10 Administering Network Services

This chapter contains information about checking on remote system status, logging in to a remote system, transferring files between systems, and administering the Network Information Service Plus (NIS+) databases. It also introduces the IPv6 Internet protocol, new in the Solaris 8 release, and describes how to display network and configuration information.

New! This chapter also provides information about the Secure Shell commands, new in the Solaris 9 release, that enable users to securely access a remote host on an unsecured network. It also contains brief instructions on creating and editing local network configuration files and on using the `snoop` command.

Configuring Systems for a Network

New!

When you install Solaris, network software is installed along with the operating system software. At installation time, certain IP configuration parameters are stored in appropriate files so that they can be read when the system boots.

The parameters that are supplied during network configuration are listed below.

- IP address of the network interface for the system.
- Host name of the system.

- NIS, NIS+, or DNS domain name in which the system resides, if applicable.
- Default router address.
- Subnet mask.

Configuring a Host for Local Files Mode

Use the following steps to configure TCP/IP on a system that runs in local files mode. You may need to use this procedure if you add a new network interface to your system after the initial Solaris software installation.

1. Become superuser.
2. Type **`cd /etc`** and press Return.
3. Create a file named **`/etc/hostname.`*`interface`*** or **`/etc/hostname6.`*`interface`*** for each network interface.

   The Solaris installation program creates this file automatically for the primary network interface. This file maps host names to interfaces for IPv4. For IPv6, you need one `/etc/hostname.`*`interface`* or `/etc/hostname6.`*`interface`* file for each system, for example, `hostname.le0` or `hostname6.le0`.
4. Edit the `/etc/hostname.`*`interface`* or `/etc/hostname6.`*`interface`* file and type either the system's IP address or its host name.

NOTE. The Solaris installation program creates the default `/etc/inet/hosts` *file for the local system. The old* `/etc/hosts` *name for this file is now a symbolic link to* `/etc/inet/hosts`. *If you are using IPv6, the installation program creates the default* `/etc/inet/ipnodes` *file.*

5. Edit the `/etc/inet/hosts` file to add any IP addresses that you have assigned to any additional network interfaces in the local system along with the corresponding host name for each interface. If you are running IPv4, you do not need to create the `/etc/inet/ipnodes` file. If you have any IPv6 systems, copy all of the IPv4 IP addresses and host names from `/etc/inet/hosts` to the `/etc/inet/ipnodes` file. Add the IP addresses and host names for IPv6 systems only to the `/etc/inet/ipnodes` file.

NOTE. Put only the host name(s) and IP address(es) of network interfaces that are in each system in the `/etc/inet/hosts` *file. DNS should handle all external host-name-to-IP-address mappings; you must, therefore, properly configure the* `/etc/nsswitch.conf` *and* `/etc/resolv.conf` *files to make this work. Follow this convention* *because you (as the system administrator) normally don't control the network or other systems on the network. If, for example, the owners of other systems or network equipment change their IP addresses or host names in DNS, the* `/etc/inet/hosts` *file on each of the systems under your control would then be out of date and each system's network configuration would mysteriously no longer work.*

6. If the `/usr` file system is NFS mounted, also add the IP address or addresses of the file server to the `/etc/inet/hosts` file.
7. Edit the `/etc/defaultrouter` file and type the router's IP address.

   This file should contain an entry for each router that is directly connected to the network. The entry should be the IP address of an interface on the router that is on the same subnet as the system you're configuring.
8. Edit the `/etc/inet/hosts` file and type the name of the default router and its IP addresses.
9. If the network is subnetted, edit the `/etc/inet/netmasks` file and type the network number and netmask.

   If you have set up an NIS, NIS+, or LDAP server, you can type netmask information in the `netmasks` database on the server if server and clients are on the same network.
10. Reboot the system.

Checking on Remote System Status

This section describes commands you use to find out the status of remote systems: `rup`, `ping`, and `rpcinfo -d`.

Determining How Long a Remote System Has Been Up (rup)

To find out how long a system has been up and to determine the load average, type **`rup`** ***`system-name`*** and press Return. The host name, uptime, and load average are displayed.

```
oak% rup ash
ash    up 59 days,  3:42, load average: 0.12, 0.12, 0.01
oak%
```

You can also display a list of all remote hosts in the subnet by typing **rup** and pressing Return. If you display a list, you can use the options shown in Table 78 to sort the output.

Table 78 Options to the rup Command

Option	Description
-h	Sort the display alphabetically by host name.
-l	Sort the display alphabetically by load average.
-t	Sort the display by uptime.

In the following example, the output is sorted alphabetically by host name.

```
oak% rup -h
ash    up  1 day,   1:42,   load average: 0.00, 0.31, 0.34

elm    up 14 days,  0 min,  load average: 0.07, 0.01, 0.00

maple  up 32 days,  14:39,  load average: 0.21, 0.05, 0.00

oak    up  8 days,  15:44,  load average: 0.02, 0.00, 0.00
oak%
```

Determining Whether a Remote System Is Up (ping, rup, rpcinfo -p)

Use the following steps to determine whether a remote system is up and to log in to the remote system.

1. Type **ping *system-name*** and press Return.

   The message `*system-name* is alive` means the system is accessible over the network. The message `ping: unknown host *system-name*` means the system name is not known on the network. The message `ping: no answer from *system-name*` means the system is known on the network but is not up at this time.
2. Type **rup *system-name*** and press Return.

   Information about how long the system has been up and the load average is displayed.
3. Type **rpcinfo -p *system-name*** and press Return.

   Information about RPC services is displayed.
4. Type **rlogin *system-name*** and press Return.

   You are logged in to the remote system.

```
cinderella% ping drusilla
drusilla is alive
```

```
cinderella% rup drusilla
   drusilla    up  3 days,  15:10   load average: 0.07, 0.08, 0.09
cinderella% rpcinfo -p drusilla
program  vers proto port  service
100000    3   udp    111  portmapper
100000    2   udp    111  portmapper
100000    3   tcp    111  portmapper
100000    2   tcp    111  portmapper
100007    3   tcp   1029  ypbind
100007    3   udp   1025  ypbind
100021    1   tcp   1030  nlockmgr
100021    1   udp   1026  nlockmgr
100024    1   tcp   1028  status
100024    1   udp   1027  status
100021    3   tcp   1030  nlockmgr
100021    3   udp   1026  nlockmgr
100020    2   tcp   4045  llockmgr
100020    2   udp   4045  llockmgr
100021    2   tcp   1030  nlockmgr
100021    2   udp   1026  nlockmgr
100087   10   udp   1031  adm_agent
100011    1   udp   1034  rquotad
100002    1   udp   1037  rusersd
100002    2   udp   1037  rusersd
100012    1   udp   1041  sprayd
100008    1   udp   1043  walld
100001    2   udp   1046  rstatd
100001    3   udp   1046  rstatd
100001    4   udp   1046  rstatd
100068    2   udp   1049  cmsd
100068    3   udp   1049  cmsd
100083    1   tcp   4049
cinderella% rlogin drusilla
Password:
Last login: Mon Mar  2 10:31:55 from cinderella
drusilla%
```

You can also use `ping` with a system's IP address by typing **`ping`** ***`IP-address`*** and pressing Return. The message *`IP-address`* `is alive` means the system is accessible over the network. The message `ping: no answer from` *`IP-address`* means the system is not available to the network. The message `ping: unknown host` *`IP-address`* means the system name is not known on the network.

```
oak% ping 129.144.52.119
129.144.52.119 is alive
oak% ping 129.137.67.234
ping: unknown host 129.137.67.234
oak% ping 129.145.52.119
ping: no answer from 129.145.52.119
oak%
```

Logging In to a Remote System (rlogin)

NOTE. Starting with the Solaris 9 release, Secure Shell is recommended for secure remote login. See "Secure Shell Commands" on page 359 for more information. New!

Use the following steps to log in to a remote system.

1. Type **`rlogin`** ***`system-name`*** and press Return. You may be prompted for a password.
2. If you have a local account on that system, type your local password. Otherwise, type your NIS, NIS+, or LDAP password.

   Unless you have a home directory that is accessible on the remote system (because it is local on that system or because it is hard-mounted or automounted), you log in to the root (/) directory.

```
oak% rlogin ash
Password:
No directory!  Logging in with home=/
Last login: Tue Sep 17 13:54:28 from 129.144.52.119
Sun Microsystems, Inc. SunOS 5.8    Generic February 2000
ash%
```

Authentication for Remote Logins (rlogin)

The remote system or the network environment can perform authentication to establish who the user is for `rlogin` operations.

The main differences between these forms of authentication are in the type of interaction they require from the user and the way the authentication is established. If a remote system tries to authenticate a user, the user is prompted for a password unless the user is included in the `/etc/hosts.equiv` or `.rhosts` file on the remote system. If the network authenticates the user, no password is required because the network already knows who the user is.

New!

Network authentication relies on either a trusting network environment set up with your local nameservice and the automounter or one of the nameservices pointed to by the remote system's `/etc/nsswitch.conf` file.

NOTE. Network authentication usually supersedes system authentication.

New!

The `rlogin` command also interacts with the Pluggable Authentication Module (PAM) subsystem for authentication and may require configuration of the `/etc/pam.conf` file for authentication to work. For complete information on PAM, refer to the Sun *System Administration Guide: Security Services* or the "Using Authentication Services" chapter in the *Solaris Advanced System Administrator's Guide* available from Sun Microsystems Press and Prentice Hall.

Remote System Authentication

When the remote system tries to authenticate a user, it relies on information in its local /etc/hosts.equiv or .rhosts files. If the user's system or host name is included in the remote system's /etc/hosts.equiv file, authentication is automatic and the user can use the rlogin command without typing a password. Alternatively, authentication is automatic with the rlogin command when the user has a remote home directory with a .rhosts file that includes the user's system name and user name.

The /etc/hosts.equiv File The /etc/hosts.equiv file contains a list of trusted hosts for a remote system, one entry per line. If a user tries to log in remotely with the rlogin command from one of the hosts listed in this file, and if the remote system can access the password entry for the user, the remote system enables the user to log in without a password.

A typical hosts.equiv file has the following structure.

```
host1
host2 user_a
+@engineering
-@marketing
```

When the /etc/hosts.equiv file contains an entry consisting of just a host name, such as the host1 entry above, the host is trusted and so is any user at that system.

If the user name is also mentioned, as in the second entry above, then the host is trusted only for that specified user.

A netgroup name preceded by a plus sign (+) means that all the systems in that netgroup are considered trusted.

A netgroup name preceded by a minus sign (–) means that none of the systems in that netgroup are considered trusted.

A single line of + in the /etc/hosts.equiv file indicates that every known host is trusted.

The /etc/hosts.equiv file presents a security risk, especially if it contains a + entry. If you maintain an /etc/hosts.equiv file on a system, include only trusted hosts in your network. Do not include any host that belongs to a different network or any systems that are in public areas. For example, do not include a host for which you do not have administrative control.

