockopt`(3N), `passwd`(4), `services`(4).

Postel, Jon, and Joyce Reynolds, *File Transfer Protocol (FTP)*, RFC 959, Network Information Center, SRI International, Menlo Park, Calif., October 1985.

NOTES

The anonymous account is inherently dangerous and should be avoided when possible.

The server must run as the super-user to create sockets with privileged port numbers. It maintains an effective user id of the logged in user, reverting to the super-user only when binding addresses to sockets. The possible security holes have been extensively scrutinized, but are possibly incomplete.

`/etc/ftpusers` contains a list of users who cannot access the system; the format of the file is one username per line.

NAME

`fumount` – forced unmount of an advertised resource

SYNOPSIS

`fumount` [-w *sec*] *resource*

DESCRIPTION

`fumount` unadvertises *resource* and disconnects remote access to the resource. The **-w** *sec* causes a delay of *sec* seconds prior to the execution of the disconnect.

When the forced unmount occurs, an administrative shell script is started on each remote computer that has the resource mounted (`/usr/bin/rfuadmin`). If a grace period of seconds is specified, `rfuadmin` is started with the `fuwarn` option. When the actual forced unmount is ready to occur, `rfuadmin` is started with the `fumount` option. See the `rfuadmin`(1M) manual page for information on the action taken in response to the forced unmount.

Only the super-user can execute this command.

ERRORS

If *resource* (1) does not physically reside on the local machine, (2) is an invalid resource name, (3) is not currently advertised and is not remotely mounted, or (4) the command is not run with super-user privilege, an error message will be sent to standard error.

SEE ALSO

`adv`(1M), `mount`(1M), `rfuadmin`(1M), `rfudaemon`(1M), `rmount`(1M), `unadv`(1M).

NAME

`fusage` – disk access profiler

SYNOPSIS

`fusage` [[*mount_point*] | [*advertised_resource*] | [*block_special_device*] [...]]

DESCRIPTION

When used with no options, `fusage` reports block I/O transfers, in kilobytes, to and from all locally mounted file systems and advertised Remote File Sharing resources on a per client basis. The count data are cumulative since the time of the mount. When used with an option, `fusage` reports on the named file system, advertised resource, or block special device.

The report includes one section for each file system and advertised resource and has one entry for each machine that has the directory remotely mounted, ordered by decreasing usage. Sections are ordered by device name; advertised resources that are not complete file systems will immediately follow the sections for the file systems they are in.

SEE ALSO

`adv`(1M), `mount`(1M), `df`(1M), `crash`(1M).

NAME

`gettable` – get DoD Internet format host table from a host

SYNOPSIS

`gettable` *host*

DESCRIPTION

`gettable` is a simple program used to obtain the DoD Internet host table from a hostname server. The indicated *host* is queried for the table. The table, if retrieved, is placed in the file `hosts.txt`.

`gettable` operates by opening a TCP connection to the port indicated in the service specification for hostname. A request is then made for all names and the resultant information is placed in the output file.

`gettable` is best used in conjunction with the `htable`(1M) program which converts the DoD Internet host table format to that used by the network library lookup routines.

SEE ALSO

`htable`(1M)

Harrenstien, Ken, Mary Stahl, and Elizabeth Feinler, *HOSTNAME Server*, RFC 953, Network Information Center, SRI International, Menlo Park, Calif., October 1985.

NOTES

Should allow requests for only part of the database.

NAME

`htable` – convert DoD Internet format host table

SYNOPSIS

`htable` *filename*

DESCRIPTION

`htable` converts a host table in the format specified by RFC 952 to the format used by the network library routines. Three files are created as a result of running `htable`: `hosts`, `networks`, and `gateways`. The `hosts` file is used by the `gethostent`(3N) routines in mapping host names to addresses. The `networks` file is used by the `getnetent`(3N) routines in mapping network names to numbers. The `gateways` file is used by the routing daemon in identifying passive Internet gateways; see `routed`(1M) for an explanation.

If any of the files `localhosts`, `localnetworks`, or `localgateways` are present in the current directory, the file's contents is prepended to the output file without interpretation. This allows sites to maintain local aliases and entries which are not normally present in the master database.

`htable` is best used in conjunction with the `gettable`(1M) program which retrieves the DoD Internet host table from a host.

FILES

`localhosts`
`localnetworks`
`localgateways`

SEE ALSO

`gethostent`(3N), `getnetent`(3N), `gettable`(1M), `routed`(1M).

Harrenstien, Ken, Mary Stahl, and Elizabeth Feinler, *DoD Internet Host Table Specification*, RFC 952, Network Information Center, SRI International, Menlo Park, Calif., October 1985.

NOTES

Does not properly calculate the `gateways` file.

NAME

idload – Remote File Sharing user and group mapping

SYNOPSIS

idload [-n] [-g *g_rules*] [-u *u_rules*] [directory]
idload -k

DESCRIPTION

idload is used on Remote File Sharing server machines to build translation tables for user and group ids. It takes your /etc/passwd and /etc/group files and produces translation tables for user and group ids from remote machines, according to the rules set down in the *u_rules* and *g_rules* files. If you are mapping by user and group name, you will need copies of remote /etc/passwd and /etc/group files. If no rules files are specified, remote user and group ids are mapped to MAXUID+1 (this is an id number that is one higher than the highest number you could assign on your system.)

By default, the remote password and group files are assumed to reside in /etc/dfs/rfs/auth.info/*domain*/*nodename*/[passwd | group]. The directory argument indicates that some directory structure other than /etc/dfs/rfs/auth.info contains the *domain*/*nodename* passwd and group files. (*nodename* is the name of the computer the files are from and *domain* is the domain that computer is a member of.)

You must run idload to put the mapping into place. Global mapping will take effect immediately for machines that have one of your resources currently mounted. Mapping for other specific machines will take effect when each machine mounts one of your resources.

–n	This is used to do a trial run of the id mapping. No translation table will be produced, however, a display of the mapping is output to the terminal (*stdout*).
–k	This is used to print the idmapping that is currently in use. (Specific mapping for remote machines will not be shown until that machine mounts one of your resources.)
–u *u_rules*	The *u_rules* file contains the rules for user id translation. The default rules file is /etc/dfs/rfs/auth.info/uid.rules.
–g *g_rules*	The *g_rules* file contains the rules for group id translation. The default rules file is /etc/dfs/rfs/auth.info/gid.rules.

Only the superuser can execute this command.

Rules

The rules files have two types of sections (both optional): global and host. There can be only one global section, though there can be one host section for each computer you want to map.

The global section describes the default conditions for translation for any machines that are not explicitly referenced in a host section. If the global section is missing, the default action is to map all remote user and group ids from undefined computers to MAXUID+1. The syntax of the first line of the global section is:

```
        global
```

A `host` section is used for each machine or group of machines that you want to map differently from the global definitions. The syntax of the first line of each `host` section is:

`host` *name* ...

where *name* is replaced by the full name of a computer (*domain.nodename*).

The format of a rules file is described below. (All lines are optional, but must appear in the order shown.)

`global`
`default` *local* | `transparent`
`exclude` *remote_id-remote_id* | *remote_id*
`map` *remote_id:local*

`host` *domain.nodename* [*domain.nodename...*]
`default` *local* | `transparent`
`exclude` *remote_id-remote_id* | *remote_id* | *remote_name*
`map` *remote:local* | *remote* | `all`

Each of these instruction types is described below.

The line

`default` *local* | `transparent`

defines the mode of mapping for remote users that are not specifically mapped in instructions in other lines. `transparent` means that each remote user and group id will have the same numeric value locally unless it appears in the **exclude** instruction. *local* can be replaced by a local user name or id to map all users into a particular local name or id number. If the default line is omitted, all users that are not specifically mapped are mapped into a "special guest" login id.

The line

`exclude` *remote_id-remote_id* | *remote_id* | *remote_name*

defines remote ids that will be excluded from the `default` mapping. The **exclude** instruction must precede any **map** instructions in a block. You can use a range of id numbers, a single id number, or a single name. (*remote_name* cannot be used in a `global` block.)

The line

`map` *remote:local* | *remote* | `all`

defines the local ids and names that remote ids and names will be mapped into. *remote* is either a remote id number or remote name; *local* is either a local id number or local name. Placing a colon between a *remote* and a *local* will give the value on the left the permissions of the value on the right. A single *remote* name or id will assign the user or group permissions of the same local name or id. `all` is a predefined alias for the set of all user and group ids found in the local `/etc/passwd` and `/etc/group` files. (You cannot map by remote name in `global` blocks.)

Note: `idload` will always output warning messages for `map all`, since password files always contain multiple administrative user names with the same id number. The first mapping attempt on the id number will succeed, each subsequent attempts will produce a warning.

Remote File Sharing doesn't need to be running to use `idload`.

EXIT STATUS

On successful completion, `idload` will produce one or more translation tables and return a successful exit status. If `idload` fails, the command will return an exit status of zero and not produce a translation table.

ERRORS

If (1) either rules file cannot be found or opened, (2) there are syntax errors in the rules file, (3) there are semantic errors in the rules file, (4) `host` password or group information could not be found, or (5) the command is not run with super-user privilege, an error message will be sent to standard error. Partial failures will cause a warning message to appear, though the process will continue.

FILES

`/etc/passwd`
`/etc/group`
`/etc/rfs/auth.info/`*domain*`/`*nodename*`/[user | group]`
`/etc/rfs/auth.info/uid.rules`
`/etc/rfs/auth.info/gid.rules`

SEE ALSO

`mount`(1M).
"Remote File Sharing" chapter of the *System Administrator's Guide* for detailed information on ID mapping.

NAME

`ifconfig` – configure network interface parameters

SYNOPSIS

`ifconfig` *interface* [*address_family*] [*address* [*dest_address*]] [*parameters*]
[**netmask** *mask*] [**broadcast** *address*] [**metric** *n*]

`ifconfig` *interface* [*protocol_family*]

DESCRIPTION

`ifconfig` is used to assign an address to a network interface and/or to configure network interface parameters. `ifconfig` must be used at boot time to define the network address of each interface present on a machine; it may also be used at a later time to redefine an interface's address or other operating parameters. Used without options, `ifconfig` displays the current configuration for a network interface. If a protocol family is specified, `ifconfig` will report only the details specific to that protocol family. Only the super-user may modify the configuration of a network interface.

The *interface* parameter is a string of the form name unit, for example **emd1**. The interface name **-a** is reserved, and causes the remainder of the arguments to be applied to each address of each interface in turn.

Since an interface may receive transmissions in differing protocols, each of which may require separate naming schemes, the parameters and addresses are interpreted according to the rules of some address family, specified by the *address_family* parameter. The address families currently supported are `ether` and `inet`. If no address family is specified, `inet` is assumed.

For the DARPA Internet family (`inet`), the address is either a host name present in the host name data base [see **hosts**(4)], or a DARPA Internet address expressed in the Internet standard dot notation. Typically, an Internet address specified in dot notation will consist of your system's network number and the machine's unique host number. A typical Internet address is `192.9.200.44`, where `192.9.200` is the network number and `44` is the machine's host number.

For the `ether` address family, the address is an Ethernet address represented as *x*:*x*:*x*:*x*:*x*:*x* where *x* is a hexadecimal number between 0 and ff. Only the super-user may use the `ether` address family.

If the *dest_address* parameter is supplied in addition to the *address* parameter, it specifies the address of the correspondent on the other end of a point to point link.

OPTIONS

The following *parameters* may be set with `ifconfig`:

up Mark an interface up. This may be used to enable an interface after an ifconfig down. It happens automatically when setting the first address on an interface. If the interface was reset when previously marked down, the hardware will be re-initialized.

down Mark an interface down. When an interface is marked down, the system will not attempt to transmit messages through that interface. If possible, the interface will be reset to disable reception as well. This action does not automatically disable routes using the interface.

`trailers`
(`inet` only) Enable the use of a trailer link level encapsulation when sending. If a network interface supports trailer encapsulation, the system will, when possible, encapsulate outgoing messages in a manner which minimizes the number of memory to memory copy operations performed by the receiver. This feature is machine-dependent, and therefore not recommended. On networks that support the Address Resolution Protocol [see `arp`(7)]; currently, only 10 Mb/s Ethernet), this flag indicates that the system should request that other systems use trailer encapsulation when sending to this host. Similarly, trailer encapsulations will be used when sending to other hosts that have made such requests.

`-trailers`
Disable the use of a trailer link level encapsulation.

`arp`
Enable the use of the Address Resolution Protocol in mapping between network level addresses and link level addresses (default). This is currently implemented for mapping between DARPA Internet addresses and 10Mb/s Ethernet addresses.

`-arp`
Disable the use of the Address Resolution Protocol.

`metric` *n*
Set the routing metric of the interface to *n*, default 0. The routing metric is used by the routing protocol [`routed`(1M)]. Higher metrics have the effect of making a route less favorable; metrics are counted as addition hops to the destination network or host.

`netmask` *mask*
(`inet` only) Specify how much of the address to reserve for subdividing networks into sub-networks. The mask includes the network part of the local address and the subnet part, which is taken from the host field of the address. The mask can be specified as a single hexadecimal number with a leading 0x, with a dot-notation Internet address, or with a pseudo-network name listed in the network table `networks`(4). The mask contains 1's for the bit positions in the 32-bit address which are to be used for the network and subnet parts, and 0's for the host part. The mask should contain at least the standard network portion, and the subnet field should be contiguous with the network portion. If a + (plus sign) is given for the **`netmask`** value, then the network number is looked up in the **`/etc/netmasks`** file.

`broadcast` *address*
(`inet` only) Specify the address to use to represent broadcasts to the network. The default broadcast address is the address with a host part of all 0's. A + (plus sign) given for the broadcast value causes the broadcast address to be reset to a default appropriate for the (possibly new) Internet address and netmask. Note that the arguments of `ifconfig` are interpreted left to right, and therefore

```
ifconfig -a netmask + broadcast +
```

and

```
ifconfig -a broadcast + netmask +
```

may result in different values being assigned for the interfaces' broadcast addresses.

EXAMPLES

If your workstation is not attached to an Ethernet, the `emdl` interface should be marked down as follows:

```
ifconfig emdl down
```

To print out the addressing information for each interface, use

```
ifconfig -a
```

To reset each interface's broadcast address after the netmasks have been correctly set, use

```
ifconfig -a broadcast +
```

FILES

`/dev/nit`
`/etc/netmasks`

SEE ALSO

`netstat`(1M), `netmasks`(4).

DIAGNOSTICS

Messages indicating the specified interface does not exist, the requested address is unknown, or the user is not privileged and tried to alter an interface's configuration.

NAME

inetd – Internet services daemon

SYNOPSIS

inetd [-d] [-s] [*configuration-file*]

DESCRIPTION

inetd, the Internet services daemon, is normally run at boot time by the Service Access Facility (SAF). When started, inetd reads its configuration information from *configuration-file*, the default being /etc/inetd.conf. See inetd.conf(4) for more information on the format of this file. It listens for connections on the Internet addresses of the services that its configuration file specifies. When a connection is found, it invokes the server daemon specified by that configuration file for the service requested. Once a server process exits, inetd continues to listen on the socket.

The -s option allows you to run inetd "stand-alone," outside the Service Access Facility (SAF).

Rather than having several daemon processes with sparsely distributed requests each running concurrently, inetd reduces the load on the system by invoking Internet servers only as they are needed.

inetd itself provides a number of simple TCP-based services. These include echo, discard, chargen (character generator), daytime (human readable time), and time (machine readable time, in the form of the number of seconds since midnight, January 1, 1900). For details of these services, consult the appropriate RFC, as listed below, from the Network Information Center.

inetd rereads its configuration file whenever it receives a hangup signal, SIGHUP. New services can be activated, and existing services deleted or modified in between whenever the file is reread.

SEE ALSO

comsat(1M), ftpd(1M), rexecd(1M), rlogind(1M), rshd(1M), telnetd(1M), tftpd(1M), inetd.conf(4).

Postel, Jon, "Echo Protocol," RFC 862, Network Information Center, SRI International, Menlo Park, Calif., May 1983.

Postel, Jon, "Discard Protocol," RFC 863, Network Information Center, SRI International, Menlo Park, Calif., May 1983.

Postel, Jon, "Character Generater Protocol," RFC 864, Network Information Center,
SRI International, Menlo Park, Calif., May 1983.

Postel, Jon, "Daytime Protocol," RFC 867, Network Information Center, SRI International, Menlo Park, Calif., May 1983.

Postel, Jon, and Ken Harrenstien, "Time Protocol," RFC 868, Network Information Center, SRI International, Menlo Park, Calif., May 1983.

NAME

keyserv – server for storing public and private keys

SYNOPSIS

keyserv [-n]

DESCRIPTION

keyserv is a daemon that is used for storing the private encryption keys of each user logged into the system. These encryption keys are used for accessing secure network services such as secure NFS.

Normally, root's key is read from the file /etc/.rootkey when the daemon is started. This is useful during power-fail reboots when no one is around to type a password.

When the -n option is used, root's key is not read from /etc/.rootkey. Instead, **keyserv** prompts the user for the password to decrypt root's key stored in the **publickey**(4) database and then stores the decrypted key in /etc/.rootkey for future use. This option is useful if the /etc/.rootkey file ever gets out of date or corrupted.

FILES

/etc/.rootkey

SEE ALSO

publickey(4).

NAME

`lockd` – network lock daemon

SYNOPSIS

`/usr/lib/nfs/lockd` [`-t` *timeout*] [`-g` *graceperiod*]

DESCRIPTION

`lockd` processes lock requests that are either sent locally by the kernel or remotely by another lock daemon. `lockd` forwards lock requests for remote data to the server site's lock daemon through RPC/XDR. `lockd` then requests the status monitor daemon, `statd`(1M), for monitor service. The reply to the lock request will not be sent to the kernel until the status daemon and the server site's lock daemon have replied.

If either the status monitor or server site's lock daemon is unavailable, the reply to a lock request for remote data is delayed until all daemons become available.

When a server recovers, it waits for a grace period for all client-site lock daemons to submit reclaim requests. Client-site lock daemons, on the other hand, are notified by the status monitor daemon of the server recovery and promptly resubmit previously granted lock requests. If a lock daemon fails to secure a previously granted lock at the server site, the it sends SIGLOST to a process.

OPTIONS

`-t` *timeout*
: Use *timeout* seconds as the interval instead of the default value (15 seconds) to retransmit lock request to the remote server.

`-g` *graceperiod*
: Use *graceperiod* seconds as the grace period duration instead of the default value (45 seconds).

SEE ALSO

`statd`(1M), `fcntl`(2), `signal`(2), `lockf`(3C).

NAME

mount – mount remote NFS resources

SYNOPSIS

mount [-F nfs] [-r] [-o *specific_options*] [*resource mountpoint*]

DESCRIPTION

The mount command attaches a named *resource* to the file system hierarchy at the pathname location *mountpoint*, which must already exist. If *mountpoint* has any contents prior to the mount operation, the contents remain hidden until the *resource* is once again unmounted.

If the resource is listed in the vfstab file, the command line can specify either *resource* or *mountpoint*, and mount will consult vfstab for more information. If the -F option is omitted, mount will take the file system type from vfstab.

mount maintains a table of mounted file systems in /etc/mnttab, described in mnttab(4).

The following options are available to the mount command:

-r Mount the specified file system read-only.

-o *specific_options*
Specify file system specific options in a comma-separated list of words from the list below.

rw | ro	*resource* is mounted read-write or read-only. The default is rw.
suid | nosuid	Setuid execution allowed or disallowed. The default is suid.
remount	If a file system is mounted read-only, remounts the file system read-write.
bg | fg	If the first attempt fails, retry in the background, or, in the foreground. The default is fg.
retry=*n*	The number of times to retry the mount operation. The default is 10000.
port=*n*	The server IP port number. The default is NFS_PORT.
grpid	Create a file with its GID set to the effective GID of the calling process. This behavior may be overridden on a per-directory basis by setting the set-GID bit of the parent directory; in this case, the GID is set to the GID of the parent directory [see open(2) and mkdir(2)]. Files created on file systems that are *not* mounted with the grpid option will obey BSD semantics; that is, the GID is unconditionally inherited from that of the parent directory.
rsize=*n*	Set the read buffer size to *n* bytes.
wsize=*n*	Set the write buffer size to *n* bytes.
timeo=*n*	Set the NFS timeout to *n* tenths of a second.
retrans=*n*	Set the number of NFS retransmissions to *n*.
soft | hard	Return an error if the server does not respond, or continue the retry request until the server responds.

`intr`	Allow keyboard interrupts to kill a process that is hung while waiting for a response on a hard-mounted file system.
`secure`	Use a more secure protocol for NFS transactions.
`noac`	Suppress attribute caching.
`acregmin=`*n*	Hold cached attributes for at least *n* seconds after file modification.
`acregmax=`*n*	Hold cached attributes for no more than *n* seconds after file modification.
`acdirmin=`*n*	Hold cached attributes for at least *n* seconds after directory update.
`acdirmax=`*n*	Hold cached attributes for no more than *n* seconds after directory update.
`actimeo=`*n*	Set *min* and *max* times for regular files and directories to *n* seconds.

NFS FILE SYSTEMS

Background vs. Foreground

File systems mounted with the `bg` option indicate that `mount` is to retry in the background if the server's mount daemon [`mountd`(1M)] does not respond. `mount` retries the request up to the count specified in the `retry=`*n* option. Once the file system is mounted, each NFS request made in the kernel waits `timeo=`*n* tenths of a second for a response. If no response arrives, the time-out is multiplied by 2 and the request is retransmitted. When the number of retransmissions has reached the number specified in the `retrans=`*n* option, a file system mounted with the `soft` option returns an error on the request; one mounted with the `hard` option prints a warning message and continues to retry the request.

