Sun Network Services

This chapter introduces the Sun Network services. We describe the services currently available, and define some terms in the network environment.

Following that, we introduce and explain each of the services now available for the Sun UNIX workstation: network file system service and yellow pages service. Within each of these sections you will find information about periodic maintenance and trouble-shooting for the service under discussion.

While some of this material tends to be theoretical, its specific implications will be seen again and again as you become familiar with system administration. For example, if you run the yellow pages, you must understand that some typical UNIX procedures have changed in the yellow pages environment. That is also true if you use the network file system. This chapter covers only those aspects of network services necessary for performing the duties of system administration. For a complete theoretical overview, see *The Network Services Guide*.

At the end of this chapter we explain how to add new users and new client machines at your site, one of the first and most important duties of system administration.

.ft

1. Introduction And Terminology

1.1. Networking Models

There are many ways to make computers and networks interface transparently. The two major ones are the distributed operating system approach and the network services approach.

A distributed operating system allows the network software designer to make grand assumptions about the other machines on the network. Two of these assumptions are usually: that the remote piece of hardware is identical to the local hardware, and that the remote and local machines are running identical software. These assumptions allow a quick and simple implementation of a network system in an environment limited to specific hardware and software. This type of distributed operating system is, by design, closed. That is to say, it's very difficult to integrate new hardware or software into a closed network environment, unless it comes from the vendor of that network system. A closed network system forces a customer to return to one vendor for solutions to all computing needs.

On the other hand, the Sun network is not closed. Sun bases its networking on network services. Network service is made up of remote programs composed of remote procedures called from the network. Optimally, a remote procedure computes results based entirely on it's own parameters. Thus, the procedure (and therefore, the network service) is not tied to any particular operating system or hardware. The design of the Sun network makes it possible for a variety of machines and software to talk to the network service, enabling the Sun Workstation to talk to various types of computers.

As you might expect, the Sun network services are more complex in design and implementation than a closed distributed operating system. Since remote procedures are operating-system independent and hardware independent, multiple remote procedures must sometimes be called in the Sun environment, where a single transaction might suffice in a closed system.

1.2. Terminology

The bulk of this chapter explains administration of two Sun network services — the Sun Network File System (NFS), and the Sun Yellow Pages (`yp`). Before discussing these two services, we define some generic concepts and terms.

Any machine that provides one of these network services is a server. A single machine may provide more than one service. In fact, a typical configuration would be for a single machine to act as both an NFS-server, and a `yp`-server.

In each of the Sun network services, servers are entirely passive. The servers wait for clients to call them, they never call the clients.

A client is any entity that accesses a network service. We use the term entity because the thing doing the accessing may be an actual machine or simply a UNIX process generated by a piece of software.

The degree to which clients are bound to their server varies with each of the Sun network services. For example, in nd service the nd client is strictly bound to the nd server because the server supplies a private area of a large disk for the exclusive use of each client. At the other extreme, a Sun yp client binds randomly to one of the yp servers by broadcasting a request. At any point, the yp client may decide to broadcast for a new server. The Sun NFS stands somewhere in the middle. An NFS client may choose to mount file systems from any number of servers at any time.

In all cases, however, the client initiates the binding. The server completes the binding subject to access control rules specific to each service. Since most network administration problems occur at bind time, a system administrator should know how a client binds to a server and what (if any) access control policy each server uses.

1.3. UNIX Meets Sun Network Services

Unlike many recently marketed distributed operating systems, UNIX was originally designed without knowledge that networks existed. This ‘‘networking ignorance’’ presents three impediments to linking UNIX with currently available high performance networks:

1) UNIX was never designed to yield to a higher authority (like a network authentication server) for critical information or services. As a result some UNIX semantics are hard to maintain ‘‘over the net’’. For example, trusting user id 0 (root) is not always a good idea.

2) Some UNIX execution semantics are difficult. For example, UNIX allows a user to remove an open, yet the file does not disappear until closed by everyone. In a network environment a client UNIX machine may not own an open file. Therefore, a server may remove a client’s open file.

3) When a UNIX machine crashes, it takes all its applications down with it. When a network node crashes (whether client or server), it should not drag all of its bound neighbors down. The treatment of node failure on a network raises difficulties in any system and is especially difficult in the UNIX environment. Sun has implemented a system of ‘‘stateless’’ protocols to circumvent the problem of a crashing server dragging down its bound clients. Stateless here means that a client is independently responsible for completing work, and that a server need not remember anything from one call to the next. In other words, the server keeps no state. With no state left on the server, there is no state to recover when the server crashes and comes back up. So, from the client’s point of view, a crashed server appears no different than a very slow server.

In implementing UNIX over the network, Sun attempted to remain compatible with UNIX whenever possible. However, certain incompatibilities have been introduced; they are typically of two kinds. First those issues that would make a networked UNIX evolve into a distributed operating system, rather than a collection of network services. And second, those issues that would make crash recovery extremely difficult from both the implementation and administration point of view.

All incompatibilities are documented in the appropriate sections of this administration manual.

1.4. A Hint About Debugging UNIX In The Network Environment

When you cannot get something done that involves Sun network services, the problem probably lies in one of the following four areas. They are listed here with the most likely problem first.

1) The Sun network access control policies do not allow the operation, or architectural constraints prevent the operation.

2) The client software or environment is broken.

3) The server software or environment is broken.

4) The network is broken.

The following sections present specific instructions on how to check for these causes of failure in the NFS and `yp` environments.

2. NFS The Network File System Service

We begin with an explanation of some NFS terms and concepts. Then we describe how to create an NFS server that exports file systems, and how to mount and utilize remote file systems. Following that a section on debugging the NFS explains what to do when problems occur. Finally, some cautionary words about incompatibilities between NFS files and normal UNIX files.

2.1. What Is The NFS Service

The NFS enables users to share file systems over the network. A client may mount or unmount file systems from an NFS server machine. The client always initiates the binding to a server's file system by using the `mount` (8) command. Typically, a client mounts one or more remote file systems at startup time by placing lines like these in the file `/etc/fstab`, which `mount` reads when the system comes up:

```
titan:/usr2 /usr2 nfs rw,hard 0 0
venus:/usr/man /usr/man nfs rw,hard 0 0
```

See `fstab` (5) for a full description of the format.