The .rhosts File The .rhosts file is the user equivalent of the /etc/hosts.equiv file. It contains a list of host-user combinations instead of hosts in general. If a host-user combination is listed in this file, the specified user is granted permission to log in remotely from the specified host without having to supply a password.

NOTE. A `.rhosts` *file must reside at the top level of a user's home directory.* `.rhosts` *files located in subdirectories are not consulted.*

Users can create `.rhosts` files in their home directories. Using the `.rhosts` file is another way to enable trusted access between an individual's user accounts on different systems without using the `/etc/hosts.equiv` file.

Unfortunately, the `.rhosts` file presents a major security problem. While the `/etc/hosts.equiv` file is under the control of system administrators and can be managed effectively, any user can create a `.rhosts` file granting access to whomever the user chooses without the system administrator's knowledge. The only secure way to manage `.rhosts` files is to completely disallow them.

Use the following procedures to search and remove `.rhosts` files.

1. Become superuser.
2. All on one line, type **`find`** ***`home-directories`*** **`-name .rhosts -print -exec rm{} \;`** and press Return.

   The `find` command starts at the designated directory and searches for any file named `.rhosts`. If any `.rhosts` files are found, the path is printed on the screen and the file is removed.

The following example removes all `.rhosts` files in the users' home directories located in the `/export/home` directory.

```
paperbark% su
Password:
# find /export/home -name .rhosts -print -exec rm{} \;
/export/home/ray/.rhosts
/export/home/des/.rhosts
#
```

Network Authentication

Network information is stored in NIS maps, NIS+ tables, or LDAP. Network authentication relies on one of the following two methods.

- A trusting network environment that has been set up with the user's local network information service and the automounters.
- One of the network information services pointed to by the `/etc/nsswitch.conf` file on the remote system that contains information about the user.

What Happens After You Log In Remotely

When you log in to a remote system, the in.rlogind daemon tries to find your home directory. If the in.rlogind daemon can't find your home directory, it assigns you to the root (/) directory on the remote system and the following message is displayed.

```
Unable to find home directory, logging in with /
```

New!

When you invoke the rlogin command on your local host, inetd(1M) on the remote host invokes the in.rlogind daemon. The server checks the client's source port. If the port is not in the range 512–1023, the server aborts the connection. The server checks the client's source address. If an entry for the client exists in both /etc/inet/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/inet/hosts or if the host name is found in the NIS or NIS+ hosts map or in DNS, the user is prompted for a password, regardless of whether an entry for the client is present in /etc/hosts.equiv.

Once the source port and address are checked, in.rlogind allocates a pseudoterminal and manipulates file descriptors so that the slave half of the pseudoterminal becomes the standard input, standard output, and standard error 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 pam(3PAM) authentication process. If the login program finds your home directory, it sources both the .cshrc and .login files for the C shell or the .profile file for the Bourne shell. Therefore, your prompt on the remote system is your standard login prompt, and the current directory is the same as for a local login. For example, if your usual prompt is your system name followed by the percent (%) sign, such as paperbark%, when you log in to a remote system, the remote system name is displayed as the login prompt.

In the following example, user winsor remotely logs in to the system castle and displays the current working directory.

```
paperbark% rlogin castle
Password:
Last login: Tue Jun 20 14:02:01 from :0
Sun Microsystems Inc.   SunOS 5.7       Generic October 1998
You have mail.
castle% pwd
/export/home/winsor
castle%
```

Logging Out from a Remote System

You use the `exit(1)` command to log out from a remote system.

The following example shows the user `winsor` logging out from the system `castle`.

```
castle% exit
castle% logout
Connection closed.
paperbark%
```

Transferring Files Between Systems (rcp, ftp)

New!

NOTE. Starting with the Solaris 9 release, Secure Shell is recommended for secure remote copy and file transfer protocol. See "Secure Shell Commands" on page 359 for more information.

If the automounter is set up for your site, you can transfer files between systems by using commands such as `cp` and `mv`. This section describes how to use the `rcp` and `ftp` commands to transfer files between systems.

Using the rcp Command

To transfer a file from a remote system to your system with the remote copy command, type **`rcp`** ***`system-name:source-pathname destination`*** and press Return. If you have proper security to access the remote system, the file is copied to the destination you specify.

In the following example, the file `quest` is copied from the `/tmp` directory on the system `ash` to the current working directory on the system `oak`.

```
oak% rcp ash:/tmp/quest .
oak%
```

To transfer a file from a local system to a remote system, type **`rcp`** ***`pathname system-name:destination-pathname`*** and press Return. If you have proper security to access the remote system, the file is copied from the local system to the remote destination you specify.

In the following example, the file `quest` is copied from the current working directory on the system `oak` to the `/tmp` directory on the system `ash`.

```
oak% rcp quest ash:/tmp
oak%
```

If you want, you can rename the file as part of the destination path name. For example, to rename the file `quest` to `questions` and put it in the `/tmp` directory, type **`/tmp/questions`** as the destination path name.

Using the File Transfer Program (ftp)

Use the following steps to transfer files from your local system to a remote system by using the file transfer program.

NOTE. You may need to have an account on each system to use the file transfer program. Some systems allow read-only `ftp` *access to anybody who logs in as* `anonymous` *and types a login name at the password prompt.*

If you have an NIS, NIS+, or LDAP account, you can use your login name and network password to access a remote system by using `ftp`.

1. Type **`ftp`** and press Return.

   The `ftp>` prompt is displayed.
2. Type **`open`** ***`remote-system-name`*** and press Return.

   System connection messages are displayed, and you are asked for a user name.
3. Type the user name for your account on the remote system and press Return.

   If a password is required, you are asked to enter it.
4. Type the password (if required) for your account on the remote system and press Return.

   A system login message and the `ftp>` prompt are displayed.
5. Type **`bin`** to set binary format or **`asc`** to set ASCII format and press Return.

   The file type is set. ASCII is the default format.
6. Type **`put`** ***`local-filename destination-filename`*** and press Return to transfer a single file.

   File transfer messages and the `ftp>` prompt are displayed.
7. Type **`quit`** and press Return.

   A goodbye message and the command prompt are displayed.

The following example establishes an `ftp` connection from the system `oak` to the system `elm`, specifies ASCII format, puts the file `quest` from `oak` into the `/tmp/quest` directory on `elm`, and quits the session.

```
oak% ftp
ftp> open elm
Connected to elm
220 elm FTP server (UNIX(r) System V Release 4.0) ready.

Name (elm:ignatz): ignatz
331 Password required for ignatz.
Password:
230 User ignatz logged in.
ftp> asc
ftp> put quest /tmp/quest
200 PORT command successful.

150 ASCII data connection for /tmp/quest (129.144.52.119,1333).

226 Transfer complete.
ftp> quit
221 Goodbye.
oak%
```

You can use the `send` command as an alternative to the `put` command. You can copy multiple files by using the `mput` command. There is no `msend` command. See the `ftp`(1) manual page for more information.

NOTE. You must have an account on each system to use the file transfer program.

If you have an NIS, NIS+, or LDAP account, you can use your login name and network password to access a remote system with `ftp`. Use the following steps to transfer files from a remote system to your local system by using the file transfer program.

1. Type **`ftp`** and press Return.

   The `ftp>` prompt is displayed.
2. Type **`open`** ***`remote-system-name`*** and press Return.

   System connection messages are displayed, and you are asked for a user name.
3. Type the user name for your account on the remote system and press Return.

   If a password is required, you are asked to enter it.
4. Type the password (if required) for your account on the remote system and press Return.

   A system login message and the `ftp>` prompt are displayed.
5. Type **`bin`** to set binary format or **`asc`** to set ASCII format and press Return.

   The file type is set. ASCII is the default format.
6. Type **`get`** ***`remote-filename destination-filename`*** and press Return.

   File transfer messages and the `ftp>` prompt are displayed.

7. Type **`quit`** and press Return. A goodbye message and the command prompt are displayed.

The following example establishes an `ftp` connection from the system `oak` to the system `elm`, specifies ASCII format, gets the file `quest` from `elm`, puts it into the `/tmp/quest` directory on `oak`, and quits the session.

```
oak% ftp
ftp> open elm
Connected to elm
220 elm FTP server (UNIX(r)System V Release 4.0) ready.

Name (elm:ignatz): ignatz
331 Password required for ignatz.
Password:
230 User ignatz logged in.

ftp> asc
ftp> get quest /tmp/quest
200 PORT command successful.
150 ASCII data connection for /tmp/quest (129.144.52.119,1333).
226 Transfer complete.

ftp> quit
221 Goodbye.
oak%
```

NOTE. You can copy multiple files by using the `mget` *command. See the* `ftp`*(1) manual page for more information.*

Administering NIS+ Databases

NIS+ provides a central store of information for network resources such as hosts, users, and mailboxes. NIS+ replaces NIS (Network Information Service) and provides the following enhancements.

NOTE. LDAP is now scheduled to replace NIS+. New!

- An organizational framework that is simpler to administer in large companies.
- Improved security.
- Improved distribution time to propagate changes through the network.

In addition, the Solaris Operating Environment provides a nameservice switch file, `/etc/nsswitch.conf`, that lets you use several different network information services at once. The `/etc/nsswitch.conf` file also lets you specify which service provides which type of information. In previous SunOS releases, selection of the nameservice was hard-coded into the services, which made it difficult to switch to a new nameservice. The `/etc/nsswitch.conf` file defines the order in which local files and network databases are searched for information. Describing how to set up NIS+ is beyond the scope of this book.

Using NIS+ Tables

NIS+ tables correspond to NIS maps. The Solaris Operating Environment provides 16 types of tables (shown in Figure 19) that store the network information used by NIS+.

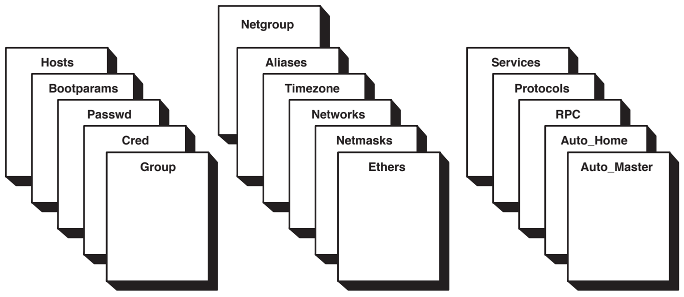

Figure 19 The 16 NIS+ Tables

Each table stores a different type of information about users, workstations, or resources on the network. For instance, the Hosts table stores the host name and network address of every workstation in the domain; the Bootparams table stores the location of the root, swap, and dump directories of the diskless clients in the domain.

Each domain can have its own set of these NIS+ tables, which store all the NIS+ information for that particular domain. Table 79 lists the 16 NIS+ tables and the information they store.

Table 79 NIS+ Tables

Table	Information in the Table
Hosts	Network address and host name of every workstation in the domain.
Bootparams	Location of the root, swap, and dump partition of every diskless client in the domain.
Password	Password information about every NIS+ principal (Nobody, Owner, Group, or World) in the domain, plus a pointer to the shadow file.
Cred	Credentials for principals who have permission to access the information or objects in the domain.
Group	Password, group ID, and members of every group in the domain.