Read-Write vs. Read-Only

File systems that are mounted `rw` (read-write) should use the `hard` option.

Secure File Systems

The `secure` option must be given if the server requires secure mounting for the file system.

File Attributes

The attribute cache retains file attributes on the client. Attributes for a file are assigned a time to be flushed. If the file is modified before the flush time, then the flush time is extended by the time since the last modification (under the assumption that files that changed recently are likely to change soon). There is a minimum and maximum flush time extension for regular files and for directories. Setting `actimeo=`*n* extends flush time by *n* seconds for both regular files and directories.

EXAMPLES

To mount a remote file system: `mount -F nfs serv:/usr/src /usr/src`
To hard mount a remote file system: `mount -o hard serv:/usr/src /usr/src`

FILES

`/etc/mnttab`	table of mounted file systems
`/etc/dfs/fstypes`	default distributed file system type
`/etc/vfstab`	table of automatically mounted resources

SEE ALSO

`mountall`(1M), `mount`(2), `umount`(2), `mnttab`(4).

NOTES

If the directory on which a file system is to be mounted is a symbolic link, the file system is mounted on *the directory to which the symbolic link refers*, rather than being mounted on top of the symbolic link itself.

NAME

mount – mount remote resources

SYNOPSIS

mount [**–F** rfs] [**–o** nocaching][,ro|rw] [,suid|nosuid] [**–cr**] *resource directory*

DESCRIPTION

The **mount** command makes a remote *resource* available to users from the mount point *directory*. The command adds an entry to the table of mounted devices, `/etc/mnttab`.

If multiple transport providers are installed and administrators attempt to mount a resource over them, the transport providers should be specified as network IDs in the `/etc/netconfig` file. The NETPATH environment variable can be used to specify the sequence of transport providers **mount** will use to attempt a connection to a server machine (**NETPATH=tcp:starlan**). If only one transport provider is installed and `/etc/netconfig` has not been set up, all resources will be mounted over this transport provider by default.

The following options are available:

–o *suboption*

nocaching	Disable client caching.
[rw|ro]	*resource* is to be mounted read/write or read-only. The default is read/write.
[suid|nosuid]	set-uid bits are to be obeyed or ignored, respectively, on execution. The default is nosuid.

–c	Disable client caching. This is the same as **–o** nocaching.
–r	*resource* is to be mounted read-only. If the *resource* is write-protected, this flag, or the **–o** `ro` flag, must be used.

FILES

`/etc/mnttab`
`/etc/netconfig`
`/etc/vfstab`

SEE ALSO

umount(1M), **share**(1M), **fuser**(1M), **unshare**(1M), **dfshares**(1M), **dfmounts**(1M), **netconfig**(4), **mnttab**(4), **vfstab**(1M)

NAME

mountd – NFS mount request server

SYNOPSIS

mountd [-n]

DESCRIPTION

mountd is an RPC server that answers file system mount requests. It reads the file /etc/dfs/sharetab, described in sharetab(4), to determine which file systems are available for mounting by which machines. It also provides information as to what file systems are mounted by which clients. This information can be printed using the dfmounts(1M) command.

The mountd daemon is automatically invoked in run level 3.

With the -n option, mountd does not check that the clients are root users. Though this option makes things slightly less secure, it does allow older versions (pre-3.0) of client NFS to work.

FILES

/etc/dfs/sharetab

SEE ALSO

dfmounts(1M), sharetab(4).

NAME

named, in.named – Internet domain name server

SYNOPSIS

in.named [-d *level*] [-p *port*] [[-b] *bootfile*]

DESCRIPTION

named is the Internet domain name server. It is used by hosts on the Internet to provide access to the Internet distributed naming database. See RFC 1034 and RFC 1035 for more details. With no arguments named reads /etc/named.boot for any initial data, and listens for queries on a privileged port.

The following options are available:

-d *level*	Print debugging information. *level* is a number indicating the level of messages printed.
-p *port*	Use a different *port* number.
-b *bootfile*	Use *bootfile* rather than /etc/named.boot.

EXAMPLE

```
;
;     boot file for name server
;
; type          domain          source file or host
;
domain          berkeley.edu
primary         berkeley.edu    named.db
secondary  cc.berkeley.edu 10.2.0.78 128.32.0.10
cache                     named.ca
```

The domain line specifies that berkeley.edu is the domain of the given server.

The primary line states that the file named.db contains authoritative data for berkeley.edu. The file named.db contains data in the master file format, described in RFC 1035, except that all domain names are relative to the origin; in this case, berkeley.edu (see below for a more detailed description).

The secondary line specifies that all authoritative data under cc.berkeley.edu is to be transferred from the name server at 10.2.0.78. If the transfer fails it will try 128.32.0.10, and continue for up to 10 tries at that address. The secondary copy is also authoritative for the domain.

The cache line specifies that data in named.ca is to be placed in the cache (typically such data as the locations of root domain servers). The file named.ca is in the same format as named.db.

The master file consists of entries of the form:

> $INCLUDE < *filename* >
> $ORIGIN < *domain* >
> < *domain* > < *opt_ttl* > < *opt_class* > < *type* > < *resource_record_data* >

where *domain* is . for the root, @ for the current origin, or a standard domain name. If *domain* is a standard domain name that does not end with ., the current origin is appended to the domain. Domain names ending with . are unmodified.

The *opt_ttl* field is an optional integer number for the time-to-live field. It defaults to zero.

The *opt_class* field is currently one token, `IN` for the Internet.

The *type* field is one of the following tokens; the data expected in the *resource_record_data* field is in parentheses.

`A`	A host address (dotted quad).
`NS`	An authoritative name server (domain).
`MX`	A mail exchanger (domain).
`CNAME`	The canonical name for an alias (domain).
`SOA`	Marks the start of a zone of authority (5 numbers). See RFC 1035.
`MB`	A mailbox domain name (domain).
`MG`	A mail group member (domain).
`MR`	A mail rename domain name (domain).
`NULL`	A null resource record (no format or data).
`WKS`	A well know service description (not implemented yet).
`PTR`	A domain name pointer (domain).
`HINFO`	Host information (cpu_type OS_type).
`MINFO`	Mailbox or mail list information (request_domain error_domain).

FILES

`/etc/named.boot`	name server configuration boot file
`/etc/named.pid`	the process ID
`/var/tmp/named.run`	debug output
`/var/tmp/named_dump.db`	dump of the name servers database

SEE ALSO

`kill`(1), `signal`(3), `resolver`(3N), `resolve.conf`(4).

Mockapetris, Paul, *Domain Names - Concepts and Facilities*, RFC 1034, Network Information Center, SRI International, Menlo Park, Calif., November 1987.

Mockapetris, Paul, *Domain Names - Implementation and Specification*, RFC 1035, Network Information Center, SRI International, Menlo Park, Calif., November 1987.

Mockapetris, Paul, *Domain System Changes and Observations*, RFC 973, Network Information Center, SRI International, Menlo Park, Calif., January 1986.

Partridge, Craig, *Mail Routing and the Domain System*, RFC 974, Network Information Center, SRI International, Menlo Park, Calif., January 1986.

NOTES

The following signals have the specified effect when sent to the server process using the `kill`(1) command.

`SIGHUP`	Reads `/etc/named.boot` and reloads database.
`SIGINT`	Dumps the current database and cache to `/var/tmp/named_dump.db`.
`SIGUSR1`	Turns on debugging; each subsequent `SIGUSR1` increments debug level.
`SIGUSR2`	Turns off debugging completely.

NAME

netstat – show network status

SYNOPSIS

`netstat` [`-aAn`] [`-f` *addr_family*] [`system`] [`core`]

`netstat` [`-n`] [`-s`] [`-i` | `-r`] [`-f` *addr_family*] [`system`] [`core`]

`netstat` [`-n`] [`-I` *interface*] *interval* [`system`] [`core`]

DESCRIPTION

`netstat` displays the contents of various network-related data structures in various formats, depending on the options you select.

The first form of the command displays a list of active sockets for each protocol. The second form selects one from among various other network data structures. The third form displays running statistics of packet traffic on configured network interfaces; the *interval* argument indicates the number of seconds in which to gather statistics between displays.

The default value for the `system` argument is `/unix`; for *core*, the default is `/dev/kmem`.

The following options are available:

`-a` Show the state of all sockets; normally sockets used by server processes are not shown.

`-A` Show the address of any protocol control blocks associated with sockets; used for debugging.

`-i` Show the state of interfaces that have been auto-configured. Interfaces that are statically configured into a system, but not located at boot time, are not shown.

`-n` Show network addresses as numbers. `netstat` normally displays addresses as symbols. This option may be used with any of the display formats.

`-r` Show the routing tables. When used with the `-s` option, show routing statistics instead.

`-s` Show per-protocol statistics. When used with the `-r` option, show routing statistics.

`-f` *addr_family*
Limit statistics or address control block reports to those of the specified *addr_family*, which can be one of:

`inet`	For the AF_INET address family, or
`unix`	For the AF_UNIX family.

`-I` *interface*
Highlight information about the indicated *interface* in a separate column; the default (for the third form of the command) is the interface with the most traffic since the system was last rebooted. *interface* can be any valid interface listed in the system configuration file, such as `emd1` or `lo0`.

DISPLAYS

Active Sockets (First Form)

The display for each active socket shows the local and remote address, the send and receive queue sizes (in bytes), the protocol, and the internal state of the protocol.

The symbolic format normally used to display socket addresses is either:

hostname.port

when the name of the host is specified, or:

network.port

if a socket address specifies a network but no specific host. Each **hostname** and *network* is shown according to its entry in the /etc/hosts or the /etc/networks file, as appropriate.

If the network or hostname for an address is not known (or if the -n option is specified), the numerical network address is shown. Unspecified, or wildcard, addresses and ports appear as *. For more information regarding the Internet naming conventions, refer to inet(7).

TCP Sockets

The possible state values for TCP sockets are as follows:

CLOSED	Closed. The socket is not being used.
LISTEN	Listening for incoming connections.
SYN_SENT	Actively trying to establish connection.
SYN_RECEIVED	Initial synchronization of the connection under way.
ESTABLISHED	Connection has been established.
CLOSE_WAIT	Remote shut down; waiting for the socket to close.
FIN_WAIT_1	Socket closed; shutting down connection.
CLOSING	Closed, then remote shutdown; awaiting acknowledgement.
LAST_ACK	Remote shut down, then closed; awaiting acknowledgement.
FIN_WAIT_2	Socket closed; waiting for shutdown from remote.
TIME_WAIT	Wait after close for remote shutdown retransmission.

Network Data Structures (Second Form)

The form of the display depends upon which of the -i or -r options you select. If you specify more than one of these options, netstat selects one in the order listed here.

Routing Table Display

The routing table display lists the available routes and the status of each. Each route consists of a destination host or network, and a gateway to use in forwarding packets. The *flags* column shows the status of the route (U if up), whether the route is to a gateway (G), and whether the route was created dynamically by a redirect (D).

Direct routes are created for each interface attached to the local host; the gateway field for such entries shows the address of the outgoing interface.

The `refcnt` column gives the current number of active uses per route. Connection-oriented protocols normally hold on to a single route for the duration of a connection, whereas connectionless protocols obtain a route while sending to the same destination.

The **use** column displays the number of packets sent per route.

The *interface* entry indicates the network interface utilized for the route.

Cumulative Traffic Statistics (Third Form)

When the *interval* argument is given, `netstat` displays a table of cumulative statistics regarding packets transferred, errors and collisions, the network addresses for the interface, and the maximum transmission unit (mtu). The first line of data displayed, and every 24th line thereafter, contains cumulative statistics from the time the system was last rebooted. Each subsequent line shows incremental statistics for the *interval* (specified on the command line) since the previous display.

SEE ALSO

`iostat`(1M), `trpt`(1M), `vmstat`(1M), `hosts`(4), `networks`(4), `protocols`(4), `services`(4).

NOTES

The notion of errors is ill-defined.

The kernel's tables can change while `netstat` is examining them, creating incorrect or partial displays.

NAME

newkey – create a new key in the publickey database

SYNOPSIS

newkey -h *hostname*

newkey -u *username*

DESCRIPTION

The **newkey** command is normally run by the network administrator on the machine that contains the `publickey`(4) database, to establish public keys for users and privileged users on the network. These keys are needed when using secure RPC or secure NFS.

newkey will prompt for a password for the given *username* or *hostname* and then create a new public/secret key pair for the user or host in `/etc/publickey`, encrypted with the given password.

The following options are available:

-h *hostname*	Create a new public/secret key pair for the privileged user at the given *hostname*. Prompts for a password for the given *hostname*.
-u *username*	Create a new public/secret key pair for the given *username*. Prompts for a password for the given *username*.

SEE ALSO

`chkey`(1), `keylogin`(1), `keylogout`(1), `keyserv`(1M), `publickey`(4)

NAME

`nfsd` – NFS daemon

SYNOPSIS

`nfsd` [`-a`] [`-p` *protocol*] [`-t` *device*] [*nservers*]

DESCRIPTION

`nfsd` starts the daemons that handle client file system requests.

The following options are recognized:

`-a`	start `nfsd`'s over all available connectionless transports
`-p` *protocol*	start `nfsd`'s over the specified protocol
`-t` *device*	start `nfsd`'s for the transport specified by the given device
nservers	the number of file system request daemons to start.

nservers should be based on the load expected on this server. Four is the usual number of *nservers*.

The `nfsd` daemons are automatically invoked in run level 3.

FILES

`.nfs`*XXX*	client machine pointer to an open-but-unlinked file

SEE ALSO

`biod`(1M), `mountd`(1M), `sharetab`(4).

NAME

`nslookup` – query name servers interactively

SYNOPSIS

`nslookup` [`-l`] [*address*]

DESCRIPTION

`nslookup` is an interactive program to query ARPA Internet domain name servers. The user can contact servers to request information about a specific host or print a list of hosts in the domain.

OPTIONS

`-l`	Use the local host's name server instead of the servers in `/etc/resolve.conf`. (If `/etc/resolve.conf` does not exist or does not contain server information, the `-l` option does not have any effect).
address	Use the name server on the host machine with the given Internet address.

USAGE

Overview

The Internet domain name-space is tree-structured, with four top-level domains at present:

COM	commercial establishments
EDU	educational institutions
GOV	government agencies
MIL	MILNET hosts

If you are looking for a specific host, you need to know something about the host's organization in order to determine the top-level domain it belongs to. For instance, if you want to find the Internet address of a machine at UCLA , do the following:

- Connect with the root server using the `root` command. The root server of the name space has knowledge of the top-level domains.
- Since UCLA is a university, its domain name is `ucla.edu`. Connect with a server for the `ucla.edu` domain with the command `server ucla.edu`. The response will print the names of hosts that act as servers for that domain. Note: the root server does not have information about `ucla.edu`, but knows the names and addresses of hosts that do. Once located by the root server, all future queries will be sent to the UCLA name server.
- To request information about a particular host in the domain (for instance, `locus`), just type the host name. To request a listing of hosts in the UCLA domain, use the `ls` command. The `ls` command requires a domain name (in this case, `ucla.edu`) as an argument.

If you are connected with a name server that handles more than one domain, all lookups for host names must be fully specified with its domain. For instance, the domain `harvard.edu` is served by `seismo.css.gov`, which also services the `css.gov` and `cornell.edu` domains. A lookup request for the host **`aiken`** in the **`harvard.edu`** domain must be specified as **`aiken.harvard.edu`**. However, the

```
set domain=name
```

and

```
set defname
```

commands can be used to automatically append a domain name to each request.

After a successful lookup of a host, use the `finger` command to see who is on the system, or to finger a specific person. To get other information about the host, use the

```
set querytype=value
```

command to change the type of information desired and request another lookup. (`finger` requires the type to be A.)

Commands

To exit, type Ctrl-D (EOF). The command line length must be less than 80 characters. An unrecognized command will be interpreted as a host name.

host [*server*]
: Look up information for *host* using the current default server or using *server* if it is specified.

`server` *domain*
`lserver` *domain*
: Change the default server to *domain*. `lserver` uses the initial server to look up information about *domain* while **`server`** uses the current default server. If an authoritative answer can't be found, the names of servers that might have the answer are returned.

`root` Changes the default server to the server for the root of the domain name space. Currently, the host `sri-nic.arpa` is used; this command is a synonym for `lserver sri-nic.arpa`.) The name of the root server can be changed with the `set root` command.

`finger` [*name*]
: Connect with the finger server on the current host, which is defined by a previous successful lookup for a host's address information (see the `set` *querytype* `=A` command). As with the shell, output can be redirected to a named file using > and >>.

`ls` [`-ah`]
: List the information available for *domain*. The default output contains host names and their Internet addresses. The `-a` option lists aliases of hosts in the domain. The `-h` option lists CPU and operating system information for the domain. As with the shell, output can be redirected to a named file using > and >>. When output is directed to a file, hash marks are printed for every 50 records received from the server.

`view` *filename*
: Sort and list the output of the `ls` command with `more`(1).

`help`
`?` Print a brief summary of commands.

`set` *keyword* [= *value*] This command is used to change state information that affects the lookups. Valid keywords are:

`all` Prints the current values of the various options to `set`. Information about the current default server and host is also printed.

[no] deb[ug]
Turn debugging mode on. A lot more information is printed about the packet sent to the server and the resulting answer. The default is nodebug.

[no] def[*name*]
Append the default domain name to every lookup. The default is **nodefname**.

do [`main`] = *filename*
Change the default domain name to *filename*. The default domain name is appended to all lookup requests if **defname** option has been set. The default is the value in `/etc/resolve.conf`.

q [querytype] = *value*
Change the type of information returned from a query to one of:

`A`	The host's Internet address (the default).
`CNAME`	The canonical name for an alias.
`HINFO`	The host CPU and operating system type.
`MD`	The mail destination.
`MX`	The mail exchanger.
`MB`	The mailbox domain name.
`MG`	The mail group member.
`MINFO`	The mailbox or mail list information.

(Other types specified in the RFC883 document are valid, but are not very useful.)

[no] `recurse`
Tell the name server to query other servers if it does not have the information. The default is `recurse`.

`ret` [`ry`] = *count*
Set the number of times to retry a request before giving up to *count*. When a reply to a request is not received within a certain amount of time (changed with `set timeout`), the request is resent. The default is *count* is 2.

`ro` [ot] = *host*
Change the name of the root server to *host*. This affects the `root` command. The default root server is `sri-nic.arpa`.

`t [ timeout ] =` *interval*
: Change the time-out for a reply to *interval* seconds. The default *interval* is 10 seconds.

`[ no ] v[ c ]`
: Always use a virtual circuit when sending requests to the server. The default is novc.

FILES

`/etc/resolve.conf` initial domain name and name server addresses.

SEE ALSO

named(1M), `resolver`(3N), `resolve.conf`(4), RFC 882, RFC 883.

DIAGNOSTICS

If the lookup request was not successful, an error message is printed. Possible errors are:

`Time-out`
: The server did not respond to a request after a certain amount of time (changed with `set timeout =`*value*) and a certain number of retries (changed with `set retry =`*value*).

`No information`
: Depending on the query type set with the `set querytype` command, no information about the host was available, though the host name is valid.

`Non-existent domain`
: The host or domain name does not exist.

`Connection refused`
`Network is unreachable`
: The connection to the name or finger server could not be made at the current time. This error commonly occurs with `finger` requests.

`Server failure`
: The name server found an internal inconsistency in its database and could not return a valid answer.

`Refused`
: The name server refused to service the request.

The following error should not occur and it indicates a bug in the program.

`Format error`
: The name server found that the request packet was not in the proper format.

NAME

nsquery – Remote File Sharing name server query

SYNOPSIS

nsquery [-h] [*name*]

DESCRIPTION

nsquery provides information about resources available to the host from both the local domain and from other domains. All resources are reported, regardless of whether the host is authorized to access them. When used with no options, **nsquery** identifies all resources in the domain that have been advertised as sharable. A report on selected resources can be obtained by specifying *name*, where *name* is:

nodename	The report will include only those resources available from *nodename*.
domain.	The report will include only those resources available from *domain*.
domain.nodename	The report will include only those resources available from *domain.nodename*.

When the name does not include the delimiter ".", it will be interpreted as a *nodename* within the local domain. If the name ends with a delimiter ".", it will be interpreted as a domain name.

The information contained in the report on each resource includes its advertised name (*domain.resource*), the read/write permissions, the server (*nodename.domain*) that advertised the resource, and a brief textual description.

When -h is used, the header is not printed.

A remote domain must be listed in your `rfmaster` file in order to query that domain.

EXIT STATUS

If no entries are found when **nsquery** is executed, the report header is printed.

SEE ALSO

adv(1M), **unadv**(1M), `rfmaster`(4).

NOTES

If your host cannot contact the domain name server, an error message will be sent to standard error.

NAME

`ping` – send ICMP `ECHO_REQUEST` packets to network hosts

SYNOPSIS

`ping` *host* [*timeout*]

`/usr/sbin/ping` [`-s`] [`-lrRv`] *host* [*packetsize*] [*count*]

DESCRIPTION

`ping` utilizes the ICMP protocol's `ECHO_REQUEST` datagram to elicit an ICMP `ECHO_RESPONSE` from the specified *host* or network gateway. If *host* responds, `ping` will print *host* `is alive` on the standard output and exit. Otherwise after *timeout* seconds, it will write `no answer from` *host*. The default value of *timeout* is 20 seconds.