Since clients initiate all remote mounts, NFS servers keep control over who may mount a file system by limiting named file systems to desired clients with an entry in the `/etc/exports` file. For example:

```
/usr/local                  # export to the world
/usr2      nixon ford reagan # export to only these machines
```

is an `/etc/exports` entry that speaks for itself. Note that pathnames given in `/etc/exports` must be the mount point of a local file system. See `exports` (5) for a full description of the format.

2.2. How The NFS Works

Two remote programs implement the NFS service — `mountd` (8) and `nfsd` (8). A client's `mount` request talks to `mountd` which checks the access permission of the client and returns a pointer to a filesystem. After the `mount` completes, access to that mount point and below goes through the pointer to the server's `nfsd` daemon using `rpc` (4). Client kernel file access requests (delayed-write and read-ahead) are handled by the `biod` (8) daemons on the client.

If this terse explanation doesn't answer all your questions, you can find more details in *The Network Services Guide*. However, you will be able to install and administer the NFS just by considering the information given here.

2.3. How To Become An NFS Server

An NFS server is simply a machine that exports a file system or systems. Here are the three steps that any machine must follow to enable it to export a file system.

1) Become super-user and place the mount-point pathname of the file system you want to export in the file `/etc/exports`. See `exports`(5) for file format details. For example, to export `/usr/src/mybin`, the export file would look like:

```
/usr/src/mybin
```

   Of course, an NFS server may only export file systems of its own.

2) As we saw above, `mountd` must be present for a remote mount to succeed. Make sure `mountd` will be available for an `rpc` call by checking the file `/etc/servers` for this line:

```
rpc     udp  /usr/etc/rpc.mountd 100005 1
```

   If it isn't there add it. For details, see `servers`(5).

3) Remote mount also needs some number (typically 4) of `nfsd`'s to execute on NFS servers. Check `/etc/rc.local` for lines like these:

```
if [ -f /etc/nfsd -a -f /etc/exports ]; then
/etc/nfsd 4 & echo -n ' nfsd'          >/dev/console
```

   Add these lines, or your own version of them, if the new NFS server's `/etc/rc.local` does not enable `nfsd`'s. You can enable `nfsd`'s without rebooting, by typing, while super-user:

```
# /etc/nfsd 4
```

After these steps, the NFS server should be able to export the named file system. Read the next section and try to remote mount.

2.4. How To Remote Mount A File System

You can mount any exported file system onto your machine, as long as you can reach its server over the network, and you are included in its `/etc/export` list for that file system. On the machine where you want to mount the file system, become super-user and type the following:

```
# mount server_name:/file_system /mount_point
```

For example, to mount the manual pages from remote machine `elvis` on my directory `/usr/elvis.man`:

```
# mount elvis:/usr/man /usr/elvis.man
```

To make sure you have mounted a file system, and mounted it where you expected to, use either `df`(1) or `mount`(8), without an argument. Each of these displays the currently mounted file systems.

Typically, you mount frequently used file systems at startup by placing an entry for them in the file `/etc/fstab`. You can also see `fstab`(5).

2.5. Typical NFS Layout

To fully explain the layout of the NFS on clients, the output from `mount` commands below shows the mounted file systems on a server, and on one of its clients — notice where the client file systems are mounted from. Following that, the output from `ls` commands shows the contents of various directories on the client machine. `lenin` is a client of `marx`:

```
marx% mount
/dev/xy0a on / type 4.2 (rw)
/dev/xy0g on /pub type 4.2 (rw)
/dev/xy0d on /usr type 4.2 (rw)
/dev/xy2g on /usr2 type 4.2 (rw)
/dev/xy2b on /usr/doctools type 4.2 (rw)

lenin% mount
/dev/nd0 on / type 4.2 (rw)
/dev/ndp0 on /pub type 4.2 (ro)
marx:/lib on /lib type nfs (rw,hard)
marx:/usr on /usr type nfs (rw,hard)
marx:/usr2 on /usr2 type nfs (rw,hard)

lenin% ls -l /
total 124
lrwxrwxrwx  1 root           8 bin -> /pub/bin
drwxr-xr-x  2 root        1536 dev
drwxr-xr-x  3 bin         1536 etc
drwxr-xr-x  2 bin          512 lib
drwxr-xr-x  2 root        4096 lost+found
drwxr-xr-x  2 bin           24 mnt
drwxr-xr-x  3 root         512 private
drwxr-xr-x  5 root         512 pub
lrwxrwxrwx  1 root          10 stand -> /stand/bin
drwxrwxrwx  2 bin          512 tmp
drwxr-xr-x 16 root         512 usr
drwxr-xr-x 34 root        1024 usr2
lrwxrwxrwx  1 root          11 vmunix -> /pub/vmunix

lenin% ls -l /pub
total 1448
drwxr-xr-x  2 bin          1024 bin
-rwxr-xr-x  1 root        22283 boot
drwxr-xr-x  2 root         4096 lost+found
drwxr-xr-x  2 bin           512 stand
-rwxr-xr-x  1 root       460800 vmunix
```

```
lenin% ls -l /usr
total 34
lrwxrwxrwx  1 root          16 adm -> /private/usr/adm
drwxr-xr-x  2 bin         2048 bin
lrwxrwxrwx  1 root          18 crash -> /private/usr/crash
drwxr-xr-x  3 bin          512 dict
drwxrwxr-x  2 root          24 doctools
drwxr-xr-x  2 bin         1024 etc
drwxr-xr-x  2 bin         7680 hosts
drwxr-xr-x 22 bin         1536 include
drwxr-xr-x 17 bin         2560 lib
drwxrwxrwx  6 root        1024 local
drwxr-xr-x  2 root        4096 lost+found
drwxrwxrwx  2 root          24 man
drwxrwxr-x  2 root         512 mdec
lrwxrwxrwx  1 root          21 preserve -> /private/usr/preserve
drwxr-xr-x  2 bin          512 pub
drwxr-xr-x  3 bin          512 sccs
lrwxrwxrwx  1 root          18 spool -> /private/usr/spool
lrwxrwxrwx  1 root          16 tmp -> /private/usr/tmp
drwxr-xr-x  2 bin         1536 ucb

lenin% ls -l /usr/lib | grep " ->"
lrwxrwxrwx  1 root          24 aliases -> /private/usr/lib/aliases
lrwxrwxrwx  1 root          28 aliases.dir -> /private/usr/lib/aliases.dir
lrwxrwxrwx  1 root          28 aliases.pag -> /private/usr/lib/aliases.pag
lrwxrwxrwx  1 root          24 crontab -> /private/usr/lib/crontab
lrwxrwxrwx  1 root          28 sendmail.cf -> /private/usr/lib/sendmail.cf

lenin% ls -l /private/usr
total 7
drwxr-xr-x  2 bin          512 adm
drwxr-xr-x  2 bin           24 crash
drwxrwxrwx  2 root         512 lib
drwxr-xr-x  2 bin           24 preserve
drwxr-xr-x 12 bin          512 spool
drwxrwxrwx  2 bin          512 tmp

lenin% ls -l /private/usr/lib
total 16
-rw-rw-rw-  1 root         1103 aliases
-rw-rw-rw-  1 root            0 aliases.dir
-rw-rw-rw-  1 root         1024 aliases.pag
-rw-r--r--  1 root          205 crontab
-r--r--r--  1 root        11828 sendmail.cf
```