Table 79 NIS+ Tables (Continued)

Table	Information in the Table
Netgroup	The netgroups to which workstations and users in the domain may belong.
Aliases	Information about the `sendmail` and e-mail aliases of individual users in the domain.
Timezone	The time zone of every workstation in the domain.
Networks	The networks in the domain and their canonical names.
Netmasks	The networks in the domain and their associated netmasks.
Ethers	The Ethernet address of every workstation in the domain.
Services	The names of IP services used in the domain and their port numbers.
Protocols	The list of IP protocols used in the domain.
RPC	The RPC program numbers for RPC services available in the domain.
Auto_Home	The location of all users' home directories in the domain.
Auto_Master	Automounter map information.

You can access information in NIS+ tables either by entry row or by column, as shown in Figure 20.

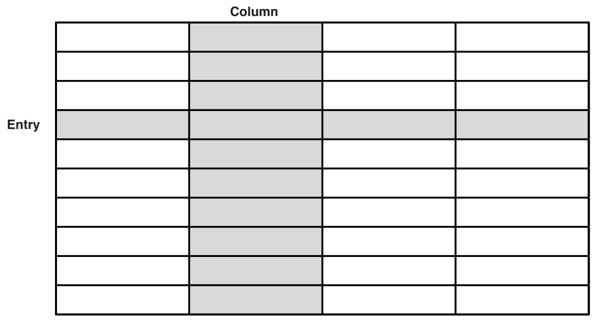

Figure 20 Entry Row and Columns in a Table

For example, if you want to find the network address of a workstation named `drusilla` in the `Hosts` database, you can ask a search program to look through the `hostname` column until it finds `drusilla`, as shown in Figure 21. The program then searches the `drusilla` entry row to find its network address, as shown in Figure 22.

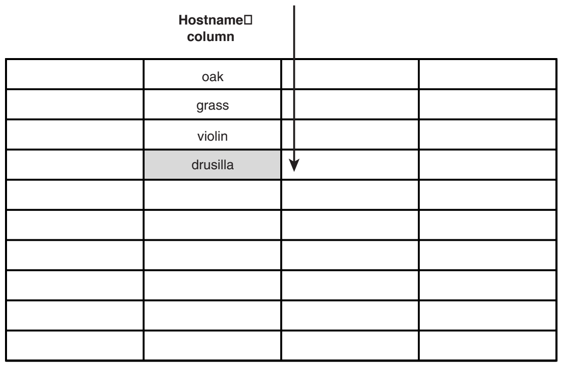

Figure 21 Searching the Hostname Column

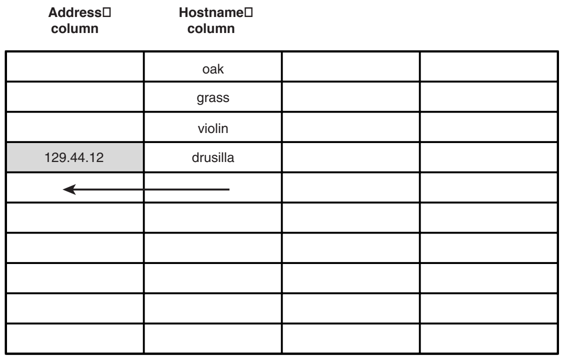

Figure 22 Finding a Network Address

You can use NIS+ commands to perform these types of searches for you. Table 80 lists the NIS+ administrative commands.

Table 80 NIS+ Administrative Commands

Command	Description
`nistbladm`	Display, add, modify, and delete information in an NIS+ table.
`nisgrep`	Search for information in an NIS+ table.
`nismatch`	Search for information in an NIS+ table.
`niscat`	Display the entire contents of an NIS+ table.

See the manual pages for information about how to use these commands.

NIS+ Security

NIS+ uses a security authorization model that is similar to the UNIX file system model. It specifies that each item in the namespace as well as each record, each column, and each row has associated with it a set of access rights that are granted to four broad classes of principals.

- The owner of the item.
- A group owner of the item.
- All other principals.
- `nobody`—the class of users not defined in the NIS+ domain or those users accessing NIS+ resources from NIS clients.

The specific access rights are different from the traditional read, write, and execute rights of file systems because of the nature of information services. Refer to your system manual for more information about NIS+ security.

Using SMC Computers and Networks Tool

New!

Starting with the Solaris 9 release, you can use the SMC System Configuration/Computers and Networks tool to administer computers and networks. With this tool, you can also create multihomed hosts and rename a computer.

Introducing the IPv6 Internet Protocol

Internet Protocol, version 6 (IPv6) was introduced in the Solaris 8 release. This new protocol version evolved from the current IPv4 version, which is also supported in the Solaris 8 Operating Environment. IPv6 adds increased address space and improves Internet functionality by use of a simplified header format, support for authentication and privacy, autoconfiguration of address assignments, and new quality-of-service capabilities. Networking commands in the Solaris 8 release have been amended to include support for both the IPv4 and IPv6 network protocols.

You can enable IPv6 on a system when you install the Solaris 8 software. If you answer yes to enable the IPv6 during the installation process, you do not need to enable IPv6 manually. Describing how to enable IPv6 manually is beyond the scope of this book. Refer to Sun's *System Administration Guide, IP Services*, for more information.

The IPv6 protocol changes are summarized below.

Expanded Routing and Addressing Capabilities

IPv6 increases the IP address size from 32 bits to 128 bits to support more levels of addressing hierarchy, provide more addressable nodes, and use simpler autoconfiguration of addresses.

A scope field improves the scalability of multicast routing to multicast addresses.

IPv6 supports three types of addresses: `unicast`, `anycast`, and `multicast`. The new `anycast` address is defined to identify sets of nodes, whereby a packet sent to an `anycast` address is delivered to one of the nodes. The use of `anycast` addresses in the IPv6 source route enables nodes to control the path over which their traffic flows.

IPv6 has no broadcast addresses. Multicast addresses are used instead.

Simplified Header Format

Some IPv4 header fields have been dropped or made optional to reduce the common-case processing cost of packet handling. Bandwidth cost of the IPv6 header is kept as low as possible, despite the increased size of the addresses. Even though the IPv6 addresses are four times longer than IPv4 addresses, the IPv6 header is only twice the size of the IPv4 header.

Improved Support for Options

IP header options are encoded to enable more efficient forwarding, less stringent limits on the length of options, and greater flexibility for introducing new options in the future.

Quality-of-Service Capabilities

A new capability enables the labeling of packets belonging to particular traffic flows for which the sender requests special handling, such as nondefault quality of service or real-time service.

Authentication and Privacy Capabilities

IPv6 includes the definition of extensions that provide support for authentication, data integrity, and confidentiality.

Showing Network Status (netstat)

You can use the `netstat`(1M) command to display the following network status information.

- A list of active sockets for each protocol.
- The state of the interfaces.
- The routing table.
- The multicast routing table.
- The state of DHCP on one or all interfaces.

The Solaris release supports both the IPv4 and IPv6 network interfaces. In the Solaris 8 release, the `netstat` command has been updated to include the IPv6 interfaces.

Displaying Status of Active TCP and UDP Ports

Use the `netstat` command with no arguments to display the status of active TCP and UDP ports. The following example shows the output of the `netstat` command with no arguments, to display the status of active TCP and UDP ports.

```
paperbark% netstat

TCP: IPv4
   Local Address        Remote Address    Swind Send-Q Rwind Recv-Q  State
-------------------- -------------------- ----- ------ ----- ------ -------
localhost.32786      localhost.32773      32768      0 32768      0 ESTABLISHED
localhost.32773      localhost.32786      32768      0 32768      0 ESTABLISHED
localhost.32789      localhost.32784      32768      0 32768      0 ESTABLISHED
localhost.32784      localhost.32789      32768      0 32768      0 ESTABLISHED
localhost.32792      localhost.32791      32768      0 32768      0 ESTABLISHED
localhost.32791      localhost.32792      32768      0 32768      0 ESTABLISHED
localhost.32795      localhost.32784      32768      0 32768      0 ESTABLISHED
localhost.32784      localhost.32795      32768      0 32768      0 ESTABLISHED
localhost.32798      localhost.32797      32768      0 32768      0 ESTABLISHED
localhost.32797      localhost.32798      32768      0 32768      0 ESTABLISHED
localhost.32813      localhost.32784      32768      0 32768      0 ESTABLISHED
localhost.32784      localhost.32813      32768      0 32768      0 ESTABLISHED
localhost.32816      localhost.32815      32767      0 32768      0 ESTABLISHED
localhost.32815      localhost.32816      32768      0 32768      0 ESTABLISHED
paperbark.32891      G3.ftp               17520      0 24820      0 ESTABLISHED
paperbark.8888       paperbark.32904      32768      0 32768      0 TIME_WAIT
paperbark.32905      paperbark.32779      32768      0 32768      0 TIME_WAIT

Active UNIX domain sockets
Address  Type          Vnode     Conn  Local Addr      Remote Addr
707f1d90 stream-ord 705b89e0 00000000 /tmp/.X11-unix/X0
707f1ea8 stream-ord 00000000 00000000
paperbark%
```

Displaying the Status of Network Interfaces

Use the `-i` option to the `netstat` command to display the status of network interfaces. The following example uses the `netstat -i` command on the system `paperbark` to display the status of network interfaces.

```
paperbark% netstat -i
Name  Mtu  Net/Dest      Address        Ipkts  Ierrs Opkts  Oerrs Collis Queue
lo0   8232 loopback      localhost      11787  0     11787  0     0      0
hme0  1500 paperbark     paperbark      8      0     5      0     0      0

paperbark%
```

Displaying Kernel Routing Tables

Use the `-r` option to the `netstat` command to display kernel routing tables, and use the `-n` option to display network addresses as numbers. The following example uses the `netstat -r -n` command to display the kernel's routing tables with the network addresses as numbers.

```
paperbark% netstat -r -n

Routing Table: IPv4
  Destination           Gateway           Flags  Ref   Use   Interface
-------------------- -------------------- ----- ----- ------ ---------
172.16.8.0           172.16.8.22           U        1      0  hme0
224.0.0.0            172.16.8.22           U        1      0  hme0
127.0.0.1            127.0.0.1             UH      16  11150  lo0
paperbark%
```

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

Displaying Network Interface Parameters (ifconfig)

You can use the `ifconfig` command to display information about specific interfaces, assign an address to a network interface, or configure network interfaces. The `/etc/rc2.d` scripts run `ifconfig` at boot time to define the network address of each interface present on a system. You can also use `ifconfig` at a later time to redefine an interface address or other operating parameters. Refer to the `ifconfig`(1M) manual page for complete information. The following sections describe how to use the `ifconfig` command to display information about specific interfaces.

The `ifconfig` command has been modified in the Solaris 8 release to create the IPv6 stack and to support new parameters.