When the `-s` flag is specified, `ping` sends one datagram per second, and prints one line of output for every `ECHO_RESPONSE` that it receives. No output is produced if there is no response. In this second form, `ping` computes round trip times and packet loss statistics; it displays a summary of this information upon termination or timeout. The default datagram packet size is 64 bytes, or you can specify a size with the *packetsize* command-line argument. If an optional *count* is given, `ping` sends only that number of requests.

When using `ping` for fault isolation, first ping the local host to verify that the local network interface is running.

OPTIONS

`-l` Loose source route. Use this option in the IP header to send the packet to the given host and back again. Usually specified with the `-R` option.

`-r` Bypass the normal routing tables and send directly to a host on an attached network. If the host is not on a directly-attached network, an error is returned. This option can be used to `ping` a local host through an interface that has been dropped by the router daemon [see `routed`(1M)].

`-R` Record route. Sets the IP record route option, which will store the route of the packet inside the IP header. The contents of the record route will only be printed if the `-v` option is given, and only be set on return packets if the target host preserves the record route option across echos, or the `-l` option is given.

`-v` Verbose output. List any ICMP packets, other than `ECHO_RESPONSE`, that are received.

SEE ALSO

`ifconfig`(1M), `netstat`(1M), `rpcinfo`(1M), `icmp`(7).

NAME

rarpd – DARPA Reverse Address Resolution Protocol server

SYNOPSIS

rarpd *interface* [*hostname*]

/usr/sbin/rarpd -a

DESCRIPTION

rarpd starts a daemon that responds to Reverse Address Resolution Protocol (RARP) requests. The daemon forks a copy of itself that runs in background. It must be run as root.

RARP is used by machines at boot time to discover their Internet Protocol (IP) address. The booting machine provides its Ethernet Address in a RARP request message. Using the **ethers** and **hosts** databases, rarpd maps this Ethernet Address into the corresponding IP address which it returns to the booting machine in an RARP reply message. The booting machine must be listed in both databases for rarpd to locate its IP address. rarpd issues no reply when it fails to locate an IP address.

In the first synopsis, the *interface* parameter names the network interface upon which rarpd is to listen for requests. The *interface* parameter takes the "name unit" form used by ifconfig(1M). The second argument, *hostname,* is used to obtain the IP address of that interface. An IP address in "decimal dot" notation may be used for *hostname*. If *hostname* is omitted, the address of the interface will be obtained from the kernel. When the first form of the command is used, rarpd must be run separately for each interface on which RARP service is to be supported. A machine that is a router may invoke rarpd multiple times, for example:

```
/usr/sbin/rarpd emd1 host
/usr/sbin/rarpd emd2 host-backbone
```

In the second synopsis, rarpd locates all of the network interfaces present on the system and starts a daemon process for each one that supports RARP.

FILES

/etc/ethers
/etc/hosts

SEE ALSO

ifconfig(1M), ethers(4), hosts(4), netconfig(4), boot(8).

Finlayson, Ross, Timothy Mann, Jeffrey Mogul, and Marvin Theimer, *A Reverse Address Resolution Protocol*, RFC 903, Network Information Center, SRI International, Menlo Park, Calif., June 1984.

NAME

`rdate` – set system date from a remote host

SYNOPSIS

`rdate` *hostname*

DESCRIPTION

`rdate` sets the local date and time from the *hostname* given as an argument. You must be super-user on the local system. Typically `rdate` can be inserted as part of a startup script.

NAME

`rexecd` – remote execution server

SYNOPSIS

`in.rexecd` *host.port*

DESCRIPTION

`rexecd` is the server for the `rexec`(3N) routine. The server provides remote execution facilities with authentication based on user names and encrypted passwords. It is invoked automatically as needed by `inetd`(1M), and then executes the following protocol:

1) The server reads characters from the socket up to a null (\0) byte. The resultant string is interpreted as an ASCII number, base 10.

2) If the number received in step 1 is non-zero, it is interpreted as the port number of a secondary stream to be used for the `stderr`. A second connection is then created to the specified port on the client's machine.

3) A null terminated user name of at most 16 characters is retrieved on the initial socket.

4) A null terminated, encrypted, password of at most 16 characters is retrieved on the initial socket.

5) A null terminated command to be passed to a shell is retrieved on the initial socket. The length of the command is limited by the upper bound on the size of the system's argument list.

6) `rexecd` then validates the user as is done at login time and, if the authentication was successful, changes to the user's home directory, and establishes the user and group protections of the user. If any of these steps fail the connection is aborted with a diagnostic message returned.

7) A null byte is returned on the connection associated with the `stderr` and the command line is passed to the normal login shell of the user. The shell inherits the network connections established by `rexecd`.

SEE ALSO

`inetd`(1M)

DIAGNOSTICS

All diagnostic messages are returned on the connection associated with the `stderr`, after which any network connections are closed. An error is indicated by a leading byte with a value of 1 (0 is returned in step 7 above upon successful completion of all the steps prior to the command execution).

`username too long`
: The name is longer than 16 characters.

`password too long`
: The password is longer than 16 characters.

`command too long`
: The command line passed exceeds the size of the argument list (as configured into the system).

`Login incorrect.`
: No password file entry for the user name existed.

`Password incorrect.`
: The wrong password was supplied.

`No remote directory.`
: The `chdir` command to the home directory failed.

`Try again.`
: A `fork` by the server failed.

`/usr/bin/sh: ...`
: The user's login shell could not be started.

NOTES

Indicating `Login incorrect` as opposed to `Password incorrect` is a security breach which allows people to probe a system for users with null passwords.

A facility to allow all data exchanges to be encrypted should be present.

NAME

`rfadmin` – Remote File Sharing domain administration

SYNOPSIS

`rfadmin`

`rfadmin -a` *hostname*

`rfadmin -r` *hostname*

`rfadmin -p` [`-t` *transport1,transport2,...*]

`rfadmin -q`

`rfadmin -o` *option*

DESCRIPTION

`rfadmin` is used to add and remove hosts, and their associated authentication information, from a *domain*/`passwd` file on a Remote File Sharing primary domain name server. It is also used to transfer domain name server responsibilities from one machine to another. Used with no options, `rfadmin` returns the *hostname* of the current domain name server for the local domain on each of the transport providers that span the domain.

`rfadmin` can only be used to modify domain files on the primary domain name server (`-a` and `-r` options). If domain name server responsibilities are temporarily passed to a secondary domain name server, that computer can use the `-p` option to pass domain name server responsibility back to the primary. The command can be directed to a specific set of transport providers by using the `-t` option with a comma-separated list of transport providers. Any host can use `rfadmin` with no options to print information about the domain. The user must have `root` permissions to use this command, except in the case when the `-q` option is used.

`-a` *hostname*	Add a host to a domain that is served by this domain name server. *hostname* must be of the form *domain.nodename*. It creates an entry for *hostname* in the *domain*/`passwd` file and prompts for an initial authentication password; the password prompting process conforms with that of `passwd`(1).
`-r` *hostname*	Remove a host, *hostname*, from its domain by removing it from the *domain*/`passwd file`.
`-p`	Used to pass the domain name server responsibilities back to a primary or to a secondary name server.
`-t` *transport1, transport2 ...*	Select transport provider(s). The `-t` option is used only with the `-p` option.
`-q`	Tells if RFS is running.
`-o` *option*	Sets RFS system option. *option* is one of the following: `loopback` - Enable loop back facility. This allows a resource advertised by a computer to be mounted by the same computer. `loopback` is off by default.

`noloopback` - Turn off the loop back facility. `noloopback` is the default.

`loopmode` - Check if the loop back facility is on or off.

ERRORS

When used with the *−a* option, if *hostname* is not unique in the domain, an error message will be sent to standard error.

When used with the *−r* option, if (1) *hostname* does not exist in the domain, (2) *hostname* is defined as a domain name server, or (3) there are resources advertised by *hostname*, an error message will be sent to standard error.

When used with the *−p* option to change the domain name server, if there are no backup name servers defined for *domain*, an error message will be sent to standard error.

FILES

`/etc/rfs/auth.info/`*domain*`/passwd`
For each *domain*, this file is created on the primary, copied to all secondaries, and copied to all hosts that want to do password verification of hosts in the *domain*.

SEE ALSO

`passwd`(1), `dname`(1M), `rfstart`(1M), `rfstop`(1M), `umount`(1M).

NAME

`rfpasswd` – change Remote File Sharing host password

SYNOPSIS

`rfpasswd`

DESCRIPTION

`rfpasswd` updates the Remote File Sharing authentication password for a host; processing of the new password follows the same criteria as **passwd**(1). The updated password is registered at the domain name server (/etc/dfs/rfs/auth.info/*domain*/passwd) and replaces the password stored at the local host (`/etc/dfs/rfs/loc.passwd` file).

This command is restricted to the super-user.

NOTE: If you change your host password, make sure that hosts that validate your password are notified of this change. To receive the new password, hosts must obtain a copy of the *domain*/**passwd** file from the domain's primary name server. If this is not done, attempts to mount remote resources may fail!

ERRORS

If (1) the old password entered from this command does not match the existing password for this machine, (2) the two new passwords entered from this command do not match, (3) the new password does not satisfy the security criteria in **passwd**(1), (4) the domain name server does not know about this machine, or (5) the command is not run with super-user privileges, an error message will be sent to standard error. Also, Remote File Sharing must be running on your host and your domain's primary name server. A new password cannot be logged if a secondary is acting as the domain name server.

FILES

`/etc/dfs/rfs/auth.info/`*domain*`/passwd`
`/etc/dfs/rfs/loc.passwd`

SEE ALSO

`rfstart`(1M), `rfadmin`(1M).
passwd(1) in the *User's Reference Manual.*

NAME

`rfstart` – start Remote File Sharing

SYNOPSIS

`rfstart` [-v] [-p *primary_addr*]

DESCRIPTION

`rfstart` starts Remote File Sharing and defines an authentication level for incoming requests. (This command can only be used after the domain name server is set up and your computer's domain name and network specification have been defined using `dname`(1M).)

–v
: Specifies that verification of all clients is required in response to initial incoming mount requests; any host not in the file `/etc/rfs/auth.info/`*domain*/passwd for the `domain` they belong to, will not be allowed to mount resources from your host. If **–v** is not specified, hosts named in *domain*/passwd will be verified. Other hosts will be allowed to connect without verification.

–p *primary_addr*
: Indicates the primary domain name server for your domain. *primary_addr* can specify any of the following: the network address of the primary name server for a domain (*addr*); a list of address tuples when RFS is used over multiple transport providers (*transport1:addr1,transport2:addr2,* ...). An example of each type of specification follows:

    –p *addr*
    –p *transport1:addr1,transport2:addr2, ...*

    If the –p option is not specified, the address of the domain name server is taken from the associated `rfmaster` files. The –p *addr* specification is valid only when one transport provider is being used. See the `rfmaster`(1M) manual page for a description of the valid address syntax.

If the host password has not been set, `rfstart` will prompt for a password. The password prompting process must match the password entered for your machine at the primary domain name server (see `rfadmin`(1M)). If you remove the `loc.passwd` file or change domains, you will also have to reenter the password.

Also, when `rfstart` is run on a domain name server, entries in the `rfmaster`(4) file are syntactically validated.

This command is restricted to the super-user.

ERRORS

If syntax errors are found when validating an `rfmaster`(4) file, a warning describing each error will be sent to standard error.

An error message will be sent to standard error if any of the following conditions are true:

1. remote file sharing is already running
2. there is no communications network
3. a domain name server cannot be found

4. a domain name server does not recognize the machine
5. the command is run without super-user privileges

Remote file sharing will not start if a host password in `/etc/rfs/`*<transport>*`/loc.passwd` is corrupted. If you suspect this has happened, remove the file and run `rfstart` again to reenter your password.

NOTE: `rfstart` will NOT fail if your host password does not match the password on the domain name server. You will simply receive a warning message. However, if you try to mount a resource from the primary, or any other host that validates your password, the mount will fail if your password does not match the one that the host has listed for your machine.

FILES

`/etc/rfs/<transport>/rfmaster`
`/etc/rfs/<transport>/loc.passwd`

SEE ALSO

`share`(1M), `dname`(1M), `mount`(1M), `rfadmin`(1M), `rfstop`(1M), `unshare`(1M).
`rfmaster`(4) in the *Programmer's Reference Manual.*

NAME

`rfstop` – stop the Remote File Sharing environment

SYNOPSIS

`rfstop`

DESCRIPTION

`rfstop` disconnects a host from the Remote File Sharing environment until another `rfstart`(1M) is executed.

When executed on the domain name server, the domain name server responsibility is moved to a secondary name server as designated in the `rfmaster`(4) file. If there is no designated secondary name server `rfstop` will issue a warning message, Remote File Sharing will be stopped, and name service will no longer be available to the domain.

This command is restricted to the super-user.

ERRORS

If (1) there are resources currently advertised by this host, (2) resources from this machine are still remotely mounted by other hosts, (3) there are still remotely mounted resources in the local file system tree, (4) `rfstart`(1M) had not previously been executed, or (5) the command is not run with super-user privileges, an error message will be sent to standard error and Remote File Sharing will not be stopped.

SEE ALSO

adv(1M), mount(1M), `rfadmin`(1M), `rfstart`(1M), unadv(1M), `rfmaster`(4).

NAME

`rfuadmin` – Remote File Sharing notification shell script

SYNOPSIS

`/etc/rfs/rfuadmin` *message remote_resource* [*seconds*]

DESCRIPTION

The `rfuadmin` administrative shell script responds to unexpected Remote File Sharing events, such as broken network connections and forced unmounts, picked up by the `rfudaemon` process. This command is not intended to be run directly from the shell.

The response to messages received by `rfudaemon` can be tailored to suit the particular system by editing the `rfuadmin` script. The following paragraphs describe the arguments passed to `rfuadmin` and the responses.

`disconnect` *remote_resource*

A link to a remote resource has been cut. `rfudaemon` executes `rfuadmin`, passing it the message `disconnect` and the name of the disconnected resource. `rfuadmin` sends this message to all terminals using `wall`(1):

Remote_resource `has been disconnected from the system.`

Then it executes `fuser`(1M) to kill all processes using the resource, unmounts the resource [`umount`(1M)] to clean up the kernel, and starts `rmount` to try to remount the resource.

`fumount` *remote_resource*

A remote server machine has forced an unmount of a resource a local machine has mounted. The processing is similar to processing for a disconnect.

`fuwarn` *remote_resource seconds*

This message notifies `rfuadmin` that a resource is about to be unmounted. `rfudaemon` sends this script the `fuwarn` message, the resource name, and the number of seconds in which the forced unmount will occur. `rfuadmin` sends this message to all terminals:

Remote_resource `is being removed from the system in # seconds.`

SEE ALSO

`fumount`(1M), `rmount`(1M), `rfudaemon`(1M), `rfstart`(1M).
`wall`(1) in the *User's Reference Manual.*

NOTES

The console must be on when Remote File Sharing is running. If it's not, `rfuadmin` will hang when it tries to write to the console (`wall`) and recovery from disconected resources will not complete.

NAME

`rfudaemon` – Remote File Sharing daemon process

SYNOPSIS

`/etc/rfs/rfudaemon`

DESCRIPTION

The `rfudaemon` command is started automatically by `rfstart`(1M) and runs as a daemon process as long as Remote File Sharing is active. Its function is to listen for unexpected events, such as broken network connections and forced unmounts, and execute appropriate administrative procedures.

When such an event occurs, `rfudaemon` executes the administrative shell script `rfuadmin`, with arguments that identify the event. This command is not intended to be run from the shell. Here are the events:

`DISCONNECT`
: A link to a remote resource has been cut. `rfudaemon` executes `rfuadmin`, with two arguments: `disconnect` and the name of the disconnected resource.

`FUMOUNT`
: A remote server machine has forced an unmount of a resource a local machine has mounted. `rfudaemon` executes `rfuadmin`, with two arguments: `fumount` and the name of the disconnected resource.

`GETUMSG`
: A remote user-level program has sent a message to the local `rfudaemon`. Currently the only message sent is *fuwarn*, which notifies `rfuadmin` that a resource is about to be unmounted. It sends `rfuadmin` the *fuwarn*, the resource name, and the number of seconds in which the forced unmount will occur.

`LASTUMSG`
: The local machine wants to stop the `rfudaemon` [`rfstop`(1M)]. This causes `rfudaemon` to exit.

SEE ALSO

`rfstart`(1M), `rfuadmin`(1M).

NAME

`rlogind` – remote login server

SYNOPSIS

`in.rlogind` *host.port*

DESCRIPTION

`rlogind` is the server for the `rlogin`(1) program. The server provides a remote login facility with authentication based on privileged port numbers.

`rlogind` is invoked by `inetd`(1M) when a remote login connection is established, and executes the following protocol:

1) The server checks the client's source port. If the port is not in the range 0-1023, the server aborts the connection. The client's address and port number are passed as arguments to `rlogind` by `inetd` in the form *host.port* with host in hexadecimal and port in decimal.

2) The server checks the client's source address. If an entry for the client exists is both `/etc/hosts` and `/etc/hosts.equiv`, a user logging in from the client is not prompted for a password. If the address is associated with a host for which no corresponding entry exists in `/etc/hosts`, the user is prompted for a password, regardless of whether or not an entry for the client is present in `/etc/hosts.equiv` [see `hosts`(4) and `hosts.equiv`(4)].

Once the source port and address have been checked, `rlogind` allocates a pseudo-terminal and manipulates file descriptors so that the slave half of the pseudo-terminal becomes the `stdin`, `stdout`, and `stderr` for a login process. The login process is an instance of the `login`(1) program, invoked with the `-r` option. The login process then proceeds with the authentication process as described in `rshd`(1M), but if automatic authentication fails, it reprompts the user to login as one finds on a standard terminal line.

The parent of the login process manipulates the master side of the pseudo-terminal, operating as an intermediary between the login process and the client instance of the `rlogin` program. In normal operation, a packet protocol is invoked to provide `Ctrl-S` / `Ctrl-Q` type facilities and propagate interrupt signals to the remote programs. The login process propagates the client terminal's baud rate and terminal type, as found in the environment variable, `TERM`; see `environ`(4).

SEE ALSO

`inetd`(1M), `hosts`(4), `hosts.equiv`(4).

DIAGNOSTICS

All diagnostic messages are returned on the connection associated with the `stderr`, after which any network connections are closed. An error is indicated by a leading byte with a value of 1.

`Hostname for your address unknown.`
    No entry in the host name database existed for the client's machine.

`Try again.`
A *fork* by the server failed.

`/usr/bin/sh:` ...
The user's login shell could not be started.

NOTES

The authentication procedure used here assumes the integrity of each client machine and the connecting medium. This is insecure, but is useful in an "open" environment.

A facility to allow all data exchanges to be encrypted should be present.

NAME

`rmntstat` – display mounted resource information

SYNOPSIS

`rmntstat` [`-h`] [*resource*]

DESCRIPTION

When used with no options, `rmntstat` displays a list of all local Remote File Sharing resources that are remotely mounted, the local path name, and the corresponding clients. `rmntstat` returns the remote mount data regardless of whether a resource is currently advertised; this ensures that resources that have been unadvertised but are still remotely mounted are included in the report. When a *resource* is specified, `rmntstat` displays the remote mount information only for that resource. The `-h` option causes header information to be omitted from the display.

EXIT STATUS

If no local resources are remotely mounted, `rmntstat` will return a successful exit status.

ERRORS

If *resource* (1) does not physically reside on the local machine or (2) is an invalid resource name, an error message will be sent to standard error.

SEE ALSO

`mount`(1M), `fumount`(1M), `unadv`(1M).

NAME

rmnttry – attempt to mount queued remote resources

SYNOPSIS

/etc/rfs/rmnttry [*resource* ...]

DESCRIPTION

rmnttry sequences through the pending mount requests stored in /etc/rfs/rmnttab, trying to mount each resource. If a mount succeeds, the resource entry is removed from the /etc/rfs/rmnttab file.

If one or more resource names are supplied, mounts are attempted only for those resources, rather than for all pending mounts. Mounts are not attempted for resources not present in the /etc/rfs/rmnttab file (see rmount(1M)). If a mount invoked from rmnttry takes over 3 minutes to complete, rmnttry aborts the mount and issues a warning message.

rmnttry is typically invoked from a cron entry in /var/spool/cron/crontabs/root to attempt mounting queued resources at periodic intervals. The default strategy is to attempt mounts at 15 minute intervals. The cron entry for this is:

```
10,25,40,55 * * * * /etc/rfs/rmnttry >/dev/null
```

FILES

/etc/rfs/rmnttab pending mount requests

SEE ALSO

mount(1M), rmount(1M), rumount(1M), mnttab(4).
crontab(1) in the *User's Reference Manual.*

DIAGNOSTICS

An exit code of 0 is returned if all requested mounts succeeded, 1 is returned if one or more mounts failed, and 2 is returned for bad usage.

NAME

rmount – queue remote resource mounts

SYNOPSIS

/usr/sbin/rmount [-d[r] *resource directory*]

DESCRIPTION

rmount queues a remote resource for mounting. The command enters the resource request into /etc/rfs/rmnttab, which is formatted identically to mnttab(4). rmnttry(1M) is used to poll entries in this file.

When used without arguments, rmount prints a list of resources with pending mounts along with their destined directories, modes, and date of request. The resources are listed chronologically, with the oldest resource request appearing first.

The following options are available:

-d indicates that the *resource* is a remote resource to be mounted on directory.