2.6. Debugging the Network File System

Before trying to debug the NFS read the section on how the NFS works and also the man pages for `mount`(8), `nfsd`(8), `biod`(8), `showmount`(8), `rpcinfo`(8), `mountd`(8), `inetd`(8), `fstab`(5), `mtab`(5) and `exports`(5). You don't have to understand them fully, but you should be familiar with the names and functions of the various daemons and database files.

When tracking down an NFS problem keep in mind that, like all network services, there are three main points of failure: the server, the client, or the network itself. The debugging strategy outlined below tries to isolate each individual component to find the one that isn't working.

For example, let's look at a sample mount request as made from an NFS client machine:

```
% mount krypton:/usr/src /krypton.src
```

and try to understand how it works and how it can fail. The example asks the server machine ''krypton'' to return a file handle (`fhandle`) for the directory `/usr/src`. This `fhandle` is then passed to the kernel in the `nfsmount`(2) system call. The kernel looks up the directory `/krypton.src` and, if everything is okay, it ties the `fhandle` to the directory in a mount record. From now on all file system requests to that directory and below will go through the `fhandle` to the server ''krypton''.

That's how it should work, but if you are reading this it probably didn't. So, how did it fail. First, some general pointers and then we will list the possible errors, and what might have caused them.

2.6.1. General Hints

When there are network or server problems, programs that access hard mounted remote files will fail differently than those which access soft mounted remote files. Hard mounted remote file systems cause programs to retry until the server responds again. Soft mounted remote file systems return an error after trying for a while. `mount` is like any other program, if the server for a remote file system fails to respond it will retry the mount request until it succeeds. A soft mount will try once in the foreground then background itself and keep trying.

Once a hard mount succeeds, programs that access hard mounted files will hang as long as the server fails to respond. In this case, NFS should print a `server not responding` message on the console. On a soft mounted file system programs will get `Connection timed out` (ETIMEDOUT) when they access a file whose server is dead. Unfortunately, many programs in UNIX do not check return conditions on file system operations so you may not see this error message when accessing soft mounted files. Nevertheless, an NFS error message should be printed on the console in this case also.

If a client is having NFS trouble, check first to make sure the server is up and running. From a client you can type

```
% /usr/etc/rpcinfo -p server_name
```

to see if the server is up at all. It should print out a list of program, version, protocol, and port numbers that resembles:

```
[program, version, protocol, port]:

[100005, 1, 17, 1072]
[100001, 2, 17, 1081]
[100001, 1, 17, 1081]
[100002, 1, 17, 1078]
[100008, 1, 17, 1075]
[100007, 1, 17, 1035]
[100007, 1,  6, 1027]
[100004, 1,  6, 1026]
[100004, 1, 17, 1024]
```

If that works you can also use `rpcinfo` to check if the `mountd` server is running:

```
% /usr/etc/rpcinfo -u server_name 100005 1
```

This should come back with the response:

```
proc 100005 vers 1 ready and waiting
```

If these fail you should go login to the server's console and see if it is okay.

If the server is alive but your machine can't reach it you should check the Ethernet connections between your machine and the server (see *Ethernet Troubleshooting* in the chapter on *Communications* in this manual).

If the server is okay and the network is okay use `ps` to check your client daemons. You should have a `portmap`, `ypbind`, and several `biod` daemons running. For example:

```
% ps ax
```

should give back lines similar to these:

```
32 ?  I     1:07 /etc/portmap
38 ?  I     0:42 /etc/ypbind
61 ?  S     0:45  (biod)
62 ?  S     0:36  (biod)
63 ?  S     0:30  (biod)
64 ?  S     0:27  (biod)
```

The four sections below deal with the most common types of failure. The first tells what to do if your remote mount fails, the next three talk about servers not responding once you have file systems mounted.

2.6.2. Remote Mount Failed

This section deals with problems related to mounting. If `mount` fails for any reason check the sections below for specific details about what to do. They are arranged according to where they occur in the mounting sequence and are labeled with the error message you are likely to see.

`mount` can get its parameters either from the command line or from the file `/etc/fstab` (see `mount` (8)). The example below assumes command line arguments, but the same debugging techniques work if `/etc/fstab` is used in the `mount -a` command.

Keep in mind the interaction of the various players in the mount request. If you understand this, the problem descriptions below will make a lot more sense.

Let's look again at the example `mount` request given above,

```
% mount krypton:/usr/src /krypton.src
```

and see what steps `mount` goes through to mount a remote file system.

1) `mount` opens `/etc/mtab` and checks that this mount has not already been done.
2) `mount` parses the first argument into host ''krypton'' and remote directory ''/usr/src''.
3) `mount` calls the yellow pages binder daemon `ypbind` to determine which server machine to find the yellow pages server on. It then calls the `ypserv` daemon on that machine to get the internet protocol (IP) address of krypton.
4) `mount` calls krypton's portmapper to get the port number of `mountd`.
5) `mount` calls krypton's `mountd` and passes it ''/usr/src''.
6) Krypton's `mountd` reads `/etc/exports` and looks for the exported file system that contains ''/usr/src''.
7) Krypton's `mountd` calls the yellow pages server `ypserv` to expand the host names and netgroups in the export list for ''/usr/src''.
8) Krypton's `mountd` does a `getfh` (2) system call on ''/usr/src'' to get the `fhandle`.
9) Krypton's `mountd` returns the `fhandle`.
10) `mount` does an `nfsmount` (2) system call with the `fhandle` and ''/krypton.src''.
11) `nfsmount` checks if the caller is superuser and if ''/krypton.src'' is a directory.
12) `nfsmount` does a `statfs` (2) call to krypton's NFS server (`nfsd`).
13) `mount` opens `/etc/mtab` and adds an entry to the end.