Displaying Information About All Interfaces on a System

Use the -a option of the ifconfig command to display information about all interfaces on a system. The following example shows the interfaces on the system paperbark.

```
paperbark% ifconfig -a
lo0: flags=1000849<UP,LOOPBACK,RUNNING,MULTICAST,IPv4> mtu 8232 index 1
        inet 127.0.0.1 netmask ff000000
hme0: flags=1000843<UP,BROADCAST,RUNNING,MULTICAST,IPv4> mtu 1500 index 2
        inet 172.16.8.22 netmask ffffff00 broadcast 172.16.8.255
paperbark%
```

The flags section shows the status of the interface. The mtu field tells you the maximum transfer size in octets. Information on the second line includes the IP address of the host you are using, the netmask currently being used, and the IP broadcast address of the interface.

The following example shows the interfaces on the system castle.

```
castle% ifconfig -a
lo0: flags=849<UP,LOOPBACK,RUNNING,MULTICAST> mtu 8232
        inet 127.0.0.1 netmask ff000000
le0: flags=863<UP,BROADCAST,NOTRAILERS,RUNNING,MULTICAST> mtu 1500
        inet 172.16.8.19 netmask ffff0000 broadcast 172.16.255.255
castle%
```

Displaying Information About Specific Interfaces

Use the following syntax to display information about the configuration of a specific interface.

```
ifconfig interface-name [protocol-family]
```

The following example displays information about the hme0 interface.

```
paperbark% su
Password
# ifconfig hme0
hme0: flags=1000843<UP,BROADCAST,RUNNING,MULTICAST,IPv4> mtu 1500 index 2
        inet 172.16.8.22 netmask ffffff00 broadcast 172.16.8.255
        ether 8:0:20:7d:79:d4
#
```

The flags section shows that the interface is configured UP, is capable of broadcasting, and not using trailer link-level encapsulation. The mtu field tells you that this interface has a maximum transfer size of 1500 octets.

Information on the second line includes the IP address of the host, the netmask currently being used, and the IP broadcast address of the interface. The third line gives the system address (in this case, Ethernet) of the host.

New! Displaying Packet Contents

You can use the snoop(1M) command to capture network packets and display their contents. You can display packets as soon as they are received or save them to a file. When snoop writes to an intermediate file, it is unlikely that you will lose packets under busy trace conditions. You can then use snoop to interpret the file. See the snoop(1M) manual page for more information about using the snoop command.

You must run snoop as root to capture packets to and from the default interface in promiscuous mode. In summary form, only data that pertains to the highest-level protocol is displayed.

Checking All Packets from Your System

Use the following steps to check all packets from your system.

1. Become superuser.
2. Type **netstat -i** and press Return.

   Review the output to determine the interfaces that are attached to the system.
3. Type **snoop** and press Return.

   Packet information is displayed.
4. Press **Control-C** to halt the process.

The following example traces packets during an FTP file transfer.

```
mopoke% netstat -i
Name  Mtu  Net/Dest      Address        Ipkts  Ierrs Opkts  Oerrs Collis Queue
lo0   8232 loopback      localhost      11197  0     11197  0     0      0
eri0  1500 mopoke        mopoke         537    0     9      3     0      0

mopoke% su
Password:
# snoop
Using device /dev/eri (promiscuous mode)
      mopoke -> G4          FTP C port=32830 PORT 172,16,8,25,128
          G4 -> mopoke      FTP R port=32830 200 PORT command suc
      mopoke -> G4          FTP C port=32830 STOR examples\r\n
          G4 -> mopoke      FTP-DATA R port=32834
      mopoke -> G4          FTP-DATA C port=32834
          G4 -> mopoke      FTP-DATA R port=32834
          G4 -> mopoke      FTP R port=32830 150 Opening BINARY m
      mopoke -> G4          FTP-DATA C port=32834 mopoke% netstat -i\nN
      mopoke -> G4          FTP-DATA C port=32834
          G4 -> mopoke      FTP-DATA R port=32834
          G4 -> mopoke      FTP-DATA R port=32834
      mopoke -> G4          FTP-DATA C port=32834
      mopoke -> G4           FTP C port=32830
          G4 -> mopoke       FTP R port=32830 226 Transfer complet
      mopoke -> G4           FTP C port=32830
          G4 -> 172.16.8.255 UDP D=631 S=631 LEN=76
          G4 -> 172.16.8.255 UDP D=631 S=631 LEN=118
          G4 -> 172.16.8.255 UDP D=631 S=631 LEN=107
^C#
```

Capturing snoop Results to a File

Use the following steps to capture `snoop` results to a file.

1. Become superuser.
2. Type **`snoop -o`** ***`filename`*** and press Return.

   Review the output to determine the interfaces that are attached to the system.
3. To inspect the file, type **`snoop -i`** ***`filename`*** and press Return.

Secure Shell Commands

New!

Secure Shell commands enable users to securely access a remote host on an unsecured network. Passwords, public keys, or both provide authentication. All network traffic is encrypted to prevent others from reading an intercepted communication or spoofing the system.

The Solaris 9 release provides the Secure Shell commands described in Table 81 to communicate among systems.

Table 81 Secure Shell Commands

Command	Description
`scp(1)`	Secure copy (remote file copy program).
`sftp(1)`	Secure file transfer program.
`ssh(1)`	Open SSH client (remote login program).
`ssh-add(1)`	Add RSA or DSA identities for the authentication agent.
`ssh-agent(1)`	Authentication agent.
`ssh-http-proxy-connect(1)`	
	Secure Shell proxy for HTTP.
`ssh-keygen(1)`	
	Authentication key generation.

Table 81 Secure Shell Commands (Continued)

Command	Description
`ssh-socks5-proxy-connect(1)`	
	Secure Shell proxy for SOCKS5.
`sshd(1M)`	Secure Shell daemon.
`ssh_config(4)`	
	SSH client configuration file.
`sshd_config(4)`	
	SSH server configuration file.

Users can be authenticated with an account password or with a public/private key pair stored on the local host in the user's home directory in the `.ssh` subdirectory. The remote host is provided with the public key, which is required to complete the authentication. Table 82 lists the default names for the identity files that store the public and private keys.

Table 82 Naming Conventions for Private/Public Keys

Private Key	Public Key	Cipher and Protocol Version
`identity`	`identity.pub`	RSA v1
`id_rsa`	`id_rsa.pub`	RSA v2
`id_dsa`	`id_dsa.pub`	DSA v2

Secure Shell supports two versions of the Secure Shell protocol: the original version 1 and the more secure version 2. Version 2 also amends some of the basic security design flaws of version 1. Version 1 use is discouraged, and the SSH server daemon's configuration file turns on only SSH v2 compatibility (see the `Protocol` line in `/etc/ssh/sshd_config`). Version 1 is provided only to assist users migrating to version 2.

Table 83 lists the authentication methods and local and remote host requirements.

Table 83 Authentication Methods for Secure Shell

Authentication Method	Local Host Requirements	Remote Host Requirements
Password-based (v1 or v2)	User account	User account

Table 83 Authentication Methods for Secure Shell (Continued)

Authentication Method	Local Host Requirements	Remote Host Requirements
RSA/DSA public key (v2)	User account Private key in `$HOME/.ssh/id_rsa` or `$HOME/.ssh/id_dsa` Public key in `$HOME/.ssh/id_rsa.pub` or `$HOME/.ssh/id_dsa.pub`	User account User's public key (`id_rsa.pub` or `id_dsa.pub`) in `$HOME/.ssh/authorized_keys`
RSA public key (v1)	User account Private key in `$HOME/.ssh/identity` Public key in `$HOME/.ssh/identity.pub`	User account User's public key (`identity.pub`) in `$HOME/.ssh/authorized_keys`
`.rhosts` with RSA (v1)	User account	User account Local host name in `/etc/hosts.equiv` `/etc/shosts/equiv` `$HOME/.rhosts` or `$home/.shosts`
`.rhosts` only (v1 or v2)	User account	User account Local host name in `/etc/hosts.equiv` `/etc/shosts/equiv` `$HOME/.rhosts` or `$home/.shosts`

`.rhosts` provides only weak security, and SSH in the Solaris 9 Operating Environment is, by default, configured to ignore `.rhosts` completely. `.rhosts` with RSA (v1) and password-based authentication (v1 or v2) provide medium security. RSA public key (v1) and RSA/DSA public key (v2) provide strong security. Password-based authentication is the default.

Benefits of SSH

SSH provides a secure replacement for the `rsh`, `rlogin`, `rcp`, `telnet`, and `ftp` commands. It automatically tunnels X11 traffic and allows authentication with passwords, Kerberos 4 and 5, and public keypairs.

With Secure Shell, you can log in to another host securely over an unsecured network, copy files securely between two hosts, and run commands securely on the remote host.

SSH Configuration

At boot time, the `/etc/init.d/sshd` script normally starts the `sshd` Secure Shell daemon. The daemon listens for connections from clients. When the user runs the `ssh`, `scp`, or `sftp` command, a Secure Shell session begins. A new `sshd` daemon is forked for each incoming connection to handle key exchange, encryption, authentication, command execution, and data exchange with the client. The client-side configuration files and server-side configuration files determine the session characteristics. After the authentication succeeds, the user can execute commands remotely and copy data between hosts.

Configuring Secure Shell Clients

The client-side characteristics of a Secure Shell session are usually governed by the systemwide configuration file `/etc/ssh/ssh_config`, which the administrator sets up. Users can override settings in the systemwide configuration file with the configuration in the user's `$HOME/.ssh_config` file. In addition, the user can override both configuration files on the command line.

The default `/etc/ssh/ssh_config` file is shown below.

```
# Copyright (c) 2001 by Sun Microsystems, Inc.
# All rights reserved.
#
# ident   "@(#)ssh_config   1.2   01/10/08 SMI"
#
# This file provides defaults for ssh(1).
# The values can be changed in per-user configuration files $HOME/.ssh/config
# or on the command line of ssh(1).

# Configuration data is parsed as follows:
#  1. command line options
#  2. user-specific file
#  3. system-wide file /etc/ssh/ssh_config
#
# Any configuration value is only changed the first time it is set.
# host-specific definitions should be at the beginning of the
# configuration file, and defaults at the end.

# Example (matches compiled in defaults):
#
# Host *
#   ForwardAgent no
#   ForwardX11 no
#   PubkeyAuthentication yes
#   PasswordAuthentication yes
#   FallBackToRsh no
#   UseRsh no
#   BatchMode no
#   CheckHostIP yes
#   StrictHostKeyChecking ask
#   EscapeChar ~
```

Lines have the format keyword arguments and are case sensitive.

Table 84 lists valid keywords and their descriptions.