-r indicates that the *resource* is to be mounted read-only. If the *resource* is write-protected, this flag must be used.

FILES

/etc/rfs/rmnttab pending mount requests

SEE ALSO

mount(1M), rmnttry(1M), rumount(1M), rmountall(1M), mnttab(4).

DIAGNOSTICS

An exit code of 0 is returned upon successful completion of rmount. Otherwise, a non-zero value is returned.

NAME

`rmountall`, `rumountall` – mount, unmount Remote File Sharing resources

SYNOPSIS

`/usr/sbin/rmountall` [–] " *file-system-table* " [...]
`/usr/sbin/rumountall [ -k ]`

DESCRIPTION

`rmountall` is a Remote File Sharing command used to mount remote resources according to a *file-system-table.* (`/etc/vfstab` is the recommended *file-system-table.*) `rmountall` also invokes the `rmnttry` command, which attempts to mount queued resources. The special file name "–" reads from the standard input.

`rumountall` causes all mounted remote resources to be unmounted and deletes all resources that were queued from `rmount`. The `-k` option sends a `SIGKILL` signal, via `fuser`, to processes that have files open.

These commands may be executed only by the super-user.

The format of the *file-system-table* is as follows:

column 1	block special file name of file system
column 2	mount-point directory
column 3	`-r` if to be mounted read-only; `-d` if remote resource
column 4	file system type (not used with Remote File Sharing)
column 5+	ignored

Columns are separated by white space. Lines beginning with a pound sign (#) are comments. Empty lines are ignored.

SEE ALSO

`fuser`(1M), `mount`(1M), `rfstart`(1M), `rmnttry`(1M), `rmount`(1M),
`sysadm`(1) in the *User's Reference Manual.*
`signal`(2) in the *Programmer's Reference Manual.*

DIAGNOSTICS

No messages are printed if the remote resources are mounted successfully.

Error and warning messages come from `mount`(1M).

NAME

`route` – manually manipulate the routing tables

SYNOPSIS

`route` [`-fn`] { `add` | `delete` } { *destination* | `default` } [`host` | `net`] [*gateway* [*metric*]]

DESCRIPTION

`route` manually manipulates the network routing tables normally maintained by the system routing daemon, `routed`(1M), or through default routes and redirect messages from routers. `route` allows the super-user to operate directly on the routing table for the specific host or network indicated by *destination*. `default` is available for gateways to use after all other routes have been attempted. The *gateway* argument, if present, indicates the network gateway to which packets should be addressed. The *metric* argument indicates the number of hops to the *destination*. The *metric* is required for *add* commands; it must be zero if the destination is on a directly-attached network, and nonzero if the route utilizes one or more gateways.

The add command instructs `route` to add a route to *destination*. `delete` deletes a route.

Routes to a particular host must be distinguished from those to a network. The optional keywords `net` and `host` force the destination to be interpreted as a network or a host, respectively. Otherwise, if the destination has a local address part of `INADDR_ANY`, then the route is assumed to be to a network; otherwise, it is presumed to be a route to a host. If the route is to a destination connected by a gateway, the *metric* parameter should be greater than 0. If adding a route with metric 0, the gateway given is the address of this host on the common network, indicating the interface to be used directly for transmission. All symbolic names specified for a *destination* (except `default`) or *gateway* are looked up in the hosts database using `gethostbyname`(3N). If this lookup fails, then the name is looked up in the networks database using `getnetbyname`(3N).

OPTIONS

`-f` Flush the routing tables of all gateway entries. If this is used in conjunction with one of the commands described above, `route` flushes the gateways before preforming the command.

`-n` Prevents attempts to print host and network names symbolically when reporting actions. This is useful, for example, when all name servers are down on your local net, so you need a route before you can contact the name server.

FILES

`/etc/hosts`
`/etc/networks`

SEE ALSO

`ioctl`(2), `gethostbyname`(3N), `getnetbyname`(3N), `routing`(4N), `routed`(1M).

DIAGNOSTICS

`add` [`host` | `net`] *destination*:*gateway*
: The specified route is being added to the tables. The values printed are from the routing table entry supplied in the `ioctl`(2) call.

`delete` [`host` | `net`] *destination*:*gateway*
: The specified route is being deleted.

destination done
: When the `-f` flag is specified, each routing table entry deleted is indicated with a message of this form.

`Network is unreachable`
: An attempt to add a route failed because the gateway listed was not on a directly-connected network. Give the next-hop gateway instead.

`not in table`
: A delete operation was attempted for an entry that is not in the table.

`routing table overflow`
: An add operation was attempted, but the system was unable to allocate memory to create the new entry.

NAME

routed – network routing daemon

SYNOPSIS

`in.routed` [`-qstv`] [*logfile*]

DESCRIPTION

`routed` is invoked at boot time to manage the network routing tables. The routing daemon uses a variant of the Xerox NS Routing Information Protocol in maintaining up to date kernel routing table entries.

In normal operation `routed` listens on `udp`(4P) socket 520 (decimal) for routing information packets. If the host is an internetwork router, it periodically supplies copies of its routing tables to any directly connected hosts and networks.

When `routed` is started, it uses the `SIOCGIFCONF` `ioctl`(2) to find those directly connected interfaces configured into the system and marked up (the software loopback interface is ignored). If multiple interfaces are present, it is assumed the host will forward packets between networks. `routed` then transmits a *request* packet on each interface (using a broadcast packet if the interface supports it) and enters a loop, listening for *request* and *response* packets from other hosts.

When a *request* packet is received, `routed` formulates a reply based on the information maintained in its internal tables. The *response* packet generated contains a list of known routes, each marked with a hop count metric (a count of 16, or greater, is considered infinite). The metric associated with each route returned provides a metric relative to the sender.

request packets received by `routed` are used to update the routing tables if one of the following conditions is satisfied:

(1) No routing table entry exists for the destination network or host, and the metric indicates the destination is reachable (that is, the hop count is not infinite).

(2) The source host of the packet is the same as the router in the existing routing table entry. That is, updated information is being received from the very internetwork router through which packets for the destination are being routed.

(3) The existing entry in the routing table has not been updated for some time (defined to be 90 seconds) and the route is at least as cost effective as the current route.

(4) The new route describes a shorter route to the destination than the one currently stored in the routing tables; the metric of the new route is compared against the one stored in the table to decide this.

When an update is applied, `routed` records the change in its internal tables and generates a *response* packet to all directly connected hosts and networks. `routed` waits a short period of time (no more than 30 seconds) before modifying the kernel's routing tables to allow possible unstable situations to settle.

In addition to processing incoming packets, `routed` also periodically checks the routing table entries. If an entry has not been updated for 3 minutes, the entry's metric is set to infinity and marked for deletion. Deletions are delayed an additional 60 seconds to insure the invalidation is propagated throughout the internet.

Hosts acting as internetwork routers gratuitously supply their routing tables every 30 seconds to all directly connected hosts and networks.

Supplying the **-s** option forces `routed` to supply routing information whether it is acting as an internetwork router or not. The **-q** option is the opposite of the **-s** option. If the **-t** option is specified, all packets sent or received are printed on the standard output. In addition, `routed` will not divorce itself from the controlling terminal so that interrupts from the keyboard will kill the process. Any other argument supplied is interpreted as the name of file in which `routed`'s actions should be logged. This log contains information about any changes to the routing tables and a history of recent messages sent and received which are related to the changed route. The **-v** option allows a logfile to be created showing the changes made to the routing tables with a timestamp.

In addition to the facilities described above, `routed` supports the notion of distant *passive* and *active* gateways. When `routed` is started up, it reads the file **gateways** to find gateways which may not be identified using the `SIOGIFCONF` `ioctl`. Gateways specified in this manner should be marked passive if they are not expected to exchange routing information, while gateways marked active should be willing to exchange routing information (that is, they should have a `routed` process running on the machine). Passive gateways are maintained in the routing tables forever and information regarding their existence is included in any routing information transmitted. Active gateways are treated equally to network interfaces. Routing information is distributed to the gateway and if no routing information is received for a period of the time, the associated route is deleted.

The **gateways** is comprised of a series of lines, each in the following format:

> < `net` | `host` > *filename1* `gateway` *filename2* `metric` *value* < `passive` | `active` >

The `net` or `host` keyword indicates if the route is to a network or specific host.

filename1 is the name of the destination network or host. This may be a symbolic name located in **networks** or **hosts**, or an Internet address specified in dot notation; see `inet`(3N).

filename2 is the name or address of the gateway to which messages should be forwarded.

value is a metric indicating the hop count to the destination host or network.

The keyword **passive** or **active** indicates if the gateway should be treated as passive or active (as described above).

FILES

`/etc/gateways`	for distant gateways
`/etc/networks`	
`/etc/hosts`	

SEE ALSO

`ioctl`(2), `inet`(7), `udp`(7).

NOTES

The kernel's routing tables may not correspond to those of `routed` for short periods of time while processes utilizing existing routes exit; the only remedy for this is to place the routing process in the kernel.

`routed` should listen to intelligent interfaces, such as an IMP, and to error protocols, such as ICMP, to gather more information.

NAME

`rpcbind` – universal addresses to RPC program number mapper

SYNOPSIS

`rpcbind`

DESCRIPTION

`rpcbind` is a server that converts RPC program numbers into universal addresses. It must be running to make RPC calls.

When an RPC service is started, it will tell `rpcbind` at what address it is listening, and what RPC program numbers it is prepared to serve. When a client wishes to make an RPC call to a given program number, it will first contact `rpcbind` on the server machine to determine the address where RPC packets should be sent.

Normally, standard RPC servers are started by port monitors, so `rpcbind` must be started before port monitors are invoked.

`rpcbind` is restricted to users with appropriate privileges.

NOTES

If `rpcbind` crashes, all RPC servers must be restarted.

SEE ALSO

rpcinfo(1M).

NAME

`rpcinfo` – report RPC information

SYNOPSIS

`rpcinfo` [*host*]
`rpcinfo -p` [*host*]
`rpcinfo -T` *transport host program version*
`rpcinfo` [`-n` *portnum*] `-u` *host program version*
`rpcinfo` [`-n` *portnum*] `-t` *host program version*
`rpcinfo -a` *serv_address* `-T` *transport program* [*version*]
`rpcinfo -b` [`-T` *transport*] *program version*
`rpcinfo -d` [`-T` *transport*] *program version*

DESCRIPTION

`rpcinfo` makes an RPC call to an RPC server and reports what it finds.

In the first synopsis, it lists all the registered RPC services with `rpcbind` on *host*. If *host* is not specified, it defaults to the local host.

In the second synopsis, it lists all the RPC services registered with `portmapper`. Also note that the format of the information is different in the first and the second synopsis; this is because in the first case, `rpcbind` (version 3) is contacted, while in the second case `portmap` (version 2) is contacted for information.

The third synopsis makes an RPC call to procedure 0 of *program* and *version* on the specified *host* and reports whether a response was received. *transport* is the transport which has to be used for contacting the given service. The remote address of the service is obtained by making a call to remote `rpcbind`.

The other ways of using `rpcinfo` are described below. See EXAMPLES.

The following options are available:

`-T` *transport* Specify the transport on which the service is required. If this option is not specified, `rpcinfo` uses the transport specified in the `NETPATH` environment variable, or if that is unset or null, in the netconfig database. This is a generic option, and can be used in conjunction with any other option, except the `-b` option.

`-a` *serv_address* Use *serv_address* as the (universal) address for the service on *transport*, to ping procedure 0 of the specified *program* and report whether a response was received. The use of `-T` option is required with `-a` option.

If version number is not specified, `rpcinfo` tries to ping all the available version numbers for that program number. This option avoids calls to remote `rpcbind` to find the address of the service. The *serv_address* is specified in universal address format of the given transport.

`-b` Make an RPC broadcast to procedure 0 of the specified *program* and *version* and report all hosts that respond. If *transport* is specified, it broadcasts its request only on the transport specified through *transport*. If broadcasting is not supported by any transport, an error message is printed. Only UDP transports support broadcasting.

-d	Delete registration for the RPC service of the specified *program* and *version*. If *transport* is specified, unregister the service on only that transport, otherwise unregister the services on all the transports on which it was registered. This option can be exercised only by the privileged user.
-n	Use *portnum* as the port number for the `-t` and `-u` options instead of the port number given by the portmapper. Use of this option avoids a call to the remote portmapper to find out the address of the service.
-p	Probe the portmapper on *host*, and print a list of all registered RPC programs. If *host* is not specified, it defaults to the local host.
-t	Make an RPC call to procedure 0 of *program* on the specified *host* using TCP, and report whether a response was received.
-u	Make an RPC call to procedure 0 of *program* on the specified *host* using UDP, and report whether a response was received.

The *program* argument is a number.

If a *version* is specified, `rpcinfo` attempts to call that version of the specified *program*. Otherwise, `rpcinfo` attempts to find all the registered version numbers for the specified *program* by calling version 0, which is presumed not to exist; if it does exist, `rpcinfo` attempts to obtain this information by calling an extremely high version number instead, and attempts to call each registered version. Note: the version number is required for -b and -d options.

EXAMPLES

To show all of the RPC services registered on the local machine use:

```
$ rpcinfo
```

To show all of the RPC services registered with rpcbind on the machine named `klaxon` use:

```
$ rpcinfo klaxon
```

To show if the RPC service with program number *prog_no* and version *vers* is registered on the machine named `klaxon` for the transport `tcp` use:

```
$ rpcinfo -T tcp klaxon prog_no vers
```

To show all of the RPC services registered with the portmapper on the local machine use:

```
$ rpcinfo -p
```

To ping version 2 of `rpcbind` (program number 100000) on host `sparky`:

```
$ rpcinfo -t sparky 100000 2
```

To delete the registration for version 1 of the walld (program number 100008) service for all transports use:

```
# rpcinfo -d 100008 1
```

SEE ALSO

rpcbind(1M), rpc(4).

NAME

`rshd` – remote shell server

SYNOPSIS

`in.rshd` *host.port*

DESCRIPTION

`rshd` is the server for the `rsh`(1) program. The server provides remote execution facilities with authentication based on privileged port numbers.

`rshd` is invoked by `inetd`(1M) each time a shell service is requested, and executes the following protocol:

1) The server checks the client's source port. If the port is not in the range 0-1023, the server aborts the connection. The clients host address (in hex) and port number (in decimal) are the argument passed to `rshd`.
2) The server reads characters from the socket up to a null (`\0`) byte. The resultant string is interpreted as an ASCII number, base 10.
3) If the number received in step 1 is non-zero, it is interpreted as the port number of a secondary stream to be used for the `stderr`. A second connection is then created to the specified port on the client's machine. The source port of this second connection is also in the range 0-1023.
4) The server checks the client's source address. If the address is associated with a host for which no corresponding entry exists in the host name data base [see `hosts`(4)], the server aborts the connection.
5) A null terminated user name of at most 16 characters is retrieved on the initial socket. This user name is interpreted as a user identity to use on the `server`'s machine.
6) A null terminated user name of at most 16 characters is retrieved on the initial socket. This user name is interpreted as the user identity on the `client`'s machine.
7) A null terminated command to be passed to a shell is retrieved on the initial socket. The length of the command is limited by the upper bound on the size of the system's argument list.
8) `rshd` then validates the user according to the following steps. The remote user name is looked up in the password file and a `chdir` is performed to the user's home directory. If the lookup or fails, the connection is terminated. If the `chdir` fails, it does a `chdir` to / (root). If the user is not the super-user, (user ID 0), the file `/etc/hosts.equiv` is consulted for a list of hosts considered equivalent. If the client's host name is present in this file, the authentication is considered successful. If the lookup fails, or the user is the super-user, then the file `.rhosts` in the home directory of the remote user is checked for the machine name and identity of the user on the client's machine. If this lookup fails, the connection is terminated.
9) A null byte is returned on the connection associated with the `stderr` and the command line is passed to the normal login shell of the user. The shell inherits the network connections established by `rshd`.

FILES

`/etc/hosts.equiv`

SEE ALSO

`rsh`(1)

DIAGNOSTICS

The following diagnostic messages are returned on the connection associated with the `stderr`, after which any network connections are closed. An error is indicated by a leading byte with a value of 1 (0 is returned in step 9 above upon successful completion of all the steps prior to the command execution).

`locuser too long`
: The name of the user on the client's machine is longer than 16 characters.

`remuser too long`
: The name of the user on the remote machine is longer than 16 characters.

`command too long`
: The command line passed exceeds the size of the argument list (as configured into the system).

`Hostname for your address unknown.`
: No entry in the host name database existed for the client's machine.

`Login incorrect.`
: No password file entry for the user name existed.

`Permission denied.`
: The authentication procedure described above failed.

`Can't make pipe.`
: The pipe needed for the `stderr` was not created.

`Try again.`
: A *fork* by the server failed.

NOTES

The authentication procedure used here assumes the integrity of each client machine and the connecting medium. This is insecure, but is useful in an open environment.

A facility to allow all data exchanges to be encrypted should be present.

NAME

rumount – cancel queued remote resource request

SYNOPSIS

/etc/rfs/rumount *resource* ...

DESCRIPTION

rumount cancels a request for one or more resources that are queued for mount. The entries for the resources are deleted from /etc/rfs/rmnttab.

FILES

/etc/rfs/rmnttab — pending mount requests

SEE ALSO

mount(1M), rmnttry(1M), rmount(1M), rumountall(1M), mnttab(4).

DIAGNOSTICS

An exit code of 0 is returned if rumount completes successfully. A 1 is returned if the resource requested for dequeuing is not in /etc/rfs/rmnttab. A 2 is returned for bad usage or an error in reading or writing /etc/rfs/rmnttab.

NAME

rpc.rusersd – network username server

SYNOPSIS

/usr/lib/netsvc/rusers/rpc.rusersd

DESCRIPTION

rusersd is a server that returns a list of users on the host. The rusersd daemon may be started by inetd(1M) or listen(1M).

SEE ALSO

inetd(1M), listen(1M), pmadm(1M), sacadm(1M).

NAME

`rwall` – write to all users over a network

SYNOPSIS

`/usr/sbin/rwall` *hostname* ...

DESCRIPTION

`rwall` reads a message from standard input until EOF. It then sends this message, preceded by the line:

```
Broadcast Message . . .
```

to all users logged in on the specified host machines.

A machine can only receive such a message if it is running `rwalld`(1M), which may be started by `inetd`(1M) or `listen`(1M).

NOTES

The timeout is fairly short to allow transmission to a large group of machines (some of which may be down) in a reasonable amount of time. Thus the message may not get through to a heavily loaded machine.

SEE ALSO

`inetd`(1M), `listen`(1M), `pmadm`(1M), `rwalld`(1M), `sacadm`(1M), `wall`(1)

NAME

`rpc.rwalld` – network rwall server

SYNOPSIS

`/usr/lib/netsvc/rwall/rpc.rwalld`

DESCRIPTION

rwalld is a server that handles `rwall`(1M) requests. It is implemented by calling `wall`(1M) on all the appropriate network machines. The **rwalld** daemon may be started by `inetd`(1M) or `listen`(1M).

SEE ALSO

`inetd`(1M), `listen`(1M), `rwall`(1M), `wall`(1M).

NAME

rwhod, in.rwhod – system status server

SYNOPSIS

in.rwhod

DESCRIPTION

rwhod is the server which maintains the database used by the rwho(1) and ruptime(1) programs. Its operation is predicated on the ability to broadcast messages on a network.

rwhod operates as both a producer and consumer of status information. As a producer of information it periodically queries the state of the system and constructs status messages which are broadcast on a network. As a consumer of information, it listens for other rwhod servers' status messages, validating them, then recording them in a collection of files located in the directory /var/spool/rwho.

The rwho server transmits and receives messages at the port indicated in the rwho service specification, see services(4). The messages sent and received, are of the form:

```
struct     outmp {
      char out_line[8];    /* tty name */
      char out_name[8];    /* user id */
      long out_time;  /* time on */
};

struct     whod {
      char wd_vers;
      char wd_type;
      char wd_fill[2];
      int  wd_sendtime;
      int  wd_recvtime;
      char wd_hostname[32];
      int  wd_loadav[3];
      int  wd_boottime;
            struct     whoent {
            struct     outmp we_utmp;
            int   we_idle;
      } wd_we[1024 / sizeof (struct whoent)];
};
```

All fields are converted to network byte order prior to transmission. The load averages are as calculated by the w(1) program, and represent load averages over the 5, 10, and 15 minute intervals prior to a server's transmission. The host name included is that returned by the gethostname(2) system call. The array at the end of the message contains information about the users logged in to the sending machine. This information includes the contents of the utmp(4) entry for each non-idle terminal line and a value indicating the time since a character was last received on the terminal line.

Messages received by the `rwho` server are discarded unless they originated at a `rwho` server's port. In addition, if the host's name, as specified in the message, contains any unprintable ASCII characters, the message is discarded. Valid messages received by `rwhod` are placed in files named `whod.`*hostname* in the directory `/var/spool/rwho`. These files contain only the most recent message, in the format described above.

Status messages are generated approximately once every 60 seconds. `rwhod` performs an `nlist`(3) on `/stand/unix` every 10 minutes to guard against the possibility that this file is not the system image currently operating.

FILES

`/var/spool/rwho`

SEE ALSO

`rwho`(1), `ruptime`(1), `w`(1), `gethostname`(3), `nlist`(3), `utmp`(4).

NOTES

This service takes up progressively more network bandwidth as the number of hosts on the local net increases. For large networks, the cost becomes prohibitive.

`rwhod` should relay status information between networks. People often interpret the server dying as a machine going down.