Any one of these steps can fail, some of them in more than one way. The sections below give detailed descriptions of the failures associated with specific error messages.

`/etc/mtab: No such file or directory`

The mounted file system table is kept in the file `/etc/mtab` (5). This file must exist before mount can succeed.

`/etc/mtab: No such file or directory`

The mounted file system table is kept in the file `/etc/mtab` (5). This file must exist before mount can succeed.

`mount: ... already mounted`

The file system that you are trying to mount is already mounted or there is a bogus entry for it in `/etc/mtab`.

`mount: ... Block device required`

You probably left off the ‘‘krypton:’’ part off

```
# mount krypton:/usr/src /krypton.src
```

The `mount` command assumes you are doing a local mount unless it sees a colon in the file system name or the file system type is ‘‘nfs’’ in `/etc/fstab`.

`mount: ... not found in /etc/fstab`

If `mount` is called with only a directory or file system name but not both it looks in `/etc/fstab` for an entry whose file system or directory field matched the argument. For example,

```
# mount /krypton.src
```

will search `/etc/fstab` for a line that has a directory name field of ‘‘/krypton.src’’. If it finds an entry, such as:

```
krypton:/usr/src /krypton.src nfs rw,hard 0 0
```

it will do the mount as if you had typed

```
# mount krypton:/usr/src /krypton.src
```

If you see this message it means the argument you gave `mount` was not in any of the entries in `/etc/fstab`.

`/etc/fstab: No such file or directory`

`mount` tried to look up the name in `/etc/fstab` but there was no `/etc/fstab`.

`... not in hosts database`

This means the yellow pages could not find the hostname you gave it in the `/etc/hosts` database or that the yellow pages daemon (`ypbind`) is dead on your machine. First check the spelling and the placement of the colon in your mount call. If it looks okay make sure that `ypbind` is running by typing:

```
# ps ax
```

Try `rlogin` or `rcp` to some other machine. If this also fails your `ypbind` is probably dead or hung. If you only get this message for one host name it means that the `/etc/hosts` entry on the yellow pages server needs to be checked. See the section on debugging the yellow pages below in this chapter.

.ft

mount: directory path must begin with '/'

The second argument to mount is the path of the directory to be covered. This must be an absolute path starting at ‘‘**/**’’.

mount: ... server not responding: RPC_PMAP_FAILURE - RPC_TIMED_OUT

Either the server you are trying to mount from is down, or its portmapper is dead or hung. Try logging in to that machine. If you can log in, try running:

rpcinfo -p *hostname*

You should get a list of registered program numbers. If you don’t, you should restart the portmapper. Note that restarting the portmapper requires that you then kill and restart both `inetd` and `ypbind`. To do this, become root and do a

ps ax

to find the process ids of `portmap`, `ypbind`, and `inetd`. Next, do a

kill -9 *portmap_pid ypbind_pid inetd_pid*

to kill the daemons. Then type

/etc/portmap
/etc/inetd
/etc/ypbind

to start new ones.

If you don’t want to mess around with this, you can just reboot the server.

If you can’t `rlogin` to the server but the server is up, you should check your Ethernet connection by trying `rlogin` to some other machine. You should also check the server’s Ethernet connection.

mount: ... server not responding: RPC_PROG_NOT_REGISTERED

This means that mount got through to the portmapper but the NFS mount daemon (`rpc.mountd`) was not registered. Go to the server and be sure that `/usr/etc/rpc.mountd` exists and that there is an entry in `/etc/servers` exactly like:

```
rpc     udp     /usr/etc/rpc.mountd 100005 1
```

Finally, use `ps` to be sure that the internet daemon (`inetd`) is running. If you had to change `/etc/servers` you will have to kill `inetd` and restart it. Look in `/etc/rc.local` to see how it is started at boot time and do that by hand.

`mount: ...: No such file or directory`

Either the remote directory or the local directory doesn't exist. Check spelling. Try to `ls` both directories.

`mount: not in export list for ...`

Your machine name is not in the export list for the file system you want to mount from the server. You can get a list of the server's exported file systems by running

showmount -e *hostname*

If the file system you want is not in the list, or your machine name or netgroup name is not in the user list for the file system, login to the server and check the `/etc/exports` file for the correct file system entry. A file system name that appears in the `/etc/exports` file but not in the output from `showmount`, indicates a failure in `mountd`. Either it could not parse that line in the file, or it could not find the file system, or the filesystem name was not a local mounted file system. See `exports` (5) for more information. If `exports` seems okay check the server's `ypbind` daemon, it may be dead or hung.

`mount: ...: Permission denied`

This message is sort of a generic indication that some authentication failed on the server. It could simply be that you are not in the export list (see above) or the server couldn't figure out who you are (`ypbind` dead), or it could be that the server doesn't believe you are who you say you are. Check the server's `/etc/exports` and `ypbind`. In this case you can just change your hostname (`hostname` (1)) and retry the `mount`.

`mount: ...: Not a directory`

Either the remote path or the local path is not a directory. Check spelling and try to `ls` both directories.

`mount: ...: Not owner`

You have to do the mount as root on your machine because it affects the file system for the whole machine, not just you.

2.6.3. Programs Hung

If programs hang doing file related work, your NFS server may be dead. You may see the message `NFS server not responding, still trying` on your console. This is probably a problem either with one of your NFS servers or with the Ethernet. Programs can also hang if an `nd` server dies (see the section on `nd` above) or if a `yp` server dies (see `yp` below).

If your machine hangs completely, check the server(s) from which you have mounted. If one of them (or more) is down don't worry, when the server comes back up your programs will continue automatically and they won't even know the server died. No files will be destroyed.

If a soft mounted server dies, other work should not be affected. Programs that time-out trying to access soft mounted remote files will fail with `errno ETIMEDOUT`, but you should still be able to get work done on your other file systems.

If all of the servers are running go ask someone else using the server or servers that you are using if they are having trouble. If more than one machine is having problems getting service, then it is probably a problem with the server's NFS daemon (`nfsd`(4)). Log in to the server and do a `ps` to see if `nfsd` is running and accumulating cpu time. If not you may be able to kill and then restart `nfsd`. If this doesn't work you will have to reboot the server.

If other people seem to be okay you should check your Ethernet connection and the connection of the server.

2.6.4. Hangs Part Way Through Boot

If your machine comes part way up after a boot, but hangs where it would normally be doing remote mounts, probably one or more servers is down or your network connection is bad. See *Program Hung* and *Remote Mount Failed* above.