Table 84   Valid Keywords for the ssh_config FIle

Keyword	Description
`BatchMode`	The argument must be `yes` or `no`. If set to `yes`, passphrase/password querying is disabled. This option is useful in scripts and other batch jobs for which no user is present to supply the password.
`CheckHostIP`	If this option is set to `yes`, `ssh` additionally checks the host IP address in the `known_hosts` file. This option enables `ssh` to detect if a host key changed because of DNS spoofing. If the option is set to `no`, the check is not executed.
`Cipher`	Specify the cipher to use for encrypting the session in protocol version 1; `blowfish` and `3des` are the only valid values. Specify the ciphers allowed for protocol version 2 in order of preference. Comma-separate multiple ciphers. The default is `3des-cbc,blowfish-cbc,aes-128-cbc`.
`Compression`	Specify whether to use compression. The argument must be `yes` or `no`.
`CompressionLevel`	
	Specify the compression level to use if compression is enabled. The argument must be an integer from `1` (fast) to `9` (slow, best). The default level is `6`, which is good for most applications.
`ConnectionAttempts`	
	Specify the number of tries (one per second) to make before falling back to `rsh` or exiting. The argument must be an integer. This option can be useful in scripts if the connection sometimes fails.

Table 84 Valid Keywords for the ssh_config FIle (Continued)

Keyword	Description
`DSAAuthentication`	
	Specify whether to try DSA authentication. The argument to this keyword must be `yes` or `no`. DSA authentication is tried only if a DSA identity file exists. Note that this option applies to protocol version 2 only.
`EscapeChar`	Set the escape character. The default is tilde (~). You can also set the escape character on the command line. The argument should be a single character, ^, followed by a letter, or `none` to disable the escape character entirely (making the connection transparent for binary data).
`FallBackToRsh`	
	Specify that if connecting with `ssh` fails because of a connection-refused error (there is no `sshd` listening on the remote host), automatically use `rsh`(1) instead (after a suitable warning about the session being unencrypted). The argument must be `yes` or `no`.
`ForwardAgent`	Specify whether to forward the connection to the authentication agent (if any) on the remote system. The argument must be `yes` or `no`. The default is `no`.
`ForwardX11`	Specify whether X11 connections are automatically redirected over the secure channel and `DISPLAY` set. The argument must be `yes` or `no`. The default is `no`.
`GatewayPorts`	Specify whether remote hosts are allowed to connect to local forwarded ports. The argument must be `yes` or `no`. The default is `no`.
`GlobalKnownHostsFile`	
	Specify a file to use instead of `/etc/ssh_known_hosts`.
`Host`	Restrict the following declarations (up to the next Host keyword) to be those only for hosts that match one of the patterns given after the keyword. You can use asterisk (`*`) and question mark (`?`) as wildcards in the patterns. To provide global defaults for all hosts, use a single `*`. The host is the *`hostname`* argument given on the command line (that is, the name is not converted to a canonicalized host name before matching).

Table 84 Valid Keywords for the ssh_config FIle (Continued)

Keyword	Description
`HostName`	Specify the real host name to log in to. You can use this option to specify nicknames or abbreviations for hosts. Default is the name given on the command line. Numeric IP addresses are also permitted (both on the command line and in `HostName` specifications).
`IdentityFile`	Specify the file from which the user's RSA authentication identity is read. The default is `$HOME/.ssh/identity` in the user's home directory. Additionally, any identities represented by the authentication agent are used for authentication. The file name can use the tilde (~) syntax to refer to a user's home directory. You can specify multiple identity files in configuration files; all of these identities are tried in sequence.
`IdentityFile2`	
	Specify the file from which the user's DSA authentication identity is read. The default is `$HOME/.ssh/id_dsa` in the user's home directory. The file name can use the tilde (~) syntax to refer to a user's home directory. You can have multiple identity files specified in configuration files; all of these identities are tried in sequence.
`KeepAlive`	Specify whether the system should send keepalive messages to the other side. If the messages are sent, death of the connection or crash of one of the systems are properly noticed. However, connections die if the route is down temporarily, which can be annoying. The default is `yes` (to send keepalives), which means the client notices if the network goes down or the remote host dies. This behavior is important in scripts, and many users also want it. To disable keepalives, set the value to `no` in both the server and the client configuration files.
`LocalForward`	Specify that a TCP/IP port on the local system be forwarded over the secure channel to a given *`host:port`* from the remote system. The first argument must be a port number, and the second must be *`host:port`*. You can specify multiple forwardings, and you can specify additional forwardings on the command line. Only superuser can forward privileged ports.

Table 84 Valid Keywords for the ssh_config FIle (Continued)

Keyword	Description
`LogLevel`	Specify the verbosity level used when logging messages from `ssh`. The possible values are `QUIET`, `FATAL`, `ERROR`, `INFO`, `VERBOSE`, and `DEBUG`. The default is `INFO`.
`NumberOfPasswordPrompts`	
	Specify the number of password prompts before giving up. The argument to this keyword must be an integer. The default is `3`.
`PasswordAuthentication`	
	Specify whether to use password authentication. The argument to this keyword must be `yes` or `no`. Note that this option applies to both protocol versions 1 and 2.
`Port`	Specify the port number to connect on the remote host. The default is `22`.
`Protocol`	Specify the protocol versions `ssh` should support, in order of preference. The possible values are `1` and `2`. Comma-separate multiple versions. The default is `1,2`, which means that `ssh` tries version 1 and falls back to version 2 if version 1 is not available.
`ProxyCommand`	Specify the command to use to connect to the server. The command string extends to the end of the line and is executed with `/bin/sh`. In the command string, for `%h` substitute the host name to connect, and for `%p` substitute the port. The string can be any valid command and should read from its standard input and write to its standard output. It should eventually connect an `sshd`(1M) server running on some system or execute `sshd -i` somewhere. Host key management is done by use of the `HostName` of the host being connected (defaulting to the name typed by the user). Note that `CheckHostIP` is not available for connections with a proxy command.
`RemoteForward`	
	Specify that a TCP/IP port on the remote system be forwarded over the secure channel to a given *`host:port`* from the local system. The first argument must be a port number, and the second must be *`host:port`*. You can

Table 84 Valid Keywords for the ssh_config FIle (Continued)

Keyword	Description
	specify multiple forwardings and give additional forwardings on the command line. Only superuser can forward privileged ports.
`RhostsAuthentication`	
	Specify whether to try `rhosts`-based authentication. Note that this declaration affects only the client side and has no effect whatsoever on security. Disabling `rhosts` authentication can reduce authentication time on slow connections when `rhosts` authentication is not used. Most servers do not permit `RhostsAuthentication`, because it is not secure (see `RhostsRSAAuthentication`). The argument to this keyword must be `yes` or `no`.
`RhostsRSAAuthentication`	
	Specify whether to try `rhosts`-based authentication with RSA host authentication. This authentication method is the primary one for most sites. The argument must be `yes` or `no`.
`StrictHostKeyChecking`	
	If this option is set to `yes`, `ssh` never automatically adds host keys to the `$HOME/.ssh/known_hosts` file and refuses to connect hosts whose host key has changed. This option provides maximum protection against Trojan horse attacks. However, it can be inconvenient if you do not have good `/etc/ssh_known_hosts` files installed, and you frequently connect new hosts. This option forces the user to manually add any new hosts. Normally, this option is disabled, and new hosts are added automatically to the known host files. The host keys of known hosts are verified automatically in either case. The argument must be `yes` or `no`.
`UsePrivilegedPort`	
	Specify whether to use a privileged port for outgoing connections. The argument must be `yes` or `no`. The default is `yes`. Note that setting this option to `no` turns off `RhostsAuthentication` and `RhostsRSAAuthentication`.

Table 84 Valid Keywords for the ssh_config FIle (Continued)

Keyword	Description
User	Specify the user to log in as. This option can be useful if you have different user names on different systems. Using this option means you do not need to enter the user name on the command line.
UserKnownHostsFile	
	Specify a file to use instead of `$HOME/.ssh/known_hosts`.
UseRsh	Use `rlogin` or `rsh` for this host. It is possible that the host does not support the `ssh` protocol. `ssh` immediately executes `rsh`(1). All other options (except `HostName`) are ignored if you specify this option. The argument must be `yes` or `no`.
XAuthLocation	
	Specify the location of the `xauth`(1) program. The default is `/usr/openwin/bin/xauth`.

You determine the authentication method for a client by setting one of the following keywords to `yes`.

- `DSAAuthentication`
- `PasswordAuthentication`
- `RhostsAuthentication`
- `RhostsRSAAuthentication`

Configuring Secure Shell Servers

The server-side characteristics of a Secure Shell session are usually governed by the systemwide configuration file `/etc/ssh/sshd_config`, which the administrator sets up. Users can override settings in the system-wide configuration file with the configuration in the user's `$HOME/.ssh_config` file only if the user runs his own copy of the `sshd` daemon on a nonprivileged port. In addition, the user can override both configuration files on the command line.