NAME

share – make local resource available for mounting by remote systems

SYNOPSIS

share [-F *fstype*] [-o *specific_options*] [-d *description*] [*pathname* [*resourcename*]]

DESCRIPTION

The **share** command makes a resource available for mounting through a remote file system of type *fstype*. If the option -F *fstype* is omitted, the first file system type listed in file **/etc/dfs/fstypes** will be used as the default. *Specific_options* as well as the semantics of *resourcename* are specific to particular distributed file systems. When invoked with only a file system type, **share** displays all resources shared by the given file system to the local system. When invoked with no arguments, **share** displays all resources shared by the local system.

The *access_spec* is used to control access of the shared resource. It may be one of the following:

rw	*pathname* is shared read/write to all clients. This is also the default behavior.
rw=*client*[:*client*]...	*pathname* is shared read/write only to the listed clients. No other systems can access *resourcename*.
ro	*pathname* is shared read-only to all clients.
ro=*client*[:*client*]...	*pathname* is shared read-only only to the listed clients. No other systems can access *pathname*.

The -d flag may be used to provide a description of the resource being shared.

FILES

/etc/dfs/dfstab
/etc/dfs/sharetab
/etc/dfs/fstypes

SEE ALSO

unshare(1M)

NAME

`share` – make local NFS resource available for mounting by remote systems

SYNOPSIS

`share` [`-F nfs`] [`-o` *specific_options*] [`-d` *description*] *pathname*

DESCRIPTION

The `share` command makes local resources available for mounting by remote systems.

If no argument is specified, then `share` displays all resources currently shared, including NFS resources and resources shared through other distributed file system packages.

The following options are recognized:

`-o` *specific_options*
: Specify options in a comma-separated list of keywords and attribute-value-assertions for interpretation by the file-system-type-specific command.

  specific_options can be any combination of the following:

  `rw` Sharing will be read-write to all clients.

  `rw=`*client*[`:`*client*]...
  Sharing will be read-write to the listed clients; overrides the `ro` suboption for the clients specified.

  `ro` Sharing will be read-only to all clients.

  `ro=`*client*[`:`*client*]...
  Sharing will be read-only to the listed clients; overrides the `rw` suboption for the clients specified.

  `anon=`*uid*
  Set *uid* to be the effective user ID of unathenticated users if AUTH_DES authentication is used, or to be root if AUTH_UNIX authentication is used. By default, unknown users are given the effective user ID `UID_NOBODY`. If *uid* is set to –1, access is denied.

  `root=`*host*[`:`*host*]...
  Only root users from the specified hosts will have root access. By default, no host has root access.

  `secure`
  Clients must use the AUTH_DES authentication of RPC. AUTH_UNIX authentication is the default.

  If *specific_options* is not specified, then by default sharing will be read-write to all clients.

`-d` *description*
: Provide a comment that describes the resource to be shared.

pathname Specify the pathname of the resource to be shared.

FILES

/etc/dfs/fstypes
/etc/dfs/sharetab

SEE ALSO

unshare(1M)

NOTES

The command will fail if both ro and rw are specified. If the same client name exists in both the ro= and rw= lists, the rw will override the ro, giving read/write access to the client specified.

ro=, rw=, and root= are guaranteed to work over UDP but may not work over other transport providers.

If a resource is shared with a ro= list and a root= list, any host that is on the root= list will be given only read-only access, regardless of whether that host is specified in the ro= list, unless rw is declared as the default, or the host is mentioned in a rw= list. The same is true if the resource is shared with ro as the default. For example, the following share commands will give read-only permissions to hostb:

```
share -F nfs -oro=hosta,root=hostb /var
share -F nfs -oro,root=hostb /var
```

While the following will give read/write permissions to hostb:

```
share -F nfs -oro=hosta,rw=hostb,root=hostb /var
share -F nfs -oroot=hostb /var
```

NAME

share – make local RFS resource available for mounting by remote systems

SYNOPSIS

share [**–F** rfs] [**–o** *access_spec*] [**–d** *description*] [*pathname resourcename*]

DESCRIPTION

The **share** command makes a resource available for mounting through Remote File Sharing. The **–F** flag may be omitted if rfs is the first file system type listed in the file **/etc/dfs/fstypes**. When invoked with only a file system type (or no arguments), **share** displays all local resources shared through Remote File Sharing.

The *access_spec* is used to control client access of the shared resource. Clients may be specified in any of the following forms:

domain.
domain.system
system

The *access_spec* can be one of the following:

rw	*resourcename* is shared read/write to all clients. This is also the default behavior.
rw=*client*[*:client*]...	*resourcename* is shared read/write only to the listed clients. No other systems can access *resourcename*.
ro	*resourcename* is shared read-only to all clients.
ro=*client*[*:client*]...	*resourcename* is shared read-only only to the listed clients. No other systems can access *resourcename*.

The **–d** flag may be used to provide a description of the resource being shared.

ERRORS

If the network is not up and running or *pathname* is not a full path, an error message will be sent to standard error. If *pathname* isn't on a file system mounted locally or the *client* is specified but syntactically incorrect, an error message will be sent to standard error. If the same *resource* name in the network over the same transport provider is to be shared more than once, an error message will be sent to standard error.

FILES

/etc/dfs/dfstab
/etc/dfs/sharetab
/etc/dfs/fstypes

SEE ALSO

unshare(1M)

NAME

`slink` – streams linker

SYNOPSIS

`slink` [`-v`] [`-p`] [`-u`] [`-f`] [`-c` *file*] [`func` [*arg1 arg2* ...]]

DESCRIPTION

`slink` is a STREAMS configuration utility which is used to link together the various STREAMS modules and drivers required for STREAMS TCP/IP. Input to `slink` is in the form of a script specifying the STREAMS operations to be performed. Input is normally taken from the file `/etc/strcf`.

The following options may be specified on the `slink` command line:

`-c` *file*	Use *file* instead of `/etc/strcf`.
`-v`	Verbose mode (each operation is logged to `stderr`).
`-p`	Don't use persistent links (i.e., slink will remain in the background).
`-f`	Don't use persistent links and don't fork (i.e., slink will remain in foreground).
`-u`	Unlink persistent links (i.e., shut down network).

The configuration file contains a list of *functions*, each of which is composed of a list of *commands*. Each command is a call to one of the functions defined in the configuration file or to one of a set of built-in functions. Among the built-in functions are the basic STREAMS operations `open`, `link`, and `push`, along with several TCP/IP-specific functions.

`slink` processing consists of parsing the input file, then calling the user-defined function `boot`, which is normally used to set up the standard configuration at boot time. If a function is specified on the `slink` command line, that function will be called instead of `boot`.

By default, `slink` establishes streams with persistent links (`I_PLINK`) and exits following the execution of the specified function. If the `-p` flag is specified, `slink` establishes streams with regular links (`I_LINK`) and remains idle in the background, holding open whatever file descriptors have been opened by the configuration commands. If the `-f` flag is specified, `slink` establishes streams with regular links (`I_LINK`) and remains in the foreground, holding open whatever file descriptors have been opened by the configuration commands.

A function definition has the following form:

```
function-name {
      command1
      command2
      . . .
}
```

The syntax for commands is:

function arg1 arg2 arg3 ...

or

`var` = *function arg1 arg2 arg3 ...*

The placement of newlines is important: a newline must follow the left and right braces and every command. Extra newlines are allowed, i.e. where one newline is required, more than one may be used. A backslash (\) followed immediately by a newline is considered equivalent to a space, i.e. may be used to continue a command on a new line. The use of other white space characters (spaces and tabs) is at the discretion of the user, except that there must be white space separating the function name and the arguments of a command.

Comments are delimited by # and newline, and are considered equivalent to a newline.

Function and variable names may be any string of characters taken from A-Z, a-z, 0-9, and _, except that the first character cannot be a digit. Function names and variable names occupy separate name spaces. All functions are global and may be forward referenced. All variables are local to the functions in which they occur.

Variables are defined when they appear to the left of an equals (=) on a command line; for example,

```
        tcp = open /dev/tcp
```

The variable acquires the value returned by the command. In the above example, the value of the variable `tcp` will be the file descriptor returned by the `open` call.

Arguments to a command may be either variables, parameters, or strings.

A variable that appears as an argument must have been assigned a value on a previous command line in that function.

Parameters take the form of a dollar sign ($) followed by one or two decimal digits, and are replaced with the corresponding argument from the function call. If a given parameter was not specified in the function call, an error results (e.g. if a command references $3 and only two arguments were passed to the function, an execution error will occur).

Strings are sequences of characters optionally enclosed in double quotes ("). Quotes may be used to prevent a string from being interpreted as a variable name or a parameter, and to allow the inclusion of spaces, tabs, and the special characters {, }, =, and #. The backslash (\) may also be used to quote the characters {, }, =, #, ", and \ individually.

The following built-in functions are provided by `slink`:

`open` *path*	Open the device specified by pathname *path*. Returns a file descriptor referencing the open stream.
`link` *fd1 fd2*	Link the stream referenced by *fd2* beneath the stream referenced by *fd1*. Returns the link identifier associated with the link. Unless the `-f` or `-p` flag is specified on the command line, the streams will be linked with persistent links. Note: *fd2* cannot be used after this operation.
`push` *fd module*	Push the module *module* onto the stream referenced by *fd*.

`sifname` *fd link name* — Send a `SIOCSIFNAME` (set interface name) ioctl down the stream referenced by *fd* for the link associated with link identifier *link* specifying the name *name*.

`unitsel` *fd unit* — Send a `IF_UNITSEL` (unit select) ioctl down the stream referenced by *fd* specifying unit *unit*.

`dlattach` *fd unit* — Send a `DL_ATTACH_REQ` message down the stream referenced by *fd* specifying unit *unit*.

`initqp` *path qname lowat hiwat ...*
Send an `INITQPARMS` (initialize queue parameters) ioctl to the driver corresponding to pathname *path*. *qname* specifies the queue for which the low and high water marks will be set, and must be one of:

`hd`	stream head
`rq`	read queue
`wq`	write queue
`muxrq`	multiplexor read queue
`muxwq`	multiplexor write queue

lowat and *hiwat* specify the new low and high water marks for the queue. Both *lowat* and *hiwat* must be present. To change only one of these parameters, the other may be replaced with a dash (–). Up to five *qname lowat hiwat* triplets may be present.

`strcat` *str1 str2* — Concatenate strings *str1* and *str2* and return the resulting string.

`return` *val* — Set the return value for the current function to *val*.
Note: executing a `return` command does not terminate execution of the current function.

FILES

`/etc/strcf`

SEE ALSO

`strcf`(4)

NAME

`spray` – spray packets

SYNOPSIS

`/usr/sbin/spray` [`-c` *count*] [`-d` *delay*] [`-l` *length*] [`-t` *nettype host*

DESCRIPTION

`spray` sends a one-way stream of packets to *host* using RPC, and reports how many were received, as well as the the transfer rate. The *host* argument can be either a name or an Internet address.

The following options are available:

`-c` *count*	Specify how many packets to send. The default value of *count* is the number of packets required to make the total stream size 100000 bytes.
`-d` *delay*	Specify how many microseconds to pause between sending each packet. The default is 0.
`-l` *length*	The *length* parameter is the numbers of bytes in the Ethernet packet that holds the RPC call message. Since the data is encoded using XDR, and XDR only deals with 32 bit quantities, not all values of *length* are possible, and `spray` rounds up to the nearest possible value. When *length* is greater than 1514, then the RPC call can no longer be encapsulated in one Ethernet packet, so the *length* field no longer has a simple correspondence to Ethernet packet size. The default value of *length* is 86 bytes (the size of the RPC and UDP headers).
`-t` *nettype*	Specify clas of transports. Defaults to `netpath`. See `rpc`(3N) for a description of supported classes.

SEE ALSO

`sprayd`(1M), `rpc`(3N)

NAME

rpc.sprayd – spray server

SYNOPSIS

/usr/lib/netsvc/spray/rpc.sprayd

DESCRIPTION

rpc.sprayd is a server which records the packets sent by spray(1M). The rpc.sprayd daemon may be started by inetd(1M) or listen(1M).

SEE ALSO

inetd(1M) listen(1M), pmadm(1M), sacadm(1M), spray(1M)

NAME

`statd` – network status monitor

SYNOPSIS

`/usr/lib/nfs/statd`

DESCRIPTION

`statd` is an intermediate version of the status monitor. It interacts with `lockd`(1M) to provide the crash and recovery functions for the locking services on NFS.

FILES

`/etc/sm`
`/etc/sm.bak`
`/etc/state`

SEE ALSO

`lockd`(1M)

NOTES

The crash of a site is only detected upon its recovery.

NAME

`talkd`, `in.talkd` – server for talk program

SYNOPSIS

`in.talkd`

DESCRIPTION

`talkd` is a server used by the `talk`(1) program. It listens at the UDP port indicated in the "talk" service description; see `services`(4). The actual conversation takes place on a TCP connection that is established by negotiation between the two machines involved.

SEE ALSO

`talk`(1), `inetd`(1M), `services`(4).

NOTES

The protocol is architecture dependent.

NAME

telnetd – DARPA TELNET protocol server

SYNOPSIS

in.telnetd

DESCRIPTION

telnetd is a server which supports the DARPA standard TELNET virtual terminal protocol. telnetd is invoked by the internet server [see inetd(1M)], normally for requests to connect to the TELNET port as indicated by the /etc/services file [see services(4)].

telnetd operates by allocating a pseudo-terminal device for a client, then creating a login process which has the slave side of the pseudo-terminal as its standard input, output, and error. telnetd manipulates the master side of the pseudo-terminal, implementing the TELNET protocol and passing characters between the remote client and the login process.

When a TELNET session is started up, telnetd sends TELNET options to the client side indicating a willingness to do *remote echo* of characters, to *suppress go ahead*, and to receive *terminal type information* from the remote client. If the remote client is willing, the remote terminal type is propagated in the environment of the created login process. The pseudo-terminal allocated to the client is configured to operate in cooked mode, and with XTABS, ICRNL, and ONLCR enabled [see termio(4)].

telnetd is willing to do: *echo*, *binary*, *suppress go ahead*, and *timing mark*.
telnetd is willing to have the remote client do: *binary*, *terminal type*, and *suppress go ahead*.

SEE ALSO

telnet(1)

Postel, Jon, and Joyce Reynolds, "Telnet Protocol Specification," RFC 854, Network Information Center, SRI International, Menlo Park, Calif., May 1983.

NOTES

Some TELNET commands are only partially implemented.

The TELNET protocol allows for the exchange of the number of lines and columns on the user's terminal, but telnetd doesn't make use of them.

Binary mode has no common interpretation except between similar operating systems

The terminal type name received from the remote client is converted to lower case.

The *packet* interface to the pseudo-terminal should be used for more intelligent flushing of input and output queues.

telnetd never sends TELNET *go ahead* commands.

telnetd can only support 64 pseudo-terminals.

NAME

`tftpd` – DARPA Trivial File Transfer Protocol server

SYNOPSIS

`in.tftpd` [**-s**] [*homedir*]

DESCRIPTION

`tftpd` is a server that supports the DARPA Trivial File Transfer Protocol (TFTP). This server is normally started by `inetd`(1M) and operates at the port indicated in the `tftp` Internet service description in the `/etc/inetd.conf` file. By default, the entry for `tftpd` in `etc/inetd.conf` is commented out. To make `tftpd` operational, the comment character(s) must be deleted from the file. See `inetd.conf`(4) for details.

Before responding to a request, the server attempts to change its current directory to *homedir*; the default value is `/tftpboot`.

OPTIONS

-s Secure. When specified, the directory change must succeed; and the daemon also changes its root directory to *homedir*.

The use of *tftp* does not require an account or password on the remote system. Due to the lack of authentication information, *tftpd* will allow only publicly readable files to be accessed. Files may be written only if they already exist and are publicly writable. Note that this extends the concept of public to include all users on all hosts that can be reached through the network; this may not be appropriate on all systems, and its implications should be considered before enabling this service.

`tftpd` runs with the user ID and group ID set to [GU]ID_NOBODY. –2, under the assumption that no files exist with that owner or group. However, nothing checks this assumption or enforces this restriction.

SEE ALSO

`tftp`(1), `inetd`(1M), `ipallocd`(1M), `netconfig`(4).

Sollins, K.R., *The TFTP Protocol (Revision 2)*, RFC 783, Network Information Center, SRI International, Menlo Park, Calif., June 1981.

NAME

`tnamed`, `in.tnamed` – DARPA trivial name server

SYNOPSIS

`in.tnamed` [**-v**]

DESCRIPTION

`tnamed` is a server that supports the DARPA Name Server Protocol. The name server operates at the port indicated in the name service description [see `services`(4)], and is invoked by `inetd`(1M) when a request is made to the name server.

OPTIONS

-v Invoke the daemon in verbose mode.

SEE ALSO

`uucp`(1C), `inetd`(1M), `services`(4).

Postel, Jon, *Internet Name Server*, IEN 116, SRI International, Menlo Park, California, August 1979.

NOTES

The protocol implemented by this program is obsolete. Its use should be phased out in favor of the Internet Domain Name Service (DNS) protocol. See `named`(1M).

NAME

`trpt` – transliterate protocol trace

SYNOPSIS

`trpt` [`-afjst`] [`-p` *hex-address*] [*system* [*core*]]

DESCRIPTION

`trpt` interrogates the buffer of TCP trace records created when a socket is marked for debugging [see `getsockopt`(3N)], and prints a readable description of these records. When no options are supplied, `trpt` prints all the trace records found in the system grouped according to TCP connection protocol control block (PCB). The following options may be used to alter this behavior.

OPTIONS

`-a` In addition to the normal output, print the values of the source and destination addresses for each packet recorded.

`-f` Follow the trace as it occurs, waiting a short time for additional records each time the end of the log is reached.

`-j` Just give a list of the protocol control block addresses for which there are trace records.

`-s` In addition to the normal output, print a detailed description of the packet sequencing information.

`-t` In addition to the normal output, print the values for all timers at each point in the trace.

`-p` *hex-address*
Show only trace records associated with the protocol control block, the address of which follows.

The recommended use of `trpt` is as follows. Isolate the problem and enable debugging on the `socket`(s) involved in the connection. Find the address of the protocol control blocks associated with the sockets using the `-A` option to `netstat`(1M). Then run `trpt` with the `-p` option, supplying the associated protocol control block addresses. The `-f` option can be used to follow the trace log once the trace is located. If there are many sockets using the debugging option, the `-j` option may be useful in checking to see if any trace records are present for the socket in question.

If debugging is being performed on a system or core file other than the default, the last two arguments may be used to supplant the defaults.

FILES

`/stand/unix`
`/dev/kmem`

SEE ALSO

`netstat`(1M), `getsockopt`(3N).

DIAGNOSTICS

`no namelist`
When the system image does not contain the proper symbols to find the trace buffer; others which should be self explanatory.

NOTES

Should also print the data for each input or output, but this is not saved in the trace record.

The output format is inscrutable and should be described here.

NAME

`unshare` – make local resource unavailable for mounting by remote systems

SYNOPSIS

`unshare` [`-F` *fstype*] [`-o` *specific_options*] [*pathname* | *resourcename*]

DESCRIPTION

The `unshare` command makes a shared local resource unavailable to file system type *fstype*. If the option `-F` *fstype* is omitted, then the first file system type listed in file `/etc/dfs/fstypes` will be used as the default. *Specific_options*, as well as the semantics of *resourcename*, are specific to particular distributed file systems.

FILES

`/etc/dfs/fstypes`
`/etc/dfs/sharetab`

SEE ALSO

`share`(1M), `shareall`(1M).

NOTES

If *pathname* or *resourcename* is not found in the shared information, an error message will be sent to standard error.

NAME

unshare – make local NFS resource unavailable for mounting by remote systems

SYNOPSIS

`unshare [ -F nfs ]` *pathname*

DESCRIPTION

The **unshare** command makes local resources unavailable for mounting by remote systems. The shared resource must correspond to a line with NFS as the *fstype* in the file `/etc/dfs/sharetab`. The `-F` option may be omitted if NFS is the first file system type listed in the files `/etc/dfs/fstypes`.

FILES

`/etc/dfs/fstypes`
`/etc/dfs/sharetab`

SEE ALSO

share(1M)

NAME

unshare – make local RFS resource unavailable for mounting by remote systems

SYNOPSIS

unshare [**-F** rfs] {*pathname* | *resourcename*}

DESCRIPTION

The **unshare** command makes a shared resource unavailable through Remote File Sharing. The shared resource must correspond to a line with rfs as the *fstype* in the file **/etc/dfs/sharetab**. The **-F** flag may be omitted if RFS is the first file system type listed in the file **/etc/dfs/fstypes**.

FILES

/etc/dfs/dfstab
/etc/dfs/fstypes
/etc/dfs/sharetab

SEE ALSO

unshare(1M), **share**(1M)

NAME

ethers – Ethernet address mapping operations

SYNOPSIS

```
#include <sys/types.h>
#include <sys/socket.h>
#include <net/if.h>
#include <netinet/in.h>
#include <netinet/if_ether.h>

     char *
     ether_ntoa(e)
          struct ether_addr *e;

     struct ether_addr *
     ether_aton(s)
          char *s;

     ether_ntohost(hostname, e)
          char *hostname;
          struct ether_addr *e;

     ether_hostton(hostname, e)
          char *hostname;
          struct ether_addr *e;

     ether_line(l, e, hostname)
          char *l;
          struct ether_addr *e;
          char *hostname;
```

DESCRIPTION

These routines are useful for mapping 48 bit Ethernet numbers to their ASCII representations or their corresponding host names, and vice versa.