2.6.5. Everything Works But Slowly

If access to remote files seems unusually slow, type:

```
# ps aux
```

on the server to be sure it is not being clobbered by a runaway daemon, bad `tty` line, etc. If the server seems okay and other people are getting good response, make sure your block I/O daemons are running; type `ps ax` and look for `biod`. If they are not running or are hung you can kill them off by typing

```
# ps ax | grep biod
```

to find the process ids, and

```
# kill -9 pid1 pid2 pid3 pid4
```

Restart them with

```
# /etc/biod 4
```

To determine whether they are hung, do a `ps` as above, then copy a large remote file, then do another `ps`. If the `biod`s don't accumulate cpu time they are probably hung.

If `biod` is okay check your Ethernet connection. The command `netstat -i` will tell you if you are dropping packets. Also, `nfsstat -c` and `nfsstat -s` can be used to tell if the client or server is doing lots of retransmitting. A retransmission rate of 5% is considered high. Excessive retransmission usually indicates a bad Ethernet board, a bad Ethernet tap, a mismatch between board and tap, or a mismatch between your Ethernet board and the server's board. (See *Ethernet Troubleshooting* in the *Communications* chapter of this manual).

2.7. Architectural Incompatibilities To Earlier UNIX Versions

A few things work differently, or don't work at all, on remote NFS file systems. Here we discuss the incompatibilities and suggest how to work around them.

2.7.1. No Superuser Access Over The Network

Under the NFS a server exports file systems it owns so clients may remote mount them. When a client becomes superuser, it is denied permission on remote mounted file systems. Let's look at the following example:

```
% cd
% touch test1 test2
% chmod 777 test1
% chmod 700 test2
% ls -l test*
-rwxrwxrwx  1 jsbach       0 Mar 24 16:12 test1
-rwx------  1 jsbach       0 Mar 24 16:12 test2
```

Now, retry it again as root.

```
% su
Password:
# touch test1
# touch test2
touch: test2: Permission denied
# ls -l test*
-rwxrwxrwx  1 jsbach       0 Mar 21 16:16 test1
-rwx------  1 jsbach       0 Mar 21 16:12 test2
```

The problem usually shows up during the execution of a set-uid root program. Programs that run as root cannot access files or directories unless the permission for ''other'' allows it.

There's another wrinkle to the problem. You cannot change ownership of remote mounted files. Since users cannot do a `chown` command and root is treated as a normal user on remote access, no one but root on the server can change the ownership of remote files. For example, as yourself, you attempt to `chown` a new program, `a.out`, which must be set-uid root. It will not work, as demonstrated here:

```
% chmod 4777 a.out
% su
Password:
# chown root a.out
a.out: Not owner
```

To change the ownership, you must login to the server as root, then make the change. Or, you can move the file to a file system owned by your machine (for example `/usr/tmp` is always owned by the local machine) and make the change there.

In a very friendly network environment, you may choose to allow root access over the network. (Sun does not recommend over-the-net root access!) You may `adb`(1) the server's kernel in the following way. (NOTE: You never change the client's kernel in this procedure, only the server's.)

- On the NFS server change the value of the kernel variable ‘‘nobody’’. On a running system you type:

```
# adb -w /vmunix /dev/kmem
```

adb responds:

```
not core file = /dev/kmem
```

This is normal. Then you type:

```
nobody/D
```

and adb responds with:

```
_nobody:
_nobody:        -2
```

If it does not, stop and call Sun Tech Support for further help. If it does, then you enter:

```
nobody/W0
```

and adb responds with:

```
_nobody:        -2              =       0
```

(Note: the last character in each of the two previous lines is zero.) Then you type <ctrl-D> to exit adb. The kernel currently running will now allow root access to NFS clients.

- If you want to do this on a binary image, for example /vmunix, then you type:

```
# adb -w /vmunix
```

adb makes no response, so you type:

```
nobody?D
```

and adb responds:

```
_nobody:
_nobody:        -2
```

Again, if adb does not say this, stop and call Sun Tech Support for further help. You then type:

```
nobody?W0
```

and adb responds:

```
_nobody:        -2              =       0
```

(Note: the last character in each of the two previous lines is zero.) Then you type <ctrl-D> to exit adb. From now on, every time you boot /vmunix, the system will allow root access across the NFS

- If you want to make new kernels that allow root access, you can either go through the procedure above for a binary image each time, or you can go the directory where the kernel object files are and type:

```
# adb -w nfs_server.o
```

  You continue as above for in the binary image example. Then every kernel you make with these object files will allow root access on the NFS.

2.7.2. File Operations Not Supported

File locking is not supported on remote file systems. Therefore, the `flock`(2) call will fail when locking a remote file.

In addition append mode and atomic writes are not guaranteed to work on remote files accessed by more than one client simultaneously.

2.7.3. Cannot Access Remote Devices

In the NFS you cannot access a remote mounted device or any other character or block special file — like named pipes.

2.8. Clock Skew In User Programs

Since the NFS architecture differs is some minor ways from earlier versions of UNIX, you should be mindful of those places where your own programs could run up against these incompatibilities. Be sure to read the section above, *Architectural Incompatibilities*, for a discussion of what will not work over the network.

Because each workstation keeps its own time, the clocks will be out of sync between the NFS server and client. Obviously this might introduce a problem in certain situations. In the 2.0 release, we have fixed all the clock skew problems we have seen. Here are examples of two problems and how they were fixed.

1) Many programs make the (reasonable) assumption that an existing file could not have been created in the future. For example `ls` does it. The command `ls -l` has two basic forms of output, depending upon how old the file is:

```
# date
Jan 22 15:27:01 PST 1985
# touch file2
# ls -l file*
-rw-r--r--  1 root          0 Dec 27  1983 file
-rw-r--r--  1 root          0 Jan 22 15:27 file2
```

   The first form of `ls` prints the year, month, and day of last file modification if the file is more than six months old. The second form prints the month, day, and minute of last file modification if the file is less than six months old.

   `ls` calculates the age of a file by simply subtracting the modification time of the file from the current time. If the results are greater than six months worth of seconds, the file is ‘‘old’’.