The default `/etc/ssh/sshd_config` file is shown below.

```
# Copyright (c) 2001 by Sun Microsystems, Inc.
# All rights reserved.
#
# ident  "@(#)sshd_config   1.3   01/10/08 SMI"
#
# Configuration file for sshd(1m)

# Protocol versions supported
```

```
#
# The sshd shipped in this release of Solaris has support for major versions
# 1 and 2.  It is recommended due to security weaknesses in the v1 protocol
# that sites run only v2 if possible. Support for v1 is provided to help sites
# with existing ssh v1 clients/servers to transition.
# Support for v1 may not be available in a future release of Solaris.
#
# To enable support for v1 an RSA1 key must be created with ssh-keygen(1).
# RSA and DSA keys for protocol v2 are created by /etc/init.d/sshd if they
# do not already exist, RSA1 keys for protocol v1 are not automatically created.

# Uncomment ONLY ONE of the following Protocol statements.

# Only v2 (recommended)
Protocol 2

# Both v1 and v2 (not recommended)
#Protocol 2,1

# Only v1 (not recommended)
#Protocol 1

# Listen port (the IANA registered port number for ssh is 22)
Port 22

# The default listen address is all interfaces, this may need to be changed
# if you wish to restrict the interfaces sshd listens on for a multi homed host.
# Multiple ListenAddress entries are allowed.

# IPv4 only
#ListenAddress 0.0.0.0
# IPv4 & IPv6
ListenAddress ::

# Port forwarding
AllowTcpForwarding no

# If port forwarding is enabled, specify if the server can bind to INADDR_ANY.
# This allows the local port forwarding to work when connections are received
# from any remote host.
GatewayPorts no

# X11 tunneling options
X11Forwarding no
X11DisplayOffset 10

# The maximum number of concurrent unauthenticated connections to sshd.
# start:rate:full see sshd(1) for more information.
# The default is 10 unauthenticated clients.
#MaxStartups 10:30:60

# Banner to be printed before authentication starts.
#Banner /etc/issue

# Should sshd print the /etc/motd file and check for mail.
# On Solaris it is assumed that the login shell will do these (eg /etc/profile).
PrintMotd no
CheckMail no

# KeepAlive specifies whether keep alive messages are sent to the client.
# See sshd(1) for detailed description of what this means.
# Note that the client may also be sending keep alive messages to the server.
KeepAlive yes

# Syslog facility and level
SyslogFacility auth
LogLevel info

#
# Authentication configuration
#

# Host private key files
# Must be on a local disk and readable only by the root user (root:sys 600).
HostKey /etc/ssh/ssh_host_rsa_key
HostKey /etc/ssh/ssh_host_dsa_key
```

```
# Default Encryption algorithms and Message Authentication codes
Ciphers   aes128-cbc,blowfish-cbc,3des-cbc
MACS      hmac-sha1,hmac-md5

# Length of the server key
# Default 768, Minimum 512
ServerKeyBits 768

# sshd regenerates the key every KeyRegenerationInterval seconds.
# The key is never stored anywhere except the memory of sshd.
# The default is 1 hour (3600 seconds).
KeyRegenerationInterval 3600

# Ensure secure permissions on users .ssh directory.
StrictModes yes

# Length of time in seconds before a client that hasn't completed
# authentication is disconnected.
# Default is 600 seconds. 0 means no time limit.
LoginGraceTime 600

# Maximum number of retries for authentication
# Default is 6. Default (if unset) for MaxAuthTriesLog is MaxAuthTries / 2
MaxAuthTries      6
MaxAuthTriesLog   3

# Are logins to accounts with empty passwords allowed.
# If PermitEmptyPasswords is no, pass PAM_DISALLOW_NULL_AUTHTOK
# to pam_authenticate(3PAM).
PermitEmptyPasswords no

# To disable tunneled clear text passwords, change PasswordAuthentication to no.
PasswordAuthentication yes

# Use PAM via keyboard interactive method for authentication.
# Depending on the setup of pam.conf(4) this may allow tunneled clear text
# passwords even when PasswordAuthentication is set to no. This is dependent
# on what the individual modules request and is out of the control of sshd
# or the protocol.
PAMAuthenticationViaKBDInt yes

# Are root logins permitted using sshd.
# Note that sshd uses pam_authenticate(3PAM) so the root (or any other) user
# maybe denied access by a PAM module regardless of this setting.
# Valid options are yes, without-password, no.
PermitRootLogin no

# sftp subsystem
Subsystem   sftp   /usr/lib/ssh/sftp-server


# SSH protocol v1 specific options
#
# The following options only apply to the v1 protocol and provide
# some form of backwards compatibility with the very weak security
# of /usr/bin/rsh.  Their use is not recommended and the functionality
# will be removed when support for v1 protocol is removed.

# Should sshd use .rhosts and .shosts for password less authentication.
IgnoreRhosts yes
RhostsAuthentication no

# Rhosts RSA Authentication
# For this to work you will also need host keys in /etc/ssh/ssh_known_hosts.
# If the user on the client side is not root then this won't work on
# Solaris since /usr/bin/ssh is not installed setuid.
RhostsRSAAuthentication no

# Uncomment if you don't trust ~/.ssh/known_hosts for RhostsRSAAuthentication.
#IgnoreUserKnownHosts yes

# Is pure RSA authentication allowed.
# Default is yes
RSAAuthentication yes
```

Lines have the format keyword arguments and are case sensitive.

Table 85 lists valid keywords and their descriptions.

Table 85 Valid Keywords and Descriptions for sshd_config

Keyword	Description
`AllowGroups`	You can follow this keyword with a space-separated list of names of groups that are allowed to log in. If specified, login is allowed only for users whose primary group matches one of the patterns. You can use asterisk (`*`) and question mark (`?`) as wildcards in the patterns. Only group names are valid; a numerical group ID is not recognized. By default, login is allowed regardless of the primary group.
`AllowTcpForwarding`	
	Specify whether TCP forwarding is permitted. The default is `yes`. Note that disabling TCP forwarding does not improve security unless users are also denied shell access, because they can always install their own forwarders.
`AllowUsers`	Follow this keyword with a space-separated list of names of users who are allowed to log in. If specified, login is allowed only for a user whose name matches one of the patterns. You can use asterisk (`*`) and question mark (`?`) as wildcards in the patterns. Only user names are valid; a numerical user ID is not recognized. By default, login is allowed regardless of the user name.
`Ciphers`	Specify the ciphers allowed for protocol version 2. Comma-separate multiple ciphers. The default is `3des-cbc,blowfish-cbc,aes-128-cbc`.
`CheckMail`	Specify whether `sshd` checks for new mail for interactive logins. The default is `no`.
`DenyGroups`	You can follow this keyword with a space-separated list of group names. Users whose primary group matches one of the patterns are not allowed to log in. You can use asterisk (`*`) and question mark (`?`) as wildcards in the patterns. Only group names are valid; a numerical group ID is not recognized. By default, login is allowed regardless of the primary group.

Table 85 Valid Keywords and Descriptions for sshd_config (Continued)

Keyword	Description
`DenyUsers`	You can follow this keyword with a space-separated list of user names. Login is disallowed for user names that match one of the patterns. You can use asterisk (`*`) and question mark (`?`) as wildcards in the patterns. Only user names are valid; a numerical user ID is not recognized. By default, login is allowed regardless of the user name.
`DSAAuthentication`	
	Specify whether DSA authentication is allowed. The default is `yes`. Note that this option applies only to protocol version 2.
`GatewayPorts`	Specify whether remote hosts are allowed to connect to ports forwarded for the client. The argument must be `yes` or `no`. The default is `no`.
`HostKey`	Specify the file containing the private RSA host key (default `/etc/ssh_host_key`) used by SSH protocols. The `/etc/ssh/sshd_config` file provides two `HostKey` lines, one for the v3 RSA key (`/etc/ssh/ssh_host_rsa_key`) and one for the v2 DSA key (`/etc/ssh/ssh_host_dsa_key`).
`IgnoreRhosts`	Specify that `.rhosts` and `.shosts` files are not used in authentication. `/etc/hosts.equiv` and `/etc/shosts.equiv` are still used. The default is `yes`.
`IgnoreUserKnownHosts`	
	Specify whether `sshd` ignores the user's `$HOME/.ssh/known_hosts` file during `RhostsRSAAuthentication`. The default is `no`.
`KeepAlive`	Specify whether the system should send keepalive messages to the other side. If they are sent, death of the connection or crash of one of the systems is properly noticed. However, connections die if the route is down temporarily, which can be annoying. On the other hand, if keepalives are not sent, sessions can hang indefinitely on the server, leaving "ghost" users and consuming server resources. The default is `yes` (to send keepalives), and the server notices if the network goes down or the client host reboots. This option avoids infinitely hanging sessions.

Table 85 Valid Keywords and Descriptions for sshd_config (Continued)

Keyword	Description
	To disable keepalives, set the value to `no` in both the server and the client configuration files.
`KeyRegenerationInterval`	
	Automatically regenerate the server key after *n* seconds (if it has been used). Regeneration prevents decryption of captured sessions by someone later breaking into the system and stealing the keys. The key is never stored anywhere. If the value is `0`, the key is never regenerated. The default is `3600` (seconds).
`ListenAddress`	
	Specify the local address on which `sshd` listens. The default is to listen to all local addresses. Multiple options of this type are permitted. Additionally, the `Ports` options must precede this option.
`LoginGraceTime`	
	Disconnect the server after *n* seconds if the user has not successfully logged in. If the value is `0`, there is no time limit. The default is `600` (seconds).
`LogLevel`	Specify the verbosity level used when messages from `sshd` are logged. The possible values are `QUIET`, `FATAL`, `ERROR`, `INFO`, `VERBOSE`, and `DEBUG`. The default is `INFO`. Logging with level `DEBUG` violates the privacy of users and is not recommended.
`MaxStartups`	Specify the maximum number of concurrent, unauthenticated connections to the `sshd` daemon. Additional connections are dropped until authentication succeeds or the `LoginGraceTime` expires for a connection. The default is `10`. Alternatively, you can enable random early drop by specifying the three colon-separated values *`start:rate:full`* (for example, `10:30:60`). For this example, `sshd` refuses connection attempts with a probability of 30 percent (*`rate`*`/100`) when there are currently `10` (from the start field) unauthenticated connections. The probability increases linearly and all connection attempts are refused if the number of unauthenticated connections reaches `60` (*`full`*).

Table 85 Valid Keywords and Descriptions for sshd_config (Continued)

Keyword	Description
PasswordAuthentication	
	Specify whether password authentication is allowed. The default is `yes`. Note that this option applies to both protocol versions 1 and 2.
PermitEmptyPasswords	
	When password authentication is allowed, it specifies whether the server allows login to accounts with empty password strings. The default is `no`.
PermitRootLogin	
	Specify whether root can log in with `ssh`. The argument must be one of `yes`, `without-password`, or `no`. The default is `no`. When this options is set to `without-password`, root can log in only through public key authentication; passwords are ignored. Note that the Secure Shell is integrated with the PAM subsystem. You can configure PAM to deny login access to root regardless of this setting. Root login with RSA authentication when the command option is specified is allowed regardless of the value of this setting. This setting might be useful for taking remote backups even if root login is normally not allowed.
Port	Specify the port number at which `sshd` listens. The default is `22`. You can specify multiple options of this type.
PrintMotd	Specify whether `sshd` displays the contents of `/etc/motd` when a user logs in interactively. (On some systems, `/etc/motd` is also displayed by the shell or a shell startup file, such as `/etc/profile`.) The default is `yes`.
Protocol	Specify the protocol versions `sshd` supports. The possible values are `1` and `2`. You must comma-separate multiple versions. The default is `2`.
RhostsAuthentication	
	Specify whether authentication with `rhosts` or `/etc/hosts.equiv` files is sufficient. Normally, you should not permit this method because it is insecure. Use

Table 85 Valid Keywords and Descriptions for sshd_config (Continued)

Keyword	Description
	`RhostsRSAAuthentication` instead because it performs RSA-based host authentication in addition to normal `rhosts` or `/etc/hosts.equiv` authentication. The default is `no`.
`RhostsRSAAuthentication`	
	Specify whether `rhosts` or `/etc/hosts.equiv` authentication together with successful RSA host authentication is allowed. The default is `no`.
`RSAAuthentication`	
	Specify whether pure RSA authentication is allowed. The default is `yes`. Note that this option applies only to protocol version 1.
`ServerKeyBits`	
	Define the number of bits in the server key. The minimum value is `512`, and the default is `768`.
`StrictModes`	Specify whether `sshd` checks file modes and ownership of the user's files and home directory before accepting login. This behavior is normally desirable because novices sometimes accidentally leave their directory or files world-writable. The default is `yes`.
`Subsystem`	Configure an external subsystem (for example, a file transfer daemon). Arguments should be a subsystem name and a command to execute on subsystem request. The command `sftp-server`(1M) implements the `sftp` file transfer subsystem. By default, no subsystems are defined. Note that this option applies only to protocol version 2.
`SyslogFacility`	
	Give the facility code used when messages from `sshd` are logged. The possible values are `DAEMON`, `USER`, `AUTH`, `LOCAL0`, `LOCAL1`, `LOCAL2`, `LOCAL3`, `LOCAL4`, `LOCAL5`, `LOCAL6`, and `LOCAL7`. The default is `AUTH`.
`X11DisplayOffset`	
	Specify the first display number available for `sshd` X11 forwarding. This option prevents `sshd` from interfering with real X11 servers. The default is `10`.