The function `ether_ntoa()` converts a 48 bit Ethernet number pointed to by `e` to its standard ASCII representation; it returns a pointer to the ASCII string. The representation is of the form *x:x:x:x:x:x* where *x* is a hexadecimal number between 0 and ff. The function `ether_aton()` converts an ASCII string in the standard representation back to a 48 bit Ethernet number; the function returns NULL if the string cannot be scanned successfully.

The function `ether_ntohost()` maps an Ethernet number (pointed to by `e`) to its associated hostname. The string pointed to by hostname must be long enough to hold the hostname and a NULL character. The function returns zero upon success and non-zero upon failure. Inversely, the function `ether_hostton()` maps a hostname string to its corresponding Ethernet number; the function modifies the Ethernet number pointed to by `e`. The function also returns zero upon success and non-zero upon failure. The function `ether_line()` scans a line (pointed to by `l`) and sets the hostname and the Ethernet number (pointed to by `e`). The string pointed to by hostname must be long enough to hold the hostname and a NULL character. The function returns zero upon success and non-zero upon failure. The format of the scanned line is described by `ethers`(4).

FILES

`/etc/ethers`

SEE ALSO

ethers(4)

NAME

resolver, res_mkquery, res_send, res_init, dn_comp, dn_expand – resolver routines

SYNOPSIS

```
#include <sys/types.h>
#include <netinet/in.h>
#include <arpa/nameser.h>
#include <resolv.h>

res_mkquery(op, dname, class, type, data, datalen, newrr, buf, buflen)
int op;
char *dname;
int class, type;
char *data;
int datalen;
struct rrec *newrr;
char *buf;
int buflen;

res_send(msg, msglen, answer, anslen)
char *msg;
int msglen;
char *answer;
int anslen;

res_init

dn_comp(exp_dn, comp_dn, length, dnptrs, lastdnptr)
char *exp_dn, *comp_dn;
int length;
char **dnptrs, **lastdnptr;

dn_expand(msg, msglen, comp_dn, exp_dn, length)
char *msg, *comp_dn, exp_dn;
int  msglen, length;
```

DESCRIPTION

These routines are used for making, sending and interpreting packets to Internet domain name servers. Global information that is used by the resolver routines is kept in the variable _res. Most of the values have reasonable defaults and can be ignored. Options are a simple bit mask and are OR'ed in to enable. Options stored in _res.options are defined in /usr/include/resolv.h and are as follows.

RES_INIT	True if the initial name server address and default domain name are initialized (that is, res_init has been called).
RES_DEBUG	Print debugging messages.
RES_AAONLY	Accept authoritative answers only. res_send will continue until it finds an authoritative answer or finds an error. Currently this is not implemented.

`RES_USEVC`	Use TCP connections for queries instead of UDP.
`RES_STAYOPEN`	Used with `RES_USEVC` to keep the TCP connection open between queries. This is useful only in programs that regularly do many queries. UDP should be the normal mode used.
`RES_IGNTC`	Unused currently (ignore truncation errors, that is, do not retry with TCP).
`RES_RECURSE`	Set the recursion desired bit in queries. This is the default. **`res_send`** does not do iterative queries and expects the name server to handle recursion.
`RES_DEFNAMES`	Append the default domain name to single label queries. This is the default.

`res_init` reads the initialization file to get the default domain name and the Internet address of the initial hosts running the name server. If this line does not exist, the host running the resolver is tried. `res_mkquery` makes a standard query message and places it in *buf*. `res_mkquery` will return the size of the query or −1 if the query is larger than *buflen*. *op* is usually `QUERY` but can be any of the query types defined in `/usr/include/arpa/nameser.h`. *dname* is the domain name. If *dname* consists of a single label and the `RES_DEFNAMES` flag is enabled (the default), *dname* will be appended with the current domain name. The current domain name is defined in a system file and can be overridden by the environment variable `LOCALDOMAIN`. *newrr* is currently unused but is intended for making update messages.

`res_send` sends a query to name servers and returns an answer. It will call `res_init` if `RES_INIT` is not set, send the query to the local name server, and handle timeouts and retries. The length of the message is returned or −1 if there were errors.

`dn_expand` expands the compressed domain name *comp_dn* to a full domain name. Expanded names are converted to upper case. *msg* is a pointer to the beginning of the message, *exp_dn* is a pointer to a buffer of size *length* for the result. The size of compressed name is returned or −1 if there was an error.

`dn_comp` compresses the domain name *exp_dn* and stores it in *comp_dn*. The size of the compressed name is returned or −1 if there were errors. *length* is the size of the array pointed to by *comp_dn*. *dnptrs* is a list of pointers to previously compressed names in the current message. The first pointer points to to the beginning of the message and the list ends with `NULL`. *lastdnptr* is a pointer to the end of the array pointed to *dnptrs*. A side effect is to update the list of pointers for labels inserted into the message by `dn_comp` as the name is compressed. If *dnptr* is `NULL`, do not try to compress names. If *lastdnptr* is `NULL`, do not update the list.

FILES

```
/usr/include/arpa/nameserv.h
/usr/include/netinet/in.h
```

```
/usr/include/resolv.h
/usr/include/sys/types.h
/etc/resolv.conf
/usr/lib/libresolv.a
```

SEE ALSO

named(1M), `resolv.conf`(4).

NOTES

`/usr/lib/libresolv.a` is necessary for compiling programs.

Programs must be loaded with the option `-lresolv`.

NAME

`rexec` – return stream to a remote command

SYNOPSIS

```
rem = rexec(ahost, inport, user, passwd, cmd, fd2p);
char **ahost;
u_short inport;
char *user, *passwd, *cmd;
int *fd2p;
```

DESCRIPTION

`rexec()` looks up the host **ahost* using `gethostbyname` [see `gethostent`(3N)], returning −1 if the host does not exist. Otherwise **ahost* is set to the standard name of the host. If a username and password are both specified, then these are used to authenticate to the foreign host; otherwise the environment and then the user's `.netrc` file in his home directory are searched for appropriate information. If all this fails, the user is prompted for the information.

The port `inport` specifies which well-known DARPA Internet port to use for the connection. The protocol for connection is described in detail in `rexecd`(1M).

If the call succeeds, a socket of type `SOCK_STREAM` is returned to the caller, and given to the remote command as its standard input and standard output. If *fd2p* is non-zero, then a auxiliary channel to a control process will be setup, and a descriptor for it will be placed in **fd2p*. The control process will return diagnostic output from the command (unit 2) on this channel, and will also accept bytes on this channel as signal numbers, to be forwarded to the process group of the command. If *fd2p* is 0, then the standard error (unit 2 of the remote command) will be made the same as its standard output and no provision is made for sending arbitrary signals to the remote process, although you may be able to get its attention by using out-of-band data.

SEE ALSO

`gethostent`(3N), `getservent`(3N), `rcmd`(3N), `rexecd`(1M).

NOTES

There is no way to specify options to the `socket()` call that `rexec()` makes.

NAME

dfstab – file containing commands for sharing resources

DESCRIPTION

dfstab resides in directory **/etc/dfs** and contains commands for sharing resources across a network. **dfstab** gives a system administrator a uniform method of controlling the automatic sharing of local resources.

Each line of the **dfstab** file consists of a **share**(1M) command. The **dfstab** file can be read by the shell directly to share all resources, or system administrators can prepare their own shell scripts to execute particular lines from **dfstab**.

The contents of **dfstab** are executed automatically when the system enters run level 3.

SEE ALSO

share(1M), **shareall**(1M).

NAME

`ethers` – Ethernet address to hostname database or domain

DESCRIPTION

The **ethers** file contains information regarding the known (48 bit) Ethernet addresses of hosts on the Internet. For each host on an Ethernet, a single line should be present with the following information:

Ethernet-address *official-host-name*

Items are separated by any number of SPACE and/or TAB characters. A '#' indicates the beginning of a comment extending to the end of line.

The standard form for Ethernet addresses is *x*:*x*:*x*:*x*:*x*:*x* where *x* is a hexadecimal number between 0 and ff, representing one byte. The address bytes are always in network order. Host names may contain any printable character other than a SPACE, TAB, NEWLINE, or comment character. It is intended that host names in the **ethers** file correspond to the host names in the **hosts**(4) file.

The `ether_line` routine from the Ethernet address manipulation library, **ethers**(3N) may be used to scan lines of the **ethers** file.

FILES

`/etc/ethers`

SEE ALSO

ethers(3N), **hosts**(4).

NAME

fstypes – file that registers distributed file system packages

DESCRIPTION

fstypes resides in directory **/etc/dfs** and lists distributed file system utilities packages installed on the system. The file system indicated in the first line of the file is the default file system. When Distributed File System (DFS) Administration commands are entered without the option **-F** *fstypes*, the system takes the file system type from the first line of the **fstypes** file.

The default package can be changed by editing the **fstypes** file with any supported text editor.

SEE ALSO

dfmounts(1M), **dfshares**(1M), **share**(1M), **shareall**(1M), **unshare**(1M).

NAME

hosts – host name data base

SYNOPSIS

/etc/hosts

DESCRIPTION

The hosts file contains information regarding the known hosts on the DARPA Internet. For each host a single line should be present with the following information:

Internet-address official-host-name aliases

Items are separated by any number of SPACE and/or TAB characters. A '#' indicates the beginning of a comment; characters up to the end of the line are not interpreted by routines which search the file. This file is normally created from the official host data base maintained at the Network Information Control Center (NIC), though local changes may be required to bring it up to date regarding unofficial aliases and/or unknown hosts.

Network addresses are specified in the conventional '.' notation using the inet_addr routine from the Internet address manipulation library, inet(3N). Host names may contain any printable character other than a field delimiter, NEWLINE, or comment character.

EXAMPLE

Here is a typical line from the /etc/hosts file:

```
192.9.1.20       gaia                          # John Smith
```

FILES

/etc/hosts

SEE ALSO

gethostent(3N), inet(3N).

NAME

hosts.equiv, .rhosts – trusted hosts by system and by user

DESCRIPTION

The /etc/hosts.equiv file contains a list of trusted hosts. When an rlogin(1) or rsh(1) request is received from a host listed in this file, and when the user making the request is listed in the /etc/passwd file, then the remote login is allowed with no further checking. The library routine **ruserok** (see rcmd(3N)) will make this verification. In this case, **rlogin** does not prompt for a password, and commands submitted through **rsh** are executed. Thus, a remote user with a local user ID is said to have equivalent access from a remote host named in this file.

The format of the hosts.equiv file consists of a one-line entry for each host, of the form:

hostname [*username*]

The *hostname* field normally contains the name of a trusted host from which a remote login can be made. However, an entry consisting of a single '+' indicates that all known hosts are to be trusted. A hostname must be the official name as listed in the hosts(4) database. This is the first name given in the hosts database entry; hostname aliases are not recognized.

The User .rhosts File

Whenever a remote login is attempted, the remote login daemon checks for a .rhosts file in the home directory of the user attempting to log in. A user's .rhosts file has the same format as the hosts.equiv file, and is used to give or deny access only for the *specific user* attempting to log in from a given host. While an entry in the hosts.equiv file allows remote login access to *any* user from the indicated host, an entry in a user's .rhosts file only allows access from a named host to the user in whose home directory the .rhosts file appears. When this file is used, permissions in the user's home directory should allow read and search access by anyone, so it may be located and read. When a user attempts a remote login, his .rhosts file is, in effect, prepended to the hosts.equiv file for permission checking. Thus, if a host is specified in the user's .rhosts file, login access is allowed.

FILES

```
/etc/hosts.equiv
/etc/passwd
~/.rhosts
/etc
```

SEE ALSO

rlogin(1), rsh(1), rcmd(3N), hosts(4), passwd(4).

NAME

`inetd.conf` – Internet servers database

DESCRIPTION

The **inetd.conf** file contains the list of servers that **inetd**(1M) invokes when it receives an Internet request over a socket. Each server entry is composed of a single line of the form:

service-name socket-type protocol wait-status uid server-program server-arguments

Fields can be separated by either SPACE or TAB characters. A '#' (pound-sign) indicates the beginning of a comment; characters up to the end of the line are not interpreted by routines that search this file.

service-name — The name of a valid service listed in the file **/etc/services**. For RPC services, the value of the *service-name* field consists of the RPC service name, followed by a slash and either a version number or a range of version numbers (for example, **mountd/1**).

socket-type — Can be one of:

stream	for a stream socket,
dgram	for a datagram socket,
raw	for a raw socket,
seqpacket	for a sequenced packet socket

protocol — Must be a recognized protocol listed in the file **/etc/protocols**. For RPC services, the field consists of the string rpc followed by a slash and the name of the protocol (for example, **rpc/udp** for an RPC service using the UDP protocol as a transport mechanism).

wait-status — **nowait** for all but single-threaded datagram servers — servers which do not release the socket until a timeout occurs (such as **comsat**(1M) and **talkd**(1M)). These must have the status **wait**. Although **tftpd**(1M) establishes separate pseudo-connections, its forking behavior can lead to a race condition unless it is also given the status **wait**.

uid — The user ID under which the server should run. This allows servers to run with access privileges other than those for root.

server-program — Either the pathname of a server program to be invoked by **inetd** to perform the requested service, or the value **internal** if **inetd** itself provides the service.

server-arguments — If a server must be invoked with command-line arguments, the entire command line (including argument 0) must appear in this field (which consists of all remaining words in the entry). If the server expects **inetd** to pass it the address of its peer (for compatibility with 4.2BSD executable daemons), then the first argument to the command should be specified as '**%A**'.

FILES

```
/etc/inetd.conf
/etc/services
/etc/protocols
```

SEE ALSO

rlogin(1), rsh(1), comsat(1M), inetd(1M), talkd(1M), tftpd(1M), services(4).

NAME

netmasks – network mask data base

DESCRIPTION

The **netmasks** file contains network masks used to implement IP standard subnetting. For each network that is subnetted, a single line should exist in this file with the network number, any number of SPACE or TAB characters, and the network mask to use on that network. Network numbers and masks may be specified in the conventional IP '.' notation (like IP host addresses, but with zeroes for the host part). For example,

```
128.32.0.0 255.255.255.0
```

can be used to specify that the Class B network 128.32.0.0 should have eight bits of subnet field and eight bits of host field, in addition to the standard sixteen bits in the network field.

FILES

`/etc/netmasks`

SEE ALSO

`ifconfig`(1M)

Postel, Jon, and Mogul, Jeff, *Internet Standard Subnetting Procedure*, RFC 950, Network Information Center, SRI International, Menlo Park, Calif., August 1985.

NAME

netrc – file for ftp remote login data

DESCRIPTION

The .netrc file contains data for logging in to a remote host over the network for file transfers by **ftp**(1). This file resides in the user's home directory on the machine initiating the file transfer. Its permissions should be set to disallow read access by group and others [see chmod(1)].

The following tokens are recognized; they may be separated by SPACE, TAB, or NEWLINE characters:

machine *name*
: Identify a remote machine name. The auto-login process searches the .netrc file for a **machine** token that matches the remote machine specified on the **ftp** command line or as an **open** command argument. Once a match is made, the subsequent .netrc tokens are processed, stopping when the EOF is reached or another **machine** token is encountered.

login *name*
: Identify a user on the remote machine. If this token is present, the auto-login process will initiate a login using the specified name.

password *string*
: Supply a password. If this token is present, the auto-login process will supply the specified string if the remote server requires a password as part of the login process. Note: if this token is present in the .netrc file, **ftp** will abort the auto-login process if the .netrc is readable by anyone besides the user.

account *string*
: Supply an additional account password. If this token is present, the auto-login process will supply the specified string if the remote server requires an additional account password, or the auto-login process will initiate an ACCT command if it does not.

macdef *name*
: Define a macro. This token functions as the **ftp macdef** command functions. A macro is defined with the specified name; its contents begin with the next .netrc line and continue until a NULL line (consecutive NEWLINE characters) is encountered. If a macro named init is defined, it is automatically executed as the last step in the auto-login process.

EXAMPLE

A .netrc file containing the following line:

```
machine ray login demo password mypassword
```

allows an autologin to the machine ray using the login name demo with password mypassword.

FILES

~/.netrc

SEE ALSO

chmod(1), ftp(1), ftpd(1M).

NAME

networks – network name data base

DESCRIPTION

The networks file contains information regarding the known networks which comprise the DARPA Internet. For each network a single line should be present with the following information:

official-network-name network-number aliases

Items are separated by any number of SPACE and/or TAB characters. A '#' indicates the beginning of a comment; characters up to the end of the line are not interpreted by routines which search the file. This file is normally created from the official network data base maintained at the Network Information Control Center (NIC), though local changes may be required to bring it up to date regarding unofficial aliases and/or unknown networks.

Network number may be specified in the conventional '.' notation using the inet_network routine from the Internet address manipulation library, inet(7). Network names may contain any printable character other than a field delimiter, NEWLINE, or comment character.

FILES

/etc/networks

SEE ALSO

getnetent(3N), inet(7).

NOTES

A name server should be used instead of a static file. A binary indexed file format should be available for fast access.

NAME

protocols – protocol name data base

SYNOPSIS

/etc/protocols

DESCRIPTION

The protocols file contains information regarding the known protocols used in the DARPA Internet. For each protocol a single line should be present with the following information:

official-protocol-name protocol-number aliases

Items are separated by any number of blanks and/or TAB characters. A '#' indicates the beginning of a comment; characters up to the end of the line are not interpreted by routines which search the file.

Protocol names may contain any printable character other than a field delimiter, NEWLINE, or comment character.

EXAMPLE

The following is a sample database:

```
#
# Internet (IP) protocols
#
ip      0       IP      # internet protocol, pseudo protocol number
icmp    1       ICMP    # internet control message protocol
ggp     3       GGP     # gateway-gateway protocol
tcp     6       TCP     # transmission control protocol
pup     12      PUP     # PARC universal packet protocol
udp     17      UDP     # user datagram protocol
```

FILES

/etc/protocols

SEE ALSO

getprotoent(3N)

NOTES

A name server should be used instead of a static file. A binary indexed file format should be available for fast access.

NAME

publickey − public key database

SYNOPSIS

/etc/publickey

DESCRIPTION

/etc/publickey is the public key database used for secure RPC. Each entry in the database consists of a network user name (which may either refer to a user or a hostname), followed by the user's public key (in hex notation), a colon, and then the user's secret key encrypted with a password (also in hex notation).

This file is altered either by the user through the **chkey**(1) command or by the system administrator through the **newkey**(1) command.

SEE ALSO

chkey(1), **newkey**(1), **publickey**(3N).

NAME

`resolv.conf` – configuration file for name server routines

DESCRIPTION

The resolver configuration file contains information that is read by the resolver routines the first time they are invoked in a process. The file is designed to be human readable and contains a list of keyword-value pairs that provide various types of resolver information.

keyword *value*

The different configuration options are:

`nameserver` *address*	The Internet address (in dot notation) of a name server that the resolver should query. At least one name server should be listed. Up to MAXNS (currently 3) name servers may be listed, in that case the resolver library queries tries them in the order listed. The algorithm used is to try a name server, and if the query times out, try the next, until out of name servers, then repeat trying all the name servers until a maximum number of retries are made.
`domain` *name*	The default domain to append to names that do not have a dot in them.
`address` *address*	An Internet address (in dot notation) of any preferred networks. The list of addresses returned by the resolver will be sorted to put any addresses on this network before any others.

The keyword-value pair must appear on a single line, and the keyword (for instance, `nameserver`) must start the line. The value follows the keyword, separated by white space.

FILES

`/etc/resolv.conf`

SEE ALSO

`named`(1M), `gethostent`(3N), `resolver`(3N).

NAME

`rfmaster` – Remote File Sharing name server master file

DESCRIPTION

Each transport provider used by Remote File Sharing has an associated `rfmaster` file that identifies the primary and secondary name servers for that transport provider. The `rfmaster` file ASCII contains a series of records, each terminated by a newline; a record may be extended over more than one line by escaping the newline character with a backslash ("\"). The fields in each record are separated by one or more tabs or spaces. Each record has three fields:

name type data

The *type* field, which defines the meaning of the *name* and *data* fields, has three possible values. These values can appear in upper case or lower case:

p The p type defines the primary domain name server. For this type, *name* is the domain name and *data* is the full host name of the machine that is the primary name server. The full host name is specified as *domain.nodename*. There can be only one primary name server per domain.

s The s type defines a secondary name server for a domain. *name* and *data* are the same as for the p type. The order of the s entries in the `rfmaster` file determines the order in which secondary name servers take over when the current domain name server fails.

a The a type defines a network address for a machine. *name* is the full domain name for the machine and *data* is the network address of the machine. The network address can be in plain ASCII text or it can be preceded by a `\x` or `\X` to be interpreted as hexadecimal notation. (See the documentation for the particular network you are using to determine the network addresses you need.)

If a line in the `rfmaster` file begins with a # character, the entire line is treated as a comment.

There are at least two lines in the `rfmaster` file per domain name server: one p and one a line, to define the primary and its network address.

This file is created and maintained on the primary domain name server. When a machine other than the primary tries to start Remote File Sharing, this file is read to determine the address of the primary. If the associated `rfmaster` for a transport provider is missing, use `rfstart -p` to identify the primary for that transport provider. After that, a copy of the primary's `rfmaster` file is automatically placed on the machine.

Domains not served by the primary can also be listed in the `rfmaster` file. By adding primary, secondary, and address information for other domains on a network, machines served by the primary will be able to share resources with machines in other domains.

A primary name server may be a primary for more than one domain. However, the secondaries must then also be the same for each domain served by the primary. There is an `rfmaster` file for each transport provider.