Now assume that the time on the server is Jan 22 15:30:31 (three minutes ahead of our local machine's time):

```
# date
Jan 22 15:27:31 PST 1985
# touch file3
# ls -l file*
-rw-r--r--  1 root          0 Dec 27  1983 file
-rw-r--r--  1 root          0 Jan 22 15:26 file2
-rw-r--r--  1 root          0 Jan 22  1985 file3
```

The problem is that the difference of the two times is huge:

(now) - (modification_time) =
(now) - (now + 180 seconds) =
-180 seconds = huge unsigned number,
which is greater than six months.

Thus, `ls` believes the new file was created long ago in the past. Sun modified `ls` to deal with files which are created a short time in the future.

2) `ranlib`(1) was also modified to deal with clock skew. `ranlib` timestamps a library when it is produced, and `ld` compares that timestamp with the last modified date of the library. If the last modified date occurred after the timestamp, then `ld` instructs the user to run `ranlib` again, and reload the library.

   If the library lives on a server whose clock is ahead of the client that runs `ranlib`, `ld` will always complain. Sun fixed `ranlib` to set the timestamp to the maximum value of the current time and the library's modify time.

In general remember, if your application depends upon local time and/or the file system timestamps, then it will have to deal with clock skew problems if it uses remote files.

3. The Sun Yellow Pages Service

We begin by introducing yp related terms. Following that we list the new yp manual pages; that section also includes pointers to yp documentation beyond this chapter. Next, you will find explanations of the installation and administration of yp. Then a section on how to debug yp when problems occur. And finally, we discuss file access policies and special security issues raised by the yp environment.

3.1. What Is The Yellow Pages Service?

The yellow pages is Sun's distributed network lookup service:

- yp is a distributed system, the database is fully replicated at several sites, each of which runs a server process for the database. These sites are known as yp servers. The multiple servers propagate updated databases among themselves to ensure database consistency. At steady state, it doesn't matter which server process answers a client request, the answer will be the same all over. This allows multiple servers per network, giving the yp service a high degree of availability and reliability.
- yp is a lookup service. It maintains a set of databases which may be queried. A client may ask for the value associated with a particular key within a database, and may enumerate every key-value pair within a database.
- yp is a network service. It uses a standard set of access procedures to hide the details of where and how data is stored.

3.1.1. The yp Map

The yp system serves information stored in yp maps. Each map contains a set of keys and associated values. For example, in a map called hosts, all the host names within a network are the keys, and the internet addresses of these host names are the values. Each yp map has a mapname used by programs to access it. Programs must know the format of the data in the map. Many of the current maps are derived from ASCII files traditionally found in /etc: hosts, group, passwd, and a few others. The format of the data within the yp map is identical (in most cases) to the format within the ASCII file. Maps are implemented by dbm(3) files located in the subdirectories of the directory /etc/yp/*your_domain* on yp server machines.

3.1.2. The yp Domain

A yp domain is a named set of yp maps. You can determine and set your yp domain with the domainname(1) command. Note that yp domains are different from both Internet domains and sendmail domains. A yp domain is simply a directory in /etc/yp containing a set of maps. A yp server holds all of the maps of a yp domain in a subdirectory of /etc/yp. The name of the subdirectory is the name of the domain. For example, maps for the literature domain would be in /etc/yp/literature. Every yp server serves at least one domain: ''yp_private'', which yp uses to find out about other useful domains particular to each site — see ypfiles(8).

A domain name is required for retrieving data from a `yp` database. Each machine on the network belongs to a default domain set at boot time in `/etc/rc.local` with the `domainname`(1) command. A domain name must be set on all machines, both servers and clients. Further, a single name should be used on all machines on a network.

3.1.3. Masters And Slaves

In the yellow pages environment only a few machines have a set of `yp` databases. The `yp` service makes the database set available over the network. A `yp` client machine runs `yp` processes and requests data from databases on other machines. Two kinds of machines have databases: a `yp` slave server and a `yp` master server. The master server updates the databases of the slave servers. Make changes to databases only on the `yp` master server. The changes will propagate from the master server to the `yp` slave servers. If `yp` databases are created or changed on slave server machines instead of master server machines, the `yp`'s update algorithm will get broken. Always do all the database creation and modification on the master server machine.

In theory, a server may be a master with regard to one map, and a slave with regard to another. Random assignment of maps to master or slave machines could introduce a great deal of confusion. We strongly urge you to make a single server the master for all the maps created by `ypinit`(8) within a single domain. This document assumes that one server is the master for all maps in the database.

3.2. Yellow Pages Overview

The yellow pages can serve up any number of databases. Typically these include some files that used to be found in `/etc`. For example, before Sun release 2.0 programs would read the `/etc/hosts` file to find an Internet address. When a new machine was added to the network, a new entry had to be added to every machine's `/etc/hosts` file. With the yellow pages, programs that want to look at the `/etc/hosts` file now do a remote procedure call (`rpc`) to the servers to get the information.

Most of the information describing the structure of the `yp` system and the commands available for that system is contained in manual pages, and is not repeated here. For quick reference, we list the man pages and an abstract of their contents here. For more information, see the *Network Services Guide*.

> `ypserv`(8) — describes the processes which comprise the `yp` system. These are `ypserv`, the `yp` database server daemon, and `ypbind`, the `yp` binder daemon. `ypserv` must run on each `yp` server machine. `ypbind` must run on all machines which use `yp` services, both servers and clients.
>
> `ypfiles`(8) — describes the database structure of the `yp` system. The domain ''yp_private'' and those maps used by the `yp` system itself are described.

`ypinit`(8) — many maps must be constructed from files located in `/etc`, such as `/etc/hosts`, `/etc/passwd` and others. The database initialization tool `ypinit`(8) does all such construction automatically. In addition, it constructs initial versions of maps required by the system but not built from files in `/etc`; examples are the maps ‘‘ypdomains’’, ‘‘ypmaps’’, and ‘‘ypservers’’. Use this tool to set up the master `yp` server and the slave `yp` servers for the first time. Do not (generally) use it as an administrative tool for running systems.

`ypmake`(8) — describes the use of `/etc/yp/Makefile`, which builds several commonly-changed components of the `yp`’s database. These are the maps built from several ASCII files normally found in `/etc`: `passwd`, `hosts`, `group`, `netgroup`, `networks`, `protocols`, and `services`.

`makedbm`(8) — describes a low-level tool for building a `dbm` file which is a valid `yp` map. Databases not built from `/etc/yp/Makefile` may be built or rebuilt using `makedbm`. `makedbm` may also be used to ‘‘disassemble’’ a map so that you can see the key-value pairs which comprise it. The disassembled form may also be modified with standard tools (such as editors, `awk`, `grep`, and `cat`), and is in the form required for input back into `makedbm`.