Table 85 Valid Keywords and Descriptions for sshd_config (Continued)

Keyword	Description
X11Forwarding	
	Specify whether X11 forwarding is permitted. The default is no. Note that disabling X11 forwarding does not improve security in any way, because users can always install their own forwarders.
XAuthLocation	
	Specify the location of the xauth(1) program. The default is /usr/openwin/bin/xauth.

You determine the authentication method for a server by setting one of the following keywords to yes.

- DSAAuthentication
- PasswordAuthentication
- RhostsAuthentication
- RhostsRSAAuthentication
- RSAAuthentication

X11 Forwarding

The X Window system (also known as X11) lets you log in to a remote system, run X11 programs on that system, and, if the X11 server program running on your local system controls the monitor at which you are working, displays the X11 program output there. If you use the Solaris rsh, rlogin, or telnet commands without SSH to log in to that remote system, you need to perform the following manual steps for this process to work properly.

- Before you log in to the remote system by using rsh, rlogin, or telnet, run xhost +remote-system to give the remote system permission to send X11 datastreams from any X11 program to your local X11 server program.
- Once you log in to the remote system, set the DISPLAY environment variable to indicate the X11 server program to which all X11 client programs send their data streams (in this case, your X11 server program).

The Secure Shell automates the X11 forwarding process and secures it by encrypting the X11 datastreams as they pass over the network. Sun disables the X11 forwarding feature by default. You must enable it for both the client and server by making the following changes to both the local and remote systems.

In the /etc/ssh/ssh_config file, change

```
# ForwardX11 no
```

to

```
Forwardx11 yes
```

Be sure to remove the # comment character at the beginning of the line as well.

This change takes effect the next time you run ssh.

In the /etc/ssh/sshd_config file, change

```
X11Forwarding no
```

to

```
X11Forwarding yes
```

To make this change take effect, restart the Secure Shell daemon on both systems by running

```
# /etc/init.d/sshd stop
# /etc/init.d/sshd start
```

You can now use ssh to log in to the remote system. Run an X11 client such as xterm to verify that X11 Forwarding works properly. If the xterm window is displayed on your local X11 display, then everything is working.

Public Key Authentication with the Secure Shell

The examples in this section assume that you have a single home directory that is automounted on every system under your control at /home/*username*. By convention, this directory is referenced with the environment variable $HOME. If you have a unique account and home directory on every system that you log in to, then a reference to the $HOME/.ssh/ directory implies that this directory exists on every system (that is, you need to copy the contents of that directory to each unique home directory you have before the procedures in this section work).

The Secure Shell uses regular password authentication by default; that is, when you use ssh to log in to a remote system, you are asked to enter a password to authenticate your account identity. Once you enter the correct password, you are allowed to log in. The Secure Shell also allows you to use public key authentication instead of password authentication. Public key authentication has the following benefits.

- When set up properly, you can log in to a remote host without entering a password. That means you get all the benefits that .rhosts previously gave you without any of the liabilities.
- It is much more difficult to break a public key's passphrase than your regular UNIX password. Accounts are better protected when you disallow the use of the rlogin, rsh, telnet, rcp, and ftp commands at your site and use only the Secure Shell commands.

The first step in using public key authentication is to generate one or more public/private keypairs with the ssh-keygen(1) command. Refer to the ssh-keygen(1) manual page for detailed information on the different types of keypairs you can generate.

NOTE. You can have more than one keypair, and you can use each for a different purpose. For example, you can have one keypair for logging in as the root user on the Solaris systems on the manufacturing floor, and another for logging in as the backup administrator on the backup server, and so on. By default, the keypairs are stored in your $HOME/.ssh/ *directory when you create them.*

The following examples create several keypairs. In these examples, $HOME is /home/gmarler.

The following example creates a default 1024-bit RSA keypair. This keypair is treated as your default RSA keypair for use with the SSH v2 protocol. The public key is stored at $HOME/.ssh/id_rsa.pub, and the private key is stored at $HOME/.ssh/id_rsa.

```
[ns3:/home/gmarler]
$ ssh-keygen
Enter file in which to save the key(/home/gmarler/.ssh/id_rsa):
Generating public/private rsa key pair.
Enter passphrase(empty for no passphrase): Enter passphrase.
Enter same passphrase again: Enter passphrase again.
Your identification has been saved in /home/gmarler/.ssh/id_rsa.
Your public key has been saved in /home/gmarler/.ssh/id_rsa.pub.
The key fingerprint is:
md5 1024 d1:88:b9:5c:f1:28:0f:dd:6e:f3:fc:ea:af:3c:21:ed gmarler@ns3
```

The following example creates a 768-bit DSA keypair. This keypair is treated as your default DSA keypair for use with the SSH v2 protocol. The public key is stored as $HOME/.ssh/id_dsa.pub, and the private key is stored as $HOME/.ssh/id_dsa.

```
[ns3:/home/gmarler]
$ ssh-keygen -b 768 -t dsa
Enter file in which to save the key(/home/gmarler/.ssh/id_dsa):
Generating public/private dsa key pair.
Enter passphrase(empty for no passphrase): Enter passphrase.
Enter same passphrase again: Enter passphrase again.
Your identification has been saved in /home/gmarler/.ssh/id_dsa.
Your public key has been saved in /home/gmarler/.ssh/id_dsa.pub.
The key fingerprint is:
md5 768 1d:f0:f5:d5:bd:35:b1:ac:9a:2a:b9:7f:95:14:02:f0 gmarler@ns3
```

The following example creates a 512-bit RSA1 keypair (for use only with SSH protocol v1). This keypair is treated as your default RSA keypair for use with SSH v1 protocol—SSH v1 supported only the use of RSA keys. The public key is stored at `$HOME/.ssh/identity.pub`, and the private key is stored at `$HOME/.ssh/identity`.

```
[ns3:/home/gmarler]
$ ssh-keygen -b 512 -t rsa1
Enter file in which to save the key(/home/gmarler/.ssh/identity):
Generating public/private rsa1 key pair.
Enter passphrase(empty for no passphrase): Enter passphrase.
Enter same passphrase again: Enter passphrase again.
Your identification has been saved in /home/gmarler/.ssh/identity.
Your public key has been saved in /home/gmarler/.ssh/identity.pub.
The key fingerprint is:
md5 512 bb:e2:c5:25:4d:d1:89:23:83:9e:89:51:4f:d0:5b:86 gmarler@ns3
```

The following example creates a 2048-bit RSA keypair for use when you log in to remote systems as the root user.

```
[ns3:/home/gmarler]
$ ssh-keygen -b 2048 -f $HOME/.ssh/rootkey -C "Root Admin Keypair"
Generating public/private rsa key pair.
Enter passphrase(empty for no passphrase): Enter passphrase.
Enter same passphrase again: Enter passphrase again.
Your identification has been saved in /home/gmarler/.ssh/rootkey.
Your public key has been saved in /home/gmarler/.ssh/rootkey.pub.
The key fingerprint is:
md5 2048 44:e0:26:4d:6a:93:6c:5c:88:ac:0a:87:e1:d6:ad:8b Root Admin Keypair
```

The following example creates a 1024-bit RSA keypair, with no passphrase, for use in automated batch jobs to remote systems.

You would use this keypair in `cron` jobs or scripts that use `ssh`.

NOTE. Because the keypair is not protected by a passphrase, it is only as secure as the permissions on the files you store it in.

```
[ns3:/home/gmarler]
$ ssh-keygen -b 1024 -f $HOME/.ssh/nopasskey -C "Batch Jobs (no passphrase)"
Generating public/private rsa key pair.
Enter passphrase(empty for no passphrase): Press Return.
Enter same passphrase again: Press Return.
Your identification has been saved in /home/gmarler/.ssh/nopasskey.
Your public key has been saved in /home/gmarler/.ssh/nopasskey.pub.
The key fingerprint is:
```

```
md5 1024 21:56:cb:8e:fb:1f:d1:1c:14:50:f2:88:09:f7:39:93 Batch Jobs (no
  passphrase)
```

Changing the Passphrase of a Private Key

Once you create keypairs, you can manipulate them in various ways. One thing you may want to do fairly often is to change the passphrase on a keypair. The following example changes the passphrase on the 2048-bit RSA keypair created in one of the previous examples.

```
[ns3:/home/gmarler]
$ ssh-keygen -p -f $HOME/.ssh/rootkey
Enter old passphrase: Enter old passphrase.
Key has comment 'rsa w/o comment'
Enter new passphrase(empty for no passphrase): Enter new passphrase.
Enter same passphrase again: Enter new assphrase again.
Your identification has been saved with the new passphrase.
```

Using the Public Key in Each Keypair

The public key in each keypair is not used by the Secure Shell client. It is used by `sshd` on a remote host whenever you try to use `ssh` to log in to that remote host. But how does `sshd` on the remote host get access to your public key?

When you use `ssh` to log in to a remote host, `ssh` on your local host contacts `sshd` on the remote host and tells `sshd` which user you want to log in as. `sshd` then looks into the `.ssh` subdirectory of that user's home directory for the `authorized_keys` file. If any of the public keys stored in that file match the private key you told `ssh` to use when logging in to the remote host, the Secure Shell grants you access to that account.

The following example logs you in to a remote host as yourself with public key authentication.

In this case, you're logging in as yourself, so you need to append one of your public keys into your `$HOME/.ssh/authorized_keys` file. For this example, assume that you are the user `gmarler` and use the key generated in the first example above.

```
[ns3:/home/gmarler]
$ cat $HOME/.ssh/id_rsa.pub >>$HOME/.ssh/authorized_keys
```

Now you can try to log in to another host (that has the same home directory automounted) with the private key (specifying it with the `-i` option to `ssh`) that matches the public key you appended to the `authorized_keys` file.

```
[ns3:/home/gmarler]
$ ssh -i $HOME/.ssh/id_rsa ns1.gmarler.com
Enter passphrase for key '/home/gmarler/.ssh/id_rsa': Enter key passphrase.
Last login: Thu Oct 10 18:57:07 2002 from dhcp101.gmarler
Sun Microsystems Inc.   SunOS 5.8       Generic February 2000
Sun Microsystems Inc.   SunOS 5.8       Generic February 2000
Agent pid 17661
[ns1.gmarler.com:/home/gmarler]
 $
```

NOTE. The passphrase you are asked for is NOT your login password, but the passphrase entered for the private key when the keypair was created (or last changed).

The following example logs in to a remote host as the root user with public key authentication.

In this case, you're trying to log in to a remote host as the root user, so you need to find some way to first log in to that host as root, then append the specific public key you want to use to that root's `authorized_keys` file (located at `/.ssh/authorized_keys` on that host). This time, use the key generated specifically for this purpose in the fourth example above.