EXAMPLES

An example of an `rfmaster` file is shown below. (The network address examples, **`comp1.serve`** and **`comp2.serve`**, are STARLAN network addresses.)

```
ccs         p     ccs.comp1
ccs         s     ccs.comp2
ccs.comp2   a     comp2.serve
ccs.comp1   a     comp1.serve
```

FILES

`/etc/rfs/`*<transport>*`/rfmaster`

SEE ALSO

`rfstart`(1M) in the *System Administrator's Reference Manual.*

NAME

routing – system supporting for packet network routing

DESCRIPTION

The network facilities provide general packet routing. Routing table maintenance may be implemented in applications processes.

A simple set of data structures compose a routing table used in selecting the appropriate network interface when transmitting packets. This table contains a single entry for each route to a specific network or host. The routing table was designed to support routing for the Internet Protocol (IP), but its implementation is protocol independent and thus it may serve other protocols as well. User programs may manipulate this data base with the aid of two ioctl(2) commands, SIOCADDRT and SIOCDELRT. These commands allow the addition and deletion of a single routing table entry, respectively. Routing table manipulations may only be carried out by privileged user.

A routing table entry has the following form, as defined in /usr/include/net/route.h:

```
struct rtentry {
    u_long  rt_hash;                    /* to speed lookups */
    struct  sockaddr rt_dst;            /* key */
    struct  sockaddr rt_gateway;        /* value */
    short   rt_flags;                   /* up/down?, host/net */
    short   rt_refcnt;                  /* # held references */
    u_long  rt_use;                     /* raw # packets forwarded */
#ifdef STRNET
    struct  ip_provider *rt_prov;       /* the answer: provider to use */
#else
    struct  ifnet *rt_ifp;              /* the answer: interface to use */
#endif /* STRNET */
};
```

with *rt_flags* defined from:

```
#define  RTF_UP        0x1        /* route usable */
#define  RTF_GATEWAY   0x2        /* destination is a gateway */
#define  RTF_HOST      0x4        /* host entry (net otherwise) */
```

Routing table entries come in three flavors: for a specific host, for all hosts on a specific network, for any destination not matched by entries of the first two types (a wildcard route). Each network interface installs a routing table entry when it it is initialized. Normally the interface specifies the route through it is a direct connection to the destination host or network. If the route is direct, the transport layer of a protocol family usually requests the packet be sent to the same host specified in the packet. Otherwise, the interface may be requested to address the packet to an entity different from the eventual recipient (that is, the packet is forwarded).

Routing table entries installed by a user process may not specify the hash, reference count, use, or interface fields; these are filled in by the routing routines. If a route is in use when it is deleted (rt_refcnt is non-zero), the resources associated with it will not be reclaimed until all references to it are removed.

User processes read the routing tables through the `/dev/kmem` device.

The *rt_use* field contains the number of packets sent along the route. This value is used to select among multiple routes to the same destination. When multiple routes to the same destination exist, the least used route is selected.

A wildcard routing entry is specified with a zero destination address value. Wildcard routes are used only when the system fails to find a route to the destination host and network. The combination of wildcard routes and routing redirects can provide an economical mechanism for routing traffic.

FILES

`/dev/kmem`

SEE ALSO

`ioctl`(2), `route`(1M), `routed`(1M).

DIAGNOSTICS

`EEXIST`	A request was made to duplicate an existing entry.
`ESRCH`	A request was made to delete a non-existent entry.
`ENOBUFS`	Insufficient resources were available to install a new route.

NAME

`services` – Internet services and aliases

DESCRIPTION

The **services** file contains an entry for each service available through the DARPA Internet. Each entry consists of a line of the form:

service-name port / protocol aliases

service-name	This is the official Internet service name.
port / protocol	This field is composed of the port number and protocol through which the service is provided (for instance, `512/tcp`).
aliases	This is a list of alternate names by which the service might be requested.

Fields can be separated by any number of SPACE and/or TAB characters. A '#' (pound-sign) indicates the beginning of a comment; characters up to the end of the line are not interpreted by routines which search the file.

Service names may contain any printable character other than a field delimiter, NEWLINE, or comment character.

FILES

`/etc/services`

SEE ALSO

`getservent`(3N), `inetd.conf`(4).

NOTES

A name server should be used instead of a static file.

NAME

sharetab – shared file system table

DESCRIPTION

sharetab resides in directory **/etc/dfs** and contains a table of local resources shared by the **share** command.

Each line of the file consists of the following fields:

pathname resource fstype specific_options description

where

pathname	Indicates the pathname of the shared resource.
resource	Indicates the symbolic name by which remote systems can access the resource.
fstype	Indicates the file system type of the shared resource.
specific_options	Indicates file-system-type-specific options that were given to the **share** command when the resource was shared.
description	Is a description of the shared resource provided by the system administrator when the resource was shared.

SEE ALSO

share(1M)

NAME

strcf – STREAMS Configuration File for STREAMS TCP/IP

DESCRIPTION

/etc/strcf contains the script that is executed by slink(1M) to perform the STREAMS configuration operations required for STREAMS TCP/IP.

The standard /etc/strcf file contains several functions that perform various configuration operations, along with a sample boot function. Normally, only the boot function must be modified to customize the configuration for a given installation. In some cases, however, it may be necessary to change existing functions or add new functions.

The following functions perform basic linking operations:

The tp function is used to set up the link between a transport provider, such as TCP, and IP.

```
#
# tp - configure transport provider (i.e. tcp, udp, icmp)
# usage: tp devname
#
tp {
     p = open $1
     ip = open /dev/ip
     link p ip
}
```

The linkint function links the specified streams and does a sifname operation with the given name.

```
#
# linkint - link interface to ip or arp
# usage: linkint top bottom ifname
#
linkint {
     x = link $1 $2
     sifname $1 x $3
}
```

The aplinkint function performs the same function as linkint for an interface that uses the app module.

```
#
# aplinkint - like linkint, but app is pushed on dev
# usage: aplinkint top bottom ifname
#
aplinkint {
     push $2 app
     linkint $1 $2 $3
}
```

The following functions are used to configure different types of Ethernet interfaces:

The uenet function is used to configure an Ethernet interface for a cloning device driver that uses the *unit select* ioctl to select the desired interface. The interface name is constructed by concatenating the supplied prefix and the unit number.

```
#
# uenet - configure ethernet-type interface for cloning driver using
#         unit select
# usage: uenet ip-fd devname ifprefix unit
#
uenet {
     ifname = strcat $3 $4
     dev = open $2
     unitsel dev $4
     aplinkint $1 dev ifname
     dev = open $2
     unitsel dev $4
     arp = open /dev/arp
     linkint arp dev ifname
}
```

The denet function performs the same function as uenet, except that DL_ATTACH is used instead of *unit select*.

```
#
# denet - configure ethernet-type interface for cloning driver using
#         DL_ATTACH
# usage: denet ip-fd devname ifprefix unit
#
denet {
     ifname = strcat $3 $4
     dev = open $2
     dlattach dev $4
     aplinkint $1 dev ifname
     dev = open $2
     dlattach dev $4
     arp = open /dev/arp
     linkint arp dev ifname
}
```

The cenet function is used to configure an Ethernet interface for a cloning device driver that uses a different major number for each interface. The device name is formed by concatenating the supplied device name prefix and the unit number. The interface name is formed in a similar manner using the interface name prefix.

```
#
# cenet - configure ethernet-type interface for cloning driver with
#         one major per interface
# usage: cenet ip-fd devprefix ifprefix unit
#
cenet {
```

```
          devname = strcat $2 $4
          ifname = strcat $3 $4
          dev = open devname
          aplinkint $1 dev ifname
          dev = open devname
          arp = open /dev/arp
          linkint arp dev ifname
     }
```

The senet function is used to configure an Ethernet interface for a non-cloning device driver. Two different device nodes must be specified for IP and ARP.

```
     #
     # senet - configure ethernet-type interface for non-cloning driver
     # usage: senet ip-fd ipdevname arpdevname ifname
     #
     senet {
          dev = open $2
          aplinkint $1 dev $4
          dev = open $3
          arp = open /dev/arp
          linkint arp dev $4
     }
```

The senetc function is like senet, except that it allows the specification of a convergence module to be used with the ethernet driver (such as, for the 3B2 emd driver).

```
     #
     # senetc - configure ethernet-type interface for non-cloning driver
     #          using convergence module
     # usage: senetc ip-fd convergence ipdevname arpdevname ifname
     #
     senetc {
          dev = open $3
          push dev $2
          aplinkint $1 dev $5
          dev = open $4
          push dev $2
          arp = open /dev/arp
          linkint arp dev $5
     }
```

The loopback function is used to configure the loopback interface.

```
     #
     # loopback - configure loopback device
     # usage: loopback ip-fd
     #
     loopback {
          dev = open /dev/loop
          linkint $1 dev lo0
     }
```

The slip function is used to configure a SLIP interface. This function is not normally executed at boot time. Rather, the slattach(1M) command runs slink specifying slip on the command line.

```
#
# slip - configure slip interface
# usage: slip unit
#
slip {
	ip = open /dev/ip
	s = open /dev/slip
	ifname = strcat sl $1
	unitsel s $1
	linkint ip s ifname
}
```

The boot function is called by default when slink is executed. Normally, only the *interfaces* section and possibly the *queue params* section will have to be customized for a given installation. Examples are provided for the various Ethernet driver types.

```
#
# boot - boot time configuration
#
boot {
	#
	# queue params
	#
	initqp /dev/udp rq 8192 40960
	initqp /dev/ip muxrq 8192 40960 rq 8192 40960
	#
	# transport
	#
	tp /dev/tcp
	tp /dev/udp
	tp /dev/icmp
	tp /dev/rawip
}
```

FILES

/etc/strcf

SEE ALSO

slattach(1M), slink(1M).

NAME

ARP – Address Resolution Protocol

SYNOPSIS

```
#include <sys/socket.h>
#include <net/if_arp.h>
#include <netinet/in.h>

s = socket(AF_INET, SOCK_DGRAM, 0);

d = open ("/dev/arp", O_RDWR);
```

DESCRIPTION

ARP is a protocol used to map dynamically between Internet Protocol (IP) and 10Mb/s Ethernet addresses. It is used by all the 10Mb/s Ethernet datalink providers (interface drivers). It is not specific to the Internet Protocol or to the 10Mb/s Ethernet, but this implementation currently supports only that combination. The STREAMS device /dev/arp is not a Transport Level Interface (TLI) transport provider and may not be used with the TLI interface.

ARP caches IP-to-Ethernet address mappings. When an interface requests a mapping for an address not in the cache, ARP queues the message that requires the mapping and broadcasts a message on the associated network requesting the address mapping. If a response is provided, the new mapping is cached and any pending message is transmitted. ARP will queue at most one packet while waiting for a mapping request to be responded to; only the most recently transmitted packet is kept.

To facilitate communications with systems which do not use ARP, `ioctl()` requests are provided to enter and delete entries in the IP-to-Ethernet tables.

USAGE

```
#include <sys/sockio.h>
#include <sys/socket.h>
#include <net/if.h>
#include <net/if_arp.h>
struct arpreq arpreq;
ioctl(s, SIOCSARP, (caddr_t)&arpreq);
ioctl(s, SIOCGARP, (caddr_t)&arpreq);
ioctl(s, SIOCDARP, (caddr_t)&arpreq);
```

Each `ioctl()` request takes the same structure as an argument. `SIOCSARP` sets an ARP entry, `SIOCGARP` gets an ARP entry, and `SIOCDARP` deletes an ARP entry. These `ioctl()` requests may be applied to any Internet family socket descriptor *s*, or to a descriptor for the ARP device, but only by the privileged user. The arpreq structure contains:

```
/*
* ARP ioctl request
*/
struct arpreq {
     struct sockaddr arp_pa;         /* protocol address */
     struct sockaddr arp_ha;         /* hardware address */
     int  arp_flags;      /* flags */
};
```

```
/*  arp_flags field values */
#define ATF_COM             0x2   /* completed entry (arp_ha valid) */
#define ATF_PERM            0x4   /* permanent entry */
#define ATF_PUBL            0x8   /* publish (respond for other host) */
#define ATF_USETRAILERS     0x10  /* send trailer packets to host */
```

The address family for the **arp_pa sockaddr** must be **AF_INET**; for the **arp_ha sockaddr** it must be **AF_UNSPEC**. The only flag bits that may be written are **ATF_PERM**, **ATF_PUBL** and **ATF_USETRAILERS**. **ATF_PERM** makes the entry permanent if the `ioctl()` request succeeds. The peculiar nature of the ARP tables may cause the `ioctl()` request to fail if too many permanent IP addresses hash to the same slot. **ATF_PUBL** specifies that the ARP code should respond to ARP requests for the indicated host coming from other machines. This allows a host to act as an ARP server, which may be useful in convincing an ARP-only machine to talk to a non-ARP machine.

ARP is also used to negotiate the use of trailer IP encapsulations; trailers are an alternate encapsulation used to allow efficient packet alignment for large packets despite variable-sized headers. Hosts that wish to receive trailer encapsulations so indicate by sending gratuitous ARP translation replies along with replies to IP requests; they are also sent in reply to IP translation replies. The negotiation is thus fully symmetrical, in that either or both hosts may request trailers. The **ATF_USETRAILERS** flag is used to record the receipt of such a reply, and enables the transmission of trailer packets to that host.

ARP watches passively for hosts impersonating the local host (that is, a host which responds to an ARP mapping request for the local host's address).

SEE ALSO

`arp`(1M), `ifconfig`(1M), `if`(3N), `inet`(7).

Plummer, Dave, "*An Ethernet Address Resolution Protocol -or- Converting Network Protocol Addresses to 48.bit Ethernet Addresses for Transmission on Ethernet Hardware*," RFC 826, Network Information Center, SRI International, Menlo Park, Calif., November 1982.

Leffler, Sam, and Michael Karels, "*Trailer Encapsulations*," RFC 893, Network Information Center, SRI International, Menlo Park, Calif., April 1984.

NAME

ICMP – Internet Control Message Protocol

SYNOPSIS

```
#include <sys/socket.h>
#include <netinet/in.h>
#include <netinet/ip_icmp.h>

s = socket(AF_INET, SOCK_RAW, proto);

t = t_open("/dev/icmp", O_RDWR);
```

DESCRIPTION

ICMP is the error and control message protocol used by the Internet protocol family. It is used by the kernel to handle and report errors in protocol processing. It may also be accessed by programs using the socket interface or the Transport Level Interface (TLI) for network monitoring and diagnostic functions. When used with the socket interface, a raw socket type is used. The protocol number for ICMP, used in the *proto* parameter to the socket call, can be obtained from `getprotobyname()` [see `getprotoent(3N)`]. ICMP file descriptors and sockets are connectionless, and are normally used with the `t_sndudata` / `t_rcvudata` and the `sendto()` / `recvfrom()` calls.

Outgoing packets automatically have an Internet Protocol (IP) header prepended to them. Incoming packets are provided to the user with the IP header and options intact.

ICMP is an datagram protocol layered above IP. It is used internally by the protcol code for various purposes including routing, fault isolation, and congestion control. Receipt of an ICMP redirect message will add a new entry in the routing table, or modify an existing one. ICMP messages are routinely sent by the protocol code. Received ICMP messages may be reflected back to users of higher-level protocols such as TCP or UDP as error returns from system calls. A copy of all ICMP message received by the system is provided to every holder of an open ICMP socket or TLI descriptor.

SEE ALSO

`send(2)`, `getprotoent(3N)`, `recvfrom(3N)`, `t_rcvudata(3N)`, `t_sndudata(3N)`, `routing(4)`, `inet(7)`, `ip(7)`.

Postel, Jon, *Internet Control Message Protocol — DARPA Internet Program Protocol Specification*, RFC 792, Network Information Center, SRI International, Menlo Park, Calif., September 1981.

DIAGNOSTICS

A socket operation may fail with one of the following errors returned:

`EISCONN`	An attempt was made to establish a connection on a socket which already has one, or when trying to send a datagram with the destination address specified and the socket is already connected.
`ENOTCONN`	An attempt was made to send a datagram, but no destination address is specified, and the socket has not been connected.

ENOBUFS	The system ran out of memory for an internal data structure.
EADDRNOTAVAIL	An attempt was made to create a socket with a network address for which no network interface exists.

NOTES

Replies to ICMP echo messages which are source routed are not sent back using inverted source routes, but rather go back through the normal routing mechanisms.

NAME

`if` – general properties of Internet Protocol network interfaces

DESCRIPTION

A network interface is a device for sending and receiving packets on a network. A network interface is usually a hardware device, although certain interfaces such as the loopback interface, `lo`(7), are implemented in software. Network interfaces used by the Internet Protocol (IP) must be STREAMS devices conforming to the Datalink Provider Interface (DLPI).

An interface becomes available to IP when it is linked below the IP STREAMS device with the `I_LINK ioctl()` call. This may be initiated by the kernel at boot time or by a user program some time after the system is running. Each IP interface must have a name assigned to it with the `SIOCSIFNAME ioctl()`. This name is used as a unique handle on the interface by all of the other network interface `ioctl()` calls. Each interface must be assigned an IP address with the `SIOCSIFADDR ioctl()` before it can be used. On interfaces where the network-to-link layer address mapping is static, only the network number is taken from the `ioctl()` request; the remainder is found in a hardware specific manner. On interfaces which provide dynamic network-to-link layer address mapping facilities [for example, 10Mb/s Ethernets using `arp`(7)], the entire address specified in the `ioctl()` is used. A routing table entry for destinations on the network of the interface is installed automatically when an interface's address is set.

IOCTLS

The following `ioctl()` calls may be used to manipulate IP network interfaces. Unless specified otherwise, the request takes an `ifreq` structure as its parameter. This structure has the form:

```
/* Interface request structure used for socket ioctl's.  All */
/* interface ioctl's must have parameter definitions which */
/* begin with ifr_name.  The remainder may be interface specific. */

struct  ifreq {
#define IFNAMSIZ     16
    char         ifr_name[IFNAMSIZ];         /* if name, e.g. "emdl" */
    union {
         struct  sockaddr ifru_addr;
         struct  sockaddr ifru_dstaddr;
         char    ifru_oname[IFNAMSIZ];       /* other if name */
         struct  sockaddr ifru_broadaddr;
         short   ifru_flags;
         int     ifru_metric;
         char    ifru_data[1];               /* interface dependent data */
         char    ifru_enaddr[6];
    } ifr_ifru;
#define ifr_addr      ifr_ifru.ifru_addr      /* address */
#define ifr_dstaddr   ifr_ifru.ifru_dstaddr   /* other end of p-to-p link */
#define ifr_oname     ifr_ifru.ifru_oname     /* other if name */
#define ifr_broadaddr ifr_ifru.ifru_broadaddr /* broadcast address */
#define ifr_flags     ifr_ifru.ifru_flags     /* flags */
#define ifr_metric    ifr_ifru.ifru_metric    /* metric */
#define ifr_data      ifr_ifru.ifru_data      /* for use by interface */
#define ifr_enaddr    ifr_ifru.ifru_enaddr    /* ethernet address */
};
```

SIOCSIFADDR	Set interface address. Following the address assignment, the initialization routine for the interface is called.
SIOCGIFADDR	Get interface address.
SIOCSIFDSTADDR	Set point to point address for interface.
SIOCGIFDSTADDR	Get point to point address for interface.
SIOCSIFFLAGS	Set interface flags field. If the interface is marked down, any processes currently routing packets through the interface are notified.
SIOCGIFFLAGS	Get interface flags.
SIOCGIFCONF	Get interface configuration list. This request takes an `ifconf` structure (see below) as a value-result parameter. The `ifc_len` field should be initially set to the size of the buffer pointed to by `ifc_buf`. On return it will contain the length, in bytes, of the configuration list.

The `ifconf` structure has the form:

```
/*
 * Structure used in SIOCGIFCONF request.
 * Used to retrieve interface configuration
 * for machine (useful for programs which
 * must know all networks accessible).
 */
structifconf {
        int     ifc_len;            /* size of associated buffer */
        union {
                caddr_t ifcu_buf;
                struct  ifreq *ifcu_req;
        } ifc_ifcu;
#define ifc_buf ifc_ifcu.ifcu_buf   /* buffer address */
#define ifc_req ifc_ifcu.ifcu_req   /* array of structures returned */
};
```

SIOCSIFNAME
Set the name of the interface.

SEE ALSO

arp(7), ip(7), lo(7).

NAME

inet – Internet protocol family

SYNOPSIS

```
#include <sys/types.h>
#include <netinet/in.h>
```

DESCRIPTION

The Internet protocol family implements a collection of protocols which are centered around the *Internet Protocol* (IP) and which share a common address format. The Internet family protocols can be accessed via the socket interface, where they support the SOCK_STREAM, SOCK_DGRAM, and SOCK_RAW socket types, or the Transport Level Interface (TLI), where they support the connectionless (T_CLTS) and connection oriented (T_COTS_ORD) service types.

PROTOCOLS

The Internet protocol family comprises the Internet Protocol (IP), the Address Resolution Protocol (ARP), the Internet Control Message Protocol (ICMP), the Transmission Control Protocol (TCP), and the User Datagram Protocol (UDP).

TCP supports the socket interface's SOCK_STREAM abstraction and TLI's T_COTS_ORD service type. UDP supports the SOCK_DGRAM socket abstraction and the TLI T_CLTS service type. See tcp(7) and udp(7). A direct interface to IP is available via both TLI and the socket interface; See ip(7). ICMP is used by the kernel to handle and report errors in protocol processing. It is also accessible to user programs; see icmp(7). ARP is used to translate 32-bit IP addresses into 48-bit Ethernet addresses; see arp(7).