`yppush`(8) — describes a set of tools to administer a running `yp` system. `yppush` tells a master `yp` server to notice a new version of a map. The new map must be one which has an entry in the map ‘‘ypmaps’’ for the domain, and of which the server is the master, according to ‘‘ypmaps’’. After noticing the new version of the map, the master `yp` server will tell all its peers within a domain that it is the master of that map, and to get a new versions from it (the master).

`yppull`(8) — tells a slave `yp` server to try to get a new copy of some map.

`yppoll`(8) — asks any `ypserv` for the information it holds internally about a single map.

`ypcat`(1) — dumps out the contents of a `yp` map. Use it when you don’t care which server’s version you are seeing. If you need to see a particular server’s map, `rlogin` to that server (or use `rsh`) and use `makedbm`.

`ypwhich`(8) — tells you which `yp` server a node is using at the moment for `yp` services.

3.3. Yellow Pages Installation and Administration

Nine installation and administration topics will be covered:

1.) setting up a master `yp` server
2.) altering a `yp` client’s database to use `yp` services
3.) setting up a slave `yp` server
4.) setting up a `yp` client
5.) modifying individual `yp` maps after `yp` installation
6.) making new `yp` maps after `yp` installation
7.) propagating new `yp` maps
8.) adding a new `yp` server not in the original set of `yp` servers
9.) changing the master server to a different machine

3.3.1. How To Set Up A Master yp Server

To create a new master server, become superuser and change your current directory to /etc/yp. Then run ypinit(8) with the -m switch. The default domain name and the hostname must be set up; the usual case is that they have been set up from /etc/rc.local. You will be asked whether you want the procedure to die at the first non-fatal error (in which case, you can fix the problem and restart ypinit — this is recommended if you haven't done the procedure before), or to continue despite non-fatal errors. In this second case you can try to fix all the problems by hand, or fix some, then restart ypinit. ypinit will prompt you for a list of other hosts which also will be yp servers. (Initially, this will be the set of yp slave servers, but at some future time any of them might become the yp master server.) You need not add any other hosts at this time, but if you know that you will be setting up some more yp servers, add them now. You will save yourself some work later if you do, and there is very little runtime penalty for doing it. (There is *some*, however, so don't name every host in the net.)

Prior to running ypinit, the following files in /etc should be complete and reflect an up-to-date picture of your system: passwd, hosts, group, networks, protocols, and services. In addition, if you know how /etc/netgroup is going to be set up, do that prior to ypinit. If you don't know, ypinit will make an empty ''netgroup'' map.

For security reasons you may restrict access to the master yp machine to a smaller set of users than that defined by the complete /etc/passwd. To do this, copy the complete file to someplace other than /etc/passwd, and edit out undesired users from the remaining /etc/passwd. For a security-conscious system, this smaller file should not include the yp escape entry discussed in the next section.

To start providing yellow pages services, invoke /etc/ypserv. It will be started up automatically from /etc/rc.local on subsequent reboots.

3.3.2. How To Alter A yp Client's File Database So That yp Services Get Used

Once the decision has been made to serve a database with the yp, it is desirable that all nodes in the net access the yp's version of the information, rather than the potentially out-of-date information in their local files. That policy is enforced by running a ypbind process on the client node (including nodes which may be running yp servers), and by abbreviating or eliminating the files which traditionally implemented the database. The files in question are: /etc/passwd, /etc/hosts, /etc/group, /etc/networks, /etc/protocols, /etc/services, /etc/netgroup, /etc/hosts.equiv, and /.rhosts. The treatment of each file is discussed in this section.

- `/etc/networks`, `/etc/protocols`, `/etc/services`, and `/etc/netgroup` need not exist at any `yp` client node. If you are squeamish about removing them, move them to backup names; for instance on a machine named ‘‘ypclient’’:

```
ypclient%  cd /etc
ypclient%  mv networks networks-
ypclient%  <The rest of the renames, similarly>
```

- `/etc/hosts.equiv` is not normally served by the `yp`. However, you can add escape sequences to activate the `yp`. This reduces problems with `rlogin`, or `rsh` which are sometimes caused by different `/etc/hosts.equiv` files on the two machines.

  To let anyone log on to a machine, `/etc/hosts.equiv` may be edited to contain a single line, with only the character ‘+’ (plus) on it. A line with only a ‘‘+’’ means that all further entries will be retrieved from the `yp` rather than the local file.

  Alternatively, more control may be exercised over logins by using lines of the form:

```
+@trusted_group1
+@trusted_group2
-@distrusted_group
```

  Each of the names to the right of the ‘‘@’’ (at) character is assumed to be a group name, defined in the global netgroup database. The netgroup database is served by the `yp`.

  If none of the escape sequences is used, only the entries in `/etc/hosts.equiv` is used, the `yp` is not used.

- `/.rhosts` is not normally served by the `yp`, either. Its format is identical to that of `/etc/hosts.equiv`. However, since this file controls remote root access to the local machine, unrestricted access is not recommended. Make the list of trusted hosts explicit, or use group names for the same purpose.

- `/etc/hosts` must contain entries for the local host’s name, and the local loopback name. These are accessed at boot time when the `yp` service is not yet available. After the system is running, and after the `ypbind` process is up, the `/etc/hosts` file is not accessed at all. An example of the hosts file for `yp` client ‘‘zippy’’ is:

```
127.1           localhost
192.9.1.87      zippy   # John Q. Adams
```

- `/etc/passwd` should contain entries for root and the primary users of the machine, and an escape entry to force the use of the `yp` service. A few additional entries are recommended: `daemon`, to allow file-transfer utilities to work; `sync`, to run `sync` on a screwed-up machine before rebooting; and `operator`, to let a dump operator login. A sample `yp` client’s `/etc/passwd` file looks like:

```
root:wAm0Y4lEnf6:0:10:God:/:/bin/csh
jrandom:uHP1gQ2:1429:10:J Random:/usr2/jrandom:/bin/csh
operator:VyZr6V9:333:20:sys op:/usr2/operator:/bin/csh
daemon:*:1:1::/:
sync::1:1::/:/bin/sync
+::0:0:::
```

The last line informs the library routines to use the `yp` service rather than give up the search. Entries which exist in `/etc/passwd` will mask analogous entries in the `yp` maps. In addition, earlier entries in the file will mask later ones with the same user name, or the same uid. Therefore, please note the order of the entries for daemon and for sync (which have the same uid) and duplicate it in your own file.