NOTE. At this point you're already logged in to the remote host as root.

```
[ns1.gmarler.com:/]
# cat /home/gmarler/.ssh/rootkey.pub >>/.ssh/authorized_keys
[ns1.gmarler.com:/]
# exit
```

NOTE. Now you're back on your original system as the user `gmarler`.

```
[ns3:/home/gmarler]
$ ssh -i $HOME/.ssh/rootkey ns1.gmarler.com -l root
Enter passphrase for key '/home/gmarler/.ssh/rootkey': Enter key passphrase.
Last login: Thu Oct 10 23:15:47 2002 from ns3
Sun Microsystems Inc.   SunOS 5.8       Generic February 2000
Sun Microsystems Inc.   SunOS 5.8       Generic February 2000
[ns1.gmarler.com:/]
 #
```

Private Keys and Passphrases

You've probably noticed that a passphrase is usually applied to the private key in each keypair. You apply the passphrase to the private key so that if someone happens to steal your private keys (you don't care if someone takes your public keys; in fact, you want everyone to have them), the thief can't use them. Why not? Because each private key is encrypted with the passphrase you put on it and is useless until it is decrypted.

But, you have to enter a passphrase before using each private key to log in to a remote host, right? That would be true if you used each private key manually, as has been done so far. But that's where the ssh-agent program comes in.

The ssh-agent The ssh-agent command has a simple and elegant purpose: it stores one or more of your decrypted private keys in memory so that ssh can use them without prompting you for the passphrase every time you use them. And, if you load all of your private keys into ssh-agent, ssh tries them all in sequence until it finds one that works. You don't have to specify a particular private key on the command line.

How do you use ssh-agent? Each user must configure his login environment to properly start and stop this program for every shell he invokes. The following example shows the necessary changes to $HOME/.profile if you use the sh, ksh, or bash shells.

```
# Set up SSH-Agent
if [ "$SSH_AUTH_SOCK" = "" -a -f /bin/ssh-agent ]; then
  eval `/bin/ssh-agent`
fi

# Kill the SSH-Agent when you log out…
trap '
   test -n "$SSH_AGENT_PID" && eval `/bin/ssh-agent -k`
' 0
```

The following example shows the changes needed to $HOME/.login and $HOME/.logout if you used the csh or tcsh shells.

```
$HOME/.login:
# Start SSH-Agent
eval `/bin/ssh-agent -c`

$HOME/.logout:
# Kill SSH-Agent
if ( "$SSH_AGENT_PID" != "" ) then
  eval `/bin/ssh-agent -k`
endif
```

Once you make these changes and log out and back in, each shell started inherits the environment variable settings that ssh-agent sets up (with the eval command) so that ssh knows how to communicate with ssh-agent. Also, the program is terminated whenever you log out, so you don't have hundreds of separate ssh-agent programs cluttering up the system.

Now that ssh-agent has been set up and automatically starts every time you log in, you need to know how to decrypt and load your private keys into it.

ssh-add You use ssh-add to decrypt and load each private key into your ssh-agent. The following example loads all the private keys you generated earlier. You can load the first three default identity keys (RSA, DSA, RSA1) just by running the ssh-add command with no arguments, as shown in the following example.

```
[ns3:/home/gmarler]
$ ssh-add
Enter passphrase for gmarler@ns3: Enter passphrase.
Identity added: /home/gmarler/.ssh/identity(gmarler@ns3)
Identity added: /home/gmarler/.ssh/id_rsa(/home/gmarler/.ssh/id_rsa)
Identity added: /home/gmarler/.ssh/id_dsa(/home/gmarler/.ssh/id_dsa)
```

NOTE. This example worked this way only because the private keys all had the same passphrase. If they did not, then you would have to enter each passphrase when prompted.

Now load `rootkey` and `nopasskey`, as shown in the following example.

```
[ns3:/home/gmarler]
$ ssh-add $HOME/.ssh/rootkey
Enter passphrase for /home/gmarler/.ssh/rootkey: Enter passphrase.
Identity added: /home/gmarler/.ssh/rootkey(/home/gmarler/.ssh/rootkey)
[ns3:/home/gmarler]
 $ ssh-add $HOME/.ssh/nopasskey
Identity added: /home/gmarler/.ssh/nopasskey(/home/gmarler/.ssh/nopasskey)
```

Notice that `nopasskey` did not prompt for a passphrase because there is no passkey. It was simply loaded into the `ssh-agent`.

You can see which keys are loaded into this particular `ssh-agent` with the `ssh-add -l` command.

```
[ns3:/home/gmarler]
$ ssh-add -l
md5 512 bb:e2:c5:25:4d:d1:89:23:83:9e:89:51:4f:d0:5b:86 gmarler@ns3(RSA1)
md5 1024 d1:88:b9:5c:f1:28:0f:dd:6e:f3:fc:ea:af:3c:21:ed
  /home/gmarler/.ssh/id_rsa(RSA)
md5 768 1d:f0:f5:d5:bd:35:b1:ac:9a:2a:b9:7f:95:14:02:f0
  /home/gmarler/.ssh/id_dsa(DSA)
md5 2048 44:e0:26:4d:6a:93:6c:5c:88:ac:0a:87:e1:d6:ad:8b
  /home/gmarler/.ssh/rootkey(RSA)
md5 1024 21:56:cb:8e:fb:1f:d1:1c:14:50:f2:88:09:f7:39:93
  /home/gmarler/.ssh/nopasskey(RSA)
```

You have now resolved the problem of having to manually enter the passphrase each time you use `ssh`. Because you now have the rootkey loaded in the `ssh-agent`, try logging into the remote system as root again.

```
[ns3:/home/gmarler]
 $ ssh ns1.gmarler.com -l root
Last login: Thu Oct 10 23:16:11 2002 from ns3
Sun Microsystems Inc.   SunOS 5.8       Generic February 2000
Sun Microsystems Inc.   SunOS 5.8       Generic February 2000
[ns1.gmarler.com:/]
 #
```

Presto! No need to enter a passphrase again (except when you first log in to your account).

The Secure Shell Commands

The following section discusses the `ssh`, `scp`, and `sftp` commands.

NOTE. The examples for these commands use public key authentication, discussed above, instead of password authentication. You won't see the commands prompting for passwords here. If you don't set up public key authentication, then you will be prompted for your account's password.

ssh The `ssh` command is a secure replacement for `rlogin`, `rsh`, and `telnet`. It takes the same parameters as `rlogin` and `rsh` (and many more), so migration to this tool is easy.

The following example logs in to a remote host as the root user.

```
[ns3:/home/gmarler]
$ ssh ns1.gmarler.com -l root
Last login: Thu Oct 10 23:51:09 2002 from ns3
Sun Microsystems Inc.   SunOS 5.8       Generic February 2000
Sun Microsystems Inc.   SunOS 5.8       Generic February 2000
[ns1.gmarler.com:/]
 #
```

The following example creates a `tar` archive datastream of the `./src` directory and transmits it to another host (by logging in to that host as the current user with `ssh`) to be extracted in the `/tmp` directory.

```
[ns3:/home/gmarler]
$ tar cf - ./src | ssh ns1.gmarler.com "(cd /tmp; tar xf -)"
[ns3:/home/gmarler]
 $
```

scp The `scp` command is a secure replacement for the `rcp` command. It takes parameters similar to those of `rcp`, but is more flexible. The following examples show some ways to use the `scp` command.

The following example copies the `connect.sql` file from the current directory to the `/tmp` directory on host `ns1.gmarler.com`, as the user `gmarler`.

```
[ns3:/home/gmarler]
$ scp connect.sql gmarler@ns1.gmarler.com:/tmp
connect.sql          100% |*****************************|    49       00:00
```

The following example logs in to host `ns1.gmarler.com` as user `gmarler` and copies the file `/tmp/connect.sql` to the `/tmp` directory on the local system.

```
[ns3:/home/gmarler]
$ scp gmarler@ns1.gmarler.com:/tmp/connect.sql /tmp
connect.sql          100% |*****************************|    49       00:00
```

The following example recursively copies the `./bin/` directory on the local system to the `/tmp/bin` directory on system `ns1.gmarler.com`, as user `gmarler`.

```
[ns3:/home/gmarler]
$ scp -r bin/ gmarler@ns1.gmarler.com:/tmp/bin
ksh                  100% |*****************************|  1609 KB    00:03
patch                100% |*****************************|   349 KB    00:00
```

The following example logs in to host `ns1.gmarler.com` as the root user and copies the `/etc/passwd` file to the `/tmp` directory on the local system.

```
[ns3:/home/gmarler]
$ scp root@ns1.gmarler.com:/etc/passwd /tmp
passwd               100% |*****************************|   931       00:00
```

sftp The `sftp` command is a secure replacement for the `ftp` command. It takes parameters similar to those of `ftp`, but is more flexible.

The following example uses `sftp` to connect to the host `ns1.gmarler.com` as the current user, changes to the `/tmp` directory on the local system, and downloads `connect.sql` from that system to `/tmp` on the local system.

```
[ns3:/home/gmarler]
$ sftp ns1.gmarler.com
Connecting to ns1.gmarler.com...
sftp > lcd /tmp
sftp > lpwd
Local working directory: /tmp
sftp > get /home/gmarler/connect.sql
sftp > quit
```

The following example uses `sftp` to connect to the host `ns1.gmarler.com` as the root user, changes to the `/tmp` directory on the local system, changes to the `/etc` directory on the remote system, and downloads the `passwd` file.

```
[ns3:/home/gmarler]
$ sftp root@ns1.gmarler.com
Connecting to ns1.gmarler.com...
sftp > lcd /tmp
sftp > lpwd
Local working directory: /tmp
sftp > cd /etc
sftp > pwd
Remote working directory: /etc
sftp > get passwd
sftp > quit
```

Common Administrative Uses for the Secure Shell

This section describes two common uses for SSH.

Transferring Files Between Systems Securely Quite often, you need to move files between systems. You can do so securely by using `ssh` instead of `rsh`. The following example copies the home directory of the user `gmarler` from system `ns3.gmarler.com` to `ns1.gmarler.com`. This action is done as the root user on `ns3.gmarler.com`, using public key authentication.

```
[ns3:/home/gmarler]
# cd /home
[ns3:/home]
# tar cf - gmarler | ssh ns1.gmarler.com -l root "(cd /home; tar xf -)"
```

Secure Root Login Without Allowing Passwords You've already seen how to use public key authentication to allow authorized system administrators to log in to remote systems as the root user. This section describes how to make public key authentication the *only* way a user can log in to a system as the root user. To force such behavior, edit the `/etc/ssh/sshd_config` file on all systems and change the following line

```
PermitRootLogin yes
```

to

```
PermitRootLogin without-password
```

Once you have made the changes, restart the `sshd` daemon on each system with the following commands.

```
# /etc/init.d/sshd stop
# /etc/init.d/sshd start
```

Secure Shell on Pre-Solaris 9 Releases

The Secure Shell provided with Solaris 9 is a Sun-supported port of OpenSSH. If you want to use the Secure Shell on releases of Solaris before Solaris 9, go to the `http://www.openssh.com/` Web site, download the source code, compile it, and install it on your pre-Solaris 9 systems.

For More Information

For more information, refer to *SSH, The Secure Shell: The Definitive Guide*, by Daniel J. Barrett and Richard E. Silverman, O'Reilly & Associates, Inc., 2001.