The 32-bit IP address is divided into network number and host number parts. It is frequency-encoded; The most-significant bit is zero in Class A addresses, in which the high-order 8 bits represent the network number. Class B addresses have their high order two bits set to 10 and use the high-order 16 bits as the network number field. Class C addresses have a 24-bit network number part of which the high order three bits are 110. Sites with a cluster of IP networks may chose to use a single network number for the cluster; This is done by using subnet addressing. The host number portion of the address is further subdivided into subnet number and host number parts. Within a subnet, each subnet appears to be an individual network; Externally, the entire cluster appears to be a single, uniform network requiring only a single routing entry. Subnet addressing is enabled and examined by the following ioctl(2) commands; They have the same form as the SIOCSIFADDR command [see if(3N)].

SIOCSIFNETMASK	Set interface network mask. The network mask defines the network part of the address; If it contains more of the address than the address type would indicate, then subnets are in use.
SIOCGIFNETMASK	Get interface network mask.

ADDRESSING

IP addresses are four byte quantities, stored in network byte order. IP addresses should be manipulated using the byte order conversion routines [see byteorder(3N)].

Addresses in the Internet protocol family use the following structure:

```
struct sockaddr_in {
     short    sin_family;
     u_short  sin_port;
     struct   in_addr sin_addr;
     char     sin_zero[8];
};
```

Library routines are provided to manipulate structures of this form; See inet(3N).

The sin_addr field of the sockaddr_in structure specifies a local or remote IP address. Each network interface has its own unique IP address. The special value INADDR_ANY may be used in this field to effect wildcard matching. Given in a bind(2) call, this value leaves the local IP address of the socket unspecified, so that the socket will receive connections or messages directed at any of the valid IP addresses of the system. This can prove useful when a process neither knows nor cares what the local IP address is or when a process wishes to receive requests using all of its network interfaces. The sockaddr_in structure given in the bind(2) call must specify an in_addr value of either IPADDR_ANY or one of the system's valid IP addresses. Requests to bind any other address will elicit the error EADDRNOTAVAI. When a connect(2) call is made for a socket that has a wildcard local address, the system sets the sin_addr field of the socket to the IP address of the network interface that the packets for that connection are routed via.

The sin_port field of the sockaddr_in structure specifies a port number used by TCP or UDP. The local port address specified in a bind(2) call is restricted to be greater than IPPORT_RESERVED (defined in <netinet/in.h>) unless the creating process is running as the super-user, providing a space of protected port numbers. In addition, the local port address must not be in use by any socket of same address family and type. Requests to bind sockets to port numbers being used by other sockets return the error EADDRINUSE. If the local port address is specified as 0, then the system picks a unique port address greater than IPPORT_RESERVED. A unique local port address is also picked when a socket which is not bound is used in a connect(2) or sendto [see send(2)] call. This allows programs which do not care which local port number is used to set up TCP connections by simply calling socket(2) and then connect(2), and to send UDP datagrams with a socket(2) call followed by a sendto(2) call.

Although this implementation restricts sockets to unique local port numbers, TCP allows multiple simultaneous connections involving the same local port number so long as the remote IP addresses or port numbers are different for each connection. Programs may explicitly override the socket restriction by setting the SO_REUSEADDR socket option with setsockopt [see getsockopt(3N)].

TLI applies somewhat different semantics to the binding of local port numbers. These semantics apply when Internet family protocols are used via the TLI.

SEE ALSO

`ioctl`(2), `send`(2), `bind`(3N), `connect`(3N), `getsockopt`(3N), `if`(3N), `byteorder`(3N), `gethostent`(3N), `getnetent`(3N), `getprotoent`(3N), `getservent`(3N), `socket`(3N), `arp`(7), `icmp`(7), `ip`(7), `tcp`(7), `udp`(7).

Network Information Center, *DDN Protocol Handbook* (3 vols.), Network Information Center, SRI International, Menlo Park, Calif., 1985.

NOTES

The Internet protocol support is subject to change as the Internet protocols develop. Users should not depend on details of the current implementation, but rather the services exported.

NAME

IP – Internet Protocol

SYNOPSIS

```
#include <sys/socket.h>
#include <netinet/in.h>

s = socket(AF_INET, SOCK_RAW, proto);

t = t_open ("/dev/rawip", O_RDWR);

d = open ("/dev/ip", O_RDWR);
```

DESCRIPTION

IP is the internetwork datagram delivery protocol that is central to the Internet protocol family. Programs may use IP through higher-level protocols such as the Transmission Control Protocol (TCP) or the User Datagram Protocol (UDP), or may interface directly to IP. See tcp(7) and udp(7). Direct access may be via the socket interface (using a raw socket) or the Transport Level Interface (TLI). The protocol options defined in the IP specification may be set in outgoing datagrams.

The STREAMS driver /dev/rawip is the TLI transport provider that provides raw access to IP. The device /dev/ip is the multiplexing STREAMS driver that implements the protocol processing of IP. The latter connects below to datalink providers [interface drivers, see if(3N)], and above to tranport providers such as TCP and UDP.

Raw IP sockets are connectionless and are normally used with the sendto() and recvfrom() calls, [(see send(2) and recv(2)] although the connect(2) call may also be used to fix the destination for future datagrams [in which case the read(2) or recv(2) and write(2) or send(2) calls may be used]. If proto is zero, the default protocol, IPPROTO_RAW, is used. If proto is non-zero, that protocol number will be set in outgoing datagrams and will be used to filter incoming datagrams. An IP header will be generated and prepended to each outgoing datagram; received datagrams are returned with the IP header and options intact.

A single socket option, IP_OPTIONS, is supported at the IP level. This socket option may be used to set IP options to be included in each outgoing datagram. IP options to be sent are set with setsockopt() [see getsockopt(2)]. The getsockopt(2) call returns the IP options set in the last setsockopt() call. IP options on received datagrams are visible to user programs only using raw IP sockets. The format of IP options given in setsockopt() matches those defined in the IP specification with one exception: the list of addresses for the source routing options must include the first-hop gateway at the beginning of the list of gateways. The first-hop gateway address will be extracted from the option list and the size adjusted accordingly before use. IP options may be used with any socket type in the Internet family.

At the socket level, the socket option SO_DONTROUTE may be applied. This option forces datagrams being sent to bypass the routing step in output. Normally, IP selects a network interface to send the datagram, and possibly an intermediate gateway, based on an entry in the routing table. See routing(4). When SO_DONTROUTE is set, the datagram will be sent using the interface whose network number or full IP address matches the destination address. If no interface matches, the error ENETUNRCH will be returned.

Raw IP datagrams can also be sent and received using the TLI connectionless primitives.

Datagrams flow through the IP layer in two directions: from the network *up* to user processes and from user processes *down* to the network. Using this orientation, IP is layered *above* the network interface drivers and *below* the transport protocols such as UDP and TCP. The Internet Control Message Protocol (ICMP) is logically a part of IP. See `icmp(7)`.

IP provides for a checksum of the header part, but not the data part of the datagram. The checksum value is computed and set in the process of sending datagrams and checked when receiving datagrams. IP header checksumming may be disabled for debugging purposes by patching the kernel variable **`ipcksum`** to have the value zero.

IP options in received datagrams are processed in the IP layer according to the protocol specification. Currently recognized IP options include: security, loose source and record route (LSRR), strict source and record route (SSRR), record route, stream identifier, and internet timestamp.

The IP layer will normally forward received datagrams that are not addressed to it. Forwarding is under the control of the kernel variable *ipforwarding*: if *ipforwarding* is zero, IP datagrams will not be forwarded; if *ipforwarding* is one, IP datagrams will be forwarded. *ipforwarding* is usually set to one only in machines with more than one network interface (internetwork routers). This kernel variable can be patched to enable or disable forwarding.

The IP layer will send an ICMP message back to the source host in many cases when it receives a datagram that can not be handled. A time exceeded ICMP message will be sent if the time to live field in the IP header drops to zero in the process of forwarding a datagram. A destination unreachable message will be sent if a datagram can not be forwarded because there is no route to the final destination, or if it can not be fragmented. If the datagram is addressed to the local host but is destined for a protocol that is not supported or a port that is not in use, a destination unreachable message will also be sent. The IP layer may send an ICMP source quench message if it is receiving datagrams too quickly. ICMP messages are only sent for the first fragment of a fragmented datagram and are never returned in response to errors in other ICMP messages.

The IP layer supports fragmentation and reassembly. Datagrams are fragmented on output if the datagram is larger than the maximum transmission unit (MTU) of the network interface. Fragments of received datagrams are dropped from the reassembly queues if the complete datagram is not reconstructed within a short time period.

Errors in sending discovered at the network interface driver layer are passed by IP back up to the user process.

SEE ALSO

`read(2)`, `write(2)`, `connect(3N)`, `getsockopt(3N)`, `recv(3N)`, `send(3N)`, `routing(4)`, `icmp(7)`, `inet(7)` `tcp(7)`, `udp(7)`.

Postel, Jon, *Internet Protocol - DARPA Internet Program Protocol Specification,* RFC 791, Network Information Center, SRI International, Menlo Park, Calif., September 1981.

DIAGNOSTICS

A socket operation may fail with one of the following errors returned:

EACCESS	A IP broadcast destination address was specified and the caller was not the privileged user.
EISCONN	An attempt was made to establish a connection on a socket which already had one, or to send a datagram with the destination address specified and the socket was already connected.
EMSGSIZE	An attempt was made to send a datagram that was too large for an interface, but was not allowed to be fragmented (such as broadcasts).
ENETUNREACH	An attempt was made to establish a connection or send a datagram, where there was no matching entry in the routing table, or if an ICMP destination unreachable message was received.
ENOTCONN	A datagrem was sent, but no destination address was specified, and the socket had not been connected.
ENOBUFS	The system ran out of memory for fragmentation buffers or other internal data structure.
EADDRNOTAVAIL	An attempt was made to create a socket with a local address that did not match any network interface, or an IP broadcast destination address was specified and the network interface does not support broadcast.

The following errors may occur when setting or getting IP options:

EINVAL	An unknown socket option name was given.
EINVAL	The IP option field was improperly formed; an option field was shorter than the minimum value or longer than the option buffer provided.

NOTES

Raw sockets should receive ICMP error packets relating to the protocol; currently such packets are simply discarded.

Users of higher-level protocols such as TCP and UDP should be able to see received IP options.

NAME

lo – software loopback network interface

SYNOPSIS

```
d = open ("/dev/loop", O_RDWR);
```

DESCRIPTION

The loopback device is a software datalink provider (interface driver) that returns all packets it receives to their source without involving any hardware devices. It is a STREAMS device conforming to the datalink provider interface (DLPI). See if(7) for a general description of network interfaces.

The loopback interface is used to access Internet services on the local machine. Because it is available on all machines, including those with no hardware network interfaces, programs can use it for guaranteed access to local servers. A typical application is the comsat(1M) server which accepts notification of mail delivery from a local client. The loopback interface is also used for performance analysis and testing.

By convention, the name of the loopback interface is lo0, and it is configured with Internet address 127.0.0.1. This address may be changed with the SIOCSIFADDR ioctl().

SEE ALSO

comsat(1M), if(7), inet(7).

NAME

TCP – Internet Transmission Control Protocol

SYNOPSIS

```
#include <sys/socket.h>
#include <netinet/in.h>

s = socket(AF_INET, SOCK_STREAM, 0);

t = t_open("/dev/tcp", O_RDWR);
```

DESCRIPTION

TCP is the virtual circuit protocol of the Internet protocol family. It provides reliable, flow-controlled, in order, two-way transmission of data. It is a byte-stream protocol layered above the Internet Protocol (IP), the Internet protocol family's internetwork datagram delivery protocol.

Programs can access TCP using the socket interface as a `SOCK_STREAM` socket type, or using the Transport Level Interface (TLI) where it supports the connection-oriented (`T_COTS_ORD`) service type.

TCP uses IP's host-level addressing and adds its own per-host collection of port addresses. The endpoints of a TCP connection are identified by the combination of an IP address and a TCP port number. Although other protocols, such as the User Datagram Protocol (UDP), may use the same host and port address format, the port space of these protocols is distinct. See `inet`(7) for details on the common aspects of addressing in the Internet protocol family.

Sockets utilizing TCP are either active or passive. Active sockets initiate connections to passive sockets. Both types of sockets must have their local IP address and TCP port number bound with the `bind`(2) system call after the socket is created. By default, TCP sockets are active. A passive socket is created by calling the `listen`(2) system call after binding the socket with `bind()`. This establishes a queueing parameter for the passive socket. After this, connections to the passive socket can be received with the `accept`(2) system call. Active sockets use the `connect`(2) call after binding to initiate connections.

By using the special value `INADDR_ANY`, the local IP address can be left unspecified in the `bind()` call by either active or passive TCP sockets. This feature is usually used if the local address is either unknown or irrelevant. If left unspecified, the local IP address will be bound at connection time to the address of the network interface used to service the connection.

Once a connection has been established, data can be exchanged using the `read`(2) and `write`(2) system calls.

TCP supports one socket option which is set with `setsockopt()` and tested with `getsockopt`(2). Under most circumstances, TCP sends data when it is presented. When outstanding data has not yet been acknowledged, it gathers small amounts of output to be sent in a single packet once an acknowledgement is received. For a small number of clients, such as window systems that send a stream of mouse events which receive no replies, this packetization may cause significant delays. Therefore, TCP provides a boolean option, `TCP_NODELAY` (defined in `/usr/include/netinet/tcp.h`), to defeat this algorithm. The option level for the `setsockopt()` call is the protocol number for TCP, available from `getprotobyname()` [see `getprotoent`(3N)].

Options at the IP level may be used with TCP; See `ip`(7).

TCP provides an urgent data mechanism, which may be invoked using the out-of-band provisions of `send`(2). The caller may mark one byte as urgent with the `MSG_OOB` flag to `send`(2). This sets an urgent pointer pointing to this byte in the TCP stream. The receiver on the other side of the stream is notified of the urgent data by a `SIGURG` signal. The `SIOCATMARK` `ioctl()` request returns a value indicating whether the stream is at the urgent mark. Because the system never returns data across the urgent mark in a single `read`(2) call, it is possible to advance to the urgent data in a simple loop which reads data, testing the socket with the `SIOCATMARK` `ioctl()` request, until it reaches the mark.

Incoming connection requests that include an IP source route option are noted, and the reverse source route is used in responding.

A checksum over all data helps TCP implement reliability. Using a window-based flow control mechanism that makes use of positive acknowledgements, sequence numbers, and a retransmission strategy, TCP can usually recover when datagrams are damaged, delayed, duplicated or delivered out of order by the underlying communication medium.

If the local TCP receives no acknowledgements from its peer for a period of time, as would be the case if the remote machine crashed, the connection is closed and an error is returned to the user. If the remote machine reboots or otherwise loses state information about a TCP connection, the connection is aborted and an error is returned to the user.

SEE ALSO

`read`(2), `write`(2), `accept`(3N), `bind`(3N), `connect`(3N), `getprotoent`(3N), `getsockopt`(3N), `listen`(3N), `send`(3N), `inet`(7), `ip`(7).

Postel, Jon, *Transmission Control Protocol - DARPA Internet Program Protocol Specification*, RFC 793, Network Information Center, SRI International, Menlo Park, Calif., September 1981.

DIAGNOSTICS

A socket operation may fail if:

`EISCONN`	A `connect()` operation was attempted on a socket on which a `connect()` operation had already been performed.
`ETIMEDOUT`	A connection was dropped due to excessive retransmissions.
`ECONNRESET`	The remote peer forced the connection to be closed (usually because the remote machine has lost state information about the connection due to a crash).
`ECONNREFUSED`	The remote peer actively refused connection establishment (usually because no process is listening to the port).

EADDRINUSE	A `bind()` operation was attempted on a socket with a network address/port pair that has already been bound to another socket.
EADDRNOTAVAIL	A `bind()` operation was attempted on a socket with a network address for which no network interface exists.
EACCES	A `bind()` operation was attempted with a reserved port number and the effective user ID of the process was not the privileged user.
ENOBUFS	The system ran out of memory for internal data structures.

NAME

UDP – Internet User Datagram Protocol

SYNOPSIS

```
#include <sys/socket.h>
#include <netinet/in.h>

s = socket(AF_INET, SOCK_DGRAM, 0);

t = t_open("/dev/udp", O_RDWR);
```

DESCRIPTION

UDP is a simple datagram protocol which is layered directly above the Internet Protocol (IP). Programs may access UDP using the socket interface, where it supports the `SOCK_DGRAM` socket type, or using the Transport Level Interface (TLI), where it supports the connectionless (`T_CLTS`) service type.

Within the socket interface, UDP is normally used with the `sendto()`, `sendmsg()`, `recvfrom()`, and `recvmsg()` calls [see `send`(2) and `recv`(2)]. If the `connect`(2) call is used to fix the destination for future packets, then the `recv`(2) or `read`(2) and `send`(2) or `write`(2) calls may be used.

UDP address formats are identical to those used by the Transmission Control Protocol (TCP). Like TCP, UDP uses a port number along with an IP address to identify the endpoint of communication. The UDP port number space is separate from the TCP port number space (that is, a UDP port may not be connected to a TCP port). The `bind`(2) call can be used to set the local address and port number of a UDP socket. The local IP address may be left unspecified in the `bind()` call by using the special value `INADDR_ANY`. If the `bind()` call is not done, a local IP address and port number will be assigned to the endpoint when the first packet is sent. Broadcast packets may be sent (assuming the underlying network supports this) by using a reserved broadcast address; This address is network interface dependent. Broadcasts may only be sent by the privileged user.

Options at the IP level may be used with UDP; see `ip`(7).

There are a variety of ways that a UDP packet can be lost or corrupted, including a failure of the underlying communication mechanism. UDP implements a checksum over the data portion of the packet. If the checksum of a received packet is in error, the packet will be dropped with no indication given to the user. A queue of received packets is provided for each UDP socket. This queue has a limited capacity. Arriving datagrams which will not fit within its *high-water* capacity are silently discarded.

UDP processes Internet Control Message Protocol (ICMP) error messages received in response to UDP packets it has sent. See `icmp`(7). ICMP source quench messages are ignored. ICMP destination unreachable, time exceeded and parameter problem messages disconnect the socket from its peer so that subsequent attempts to send packets using that socket will return an error. UDP will not guarantee that packets are delivered in the order they were sent. As well, duplicate packets may be generated in the communication process.

SEE ALSO

read(2), write(2), bind(3N), connect(3N), recv(3N), send(3N), icmp(7), inet(7), ip(7), tcp(7).

Postel, Jon, *User Datagram Protocol*, RFC 768, Network Information Center, SRI International, Menlo Park, Calif., August 1980.

DIAGNOSTICS

A socket operation may fail if:

EISCONN	A connect() operation was attempted on a socket on which a connect() operation had already been performed, and the socket could not be successfully disconnected before making the new connection.
EISCONN	A sendto() or sendmsg() operation specifying an address to which the message should be sent was attempted on a socket on which a connect() operation had already been performed.
ENOTCONN	A send() or write() operation, or a sendto() or sendmsg() operation not specifying an address to which the message should be sent, was attempted on a socket on which a connect() operation had not already been performed.
EADDRINUSE	A bind() operation was attempted on a socket with a network address/port pair that has already been bound to another socket.
EADDRNOTAVAIL	A bind() operation was attempted on a socket with a network address for which no network interface exists.
EINVAL	A sendmsg() operation with a non-NULL msg_accrights was attempted.
EACCES	A bind() operation was attempted with a reserved port number and the effective user ID of the process was not the privileged user.
ENOBUFS	The system ran out of memory for internal data structures.

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 365, ENGLEWOOD CLIFFS, NJ

POSTAGE WILL BE PAID BY ADDRESSEE

PRENTICE HALL
Attn: PTR Marketing Manager
College Marketing Department
Route 9W
Englewood Cliffs, NJ 07632-9940

AT&T

What do YOU think?

AT&T values your opinion. Please indicate your opinions in each of the following areas. We'd like to know how well this document meets your needs.

Book Title: ____________________

	Excellent	Good	Fair	Poor
Accuracy - Is the information correct?	☐	☐	☐	☐
Completeness - Is information missing?	☐	☐	☐	☐
Organization - Is information easy to find?	☐	☐	☐	☐
Clarity - Do you understand the information?	☐	☐	☐	☐
Examples - Are there enough?	☐	☐	☐	☐
Illustrations - Are there enough?	☐	☐	☐	☐
Appearance - Do you like the page format?	☐	☐	☐	☐
Physical binding - Do you like the cover and binding?	☐	☐	☐	☐

Does the document meet your needs? Why or why not?

____

____

____

What is the single most important improvement that we could make to this document?

____

____

____

Please complete the following information.

Name (Optional): ____________________

Job Title or Function: ____________________

Organization: ____________________

Address: ____________________

____

Phone: (   ) ____________________

If we need more information may we contact you? Yes ☐ No ☐

Thank you.

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS MAIL · PERMIT NO. 199 SUMMIT, NJ

POSTAGE WILL BE PAID BY ADDRESSEE

AT&T
Department Head
UNIX System Documentation and Development Dept.
Room F-308
190 River Road
Summit, NJ 07901-9907

307-305

UNIX® SYSTEM V RELEASE 4
Network User's and Administrator's Guide

ISBN 0-13-933813-6

90000>

9 780139 338137

UNIX
PRESS
A Prentice Hall Title

PRESS