- `/etc/group` may be reduced to a single line:

  ```
  +:
  ```

  which will force all translation of group names and group ids to be made via the `yp` service. This is the recommended procedure.

3.3.3. How To Set Up A Slave `yp` Server

The network must be working to set up a slave `yp` server — in particular, you must be able to `rcp` files from the master `yp` server to `yp` slaves.

To create a new slave server, change directory to `/etc/yp`. From there run `ypinit`(8) with the **-s** switch, and name a host already set up as a `yp` server, as the master. Ideally, the named host really is the master server, but it can be any host which has its `yp` database set up. The `ypserv` process need not be running on the master host, but the host has to be reachable. You must be superuser when you run `ypinit`. The default domain name on the machine intended to be the `yp` slave server must be set up, and must be set to the same domain name as the default domain name on the machine named as the master. In addition, an entry for ‘‘daemon’’ must exist in the `/etc/passwd` files of both slave and master, and that entry must precede any other entries which have the same uid. (`setup` creates `/etc/passwd`; make sure your password file has not been altered to change the order. Note the example shown in the section above.) You won’t be prompted for a list of other servers, but you will have the opportunity to choose whether or not the procedure gives up at the first non-fatal error.

After running `ypinit`, make copies of `/etc/passwd`, `/etc/hosts`, `/etc/group`, `/etc/networks`, `/etc/protocols`, `/etc/netgroup`, and `/etc/services`. For instance on a machine named ‘‘ypslave’’:

```
ypslave% cp /etc/passwd /etc/passwd-
```

Edit the original files in accordance with the section above on altering the client’s database, so as to insure that processes on the slave `yp` server will actually make use of the `yp` services, rather than the local ASCII files. (That is, make sure the `yp` slave server is also a `yp` client.) Make backup copies of the edited files, as well. For instance:

```
ypslave% cp /etc/passwd /etc/passwd+
```

After the yp database gets set up by ypinit, type /etc/ypserv to begin supplying yp services. On subsequent reboots, it will start automatically from /etc/rc.local

3.3.4. How To Set Up A yp Client

To set up a yp client, edit the local files as described in the section above on altering a yp client's file database. If /etc/ypbind is not running already, start it. With the ASCII databases of /etc abbreviated and /etc/ypbind running, the processes on the machine will be clients of the yp services. At this point, there must be a yp server available; all sorts of stuff will hang up if no yp server is available while ypbind is running. Note the possible alterations to the client's /etc database as discussed above in the section on altering the client. Because some files may not be there, or some may be specially altered, it is not always obvious how the ASCII databases are being used. The escape conventions used within those files to force inclusion and exclusion of data from the yp databases are found in the following man pages: passwd(5), hosts(5), netgroup(5), host.equiv(5), group(5). In particular, notice that changing passwords in /etc/passwd (by editing the file, or by running passwd(1)), will only affect the local client's environment. Change /etc/passwd on the master, as described in the next section, to make a global password change.

3.3.5. How To Modify Existing yp Maps After yp Installation

Databases served by the yp must be changed ON THE MASTER SERVER. The databases expected to change most frequently, like /etc/passwd, may be changed by first editing the ASCII file, and then running make(1) on /etc/yp/Makefile — also see ypmake(8).

Non-standard databases (that is, databases which are specific to the applications of a particular vendor or site, but which are not part of Sun's release), or databases which are expected to change rarely, or databases for which no ASCII form exists (for example, databases not around before the yp), may be modified ''manually''. The general procedure is to use makedbm(8) with the **-u** switch to disassemble them into a form which can be modified using standard tools (such as awk, sed, or vi). Then build a new version again using makedbm(8). This may be done by hand in two ways:

1) The output of makedbm can be redirected to a temporary file which can be modified, then fed back into makedbm, or

2) The output of makedbm can be operated on within a pipeline which feeds into makedbm again directly. This is appropriate if the disassembled map can be updated by modifying it with awk, sed, or a *cat* append, for instance.

Suppose we want to create a non-standard `yp` map, called ‘‘mymap’’. We want it to consist of key-value pairs in which the keys are strings like al, bl, cl, etc. (the ‘‘l’’ is for ‘‘left’’), and the values are ar, br, cr (the ‘‘r’’ is for ‘‘right’’). There are two possible procedures to follow when creating new maps. In one we use an existing ASCII file as input; in the other we use standard input.

For example, suppose there is an existing ASCII file named ‘‘/etc/yp/mymap.asc’’, created with an editor or a shell script on our machine ‘‘ypmaster’’. ‘‘home_domain’’ is the subdirectory where the map is located. We can create the `yp` map for this file by:

```
ypmaster%  cd /etc/yp
ypmaster%  makedbm mymap.asc home_domain/mymap
```

But at this point we notice the map really should have included another 2-tuple: (dl, dr). We can make the modification quite simply. In all situations like this, remember to modify the map by first modifying the ASCII file. Modifications made to the map, not also in the ASCII file, will be lost. Make the modification like this:

```
ypmaster%  cd /etc/yp <if not already there>
ypmaster%  <make editorial change to mymap.asc>
ypmaster% makedbm mymap.asc home_domain/mymap
```

When there is no original ASCII file, we can create the `yp` map from the keyboard by typing input like this (our machine name is ‘‘ypmaster’’, and the default domain is ‘‘home_domain’’):

```
ypmaster% cd /etc/yp
ypmaster% makedbm - home_domain/mymap
al ar
bl br
cl cr
<ctl D>
```

When you need to modify that map, you can use `makedbm` to create a temporary ASCII intermediate file which can be edited using standard tools. For instance:

```
ypmaster%  cd /etc/yp
ypmaster%  makedbm -u home_domain/mymap > mymap.temp
```

At this point ‘‘mymap.temp’’ can be edited to contain the correct information. A new version of the database is created by the commands

```
ypmaster%  makedbm mymap.temp home_domain/mymap
ypmaster%  rm mymap.temp
```

The preceding paragraphs explained how to use some tools, but in reality almost everything you actually have to do can be done by `ypinit`(8) and `/etc/yp/Makefile`, unless you add non-standard maps to the database, or change the set of `yp` servers after the system is already up and running.

Whether you use the Makefile in `/etc/yp` or some other procedure — Makefile is one of many possible — the goal is the same: a new pair of well-formed `dbm` files must end up in the domain directory on the master `yp` server.

3.3.6. How To Make New yp Maps After yp Installation

When you make a new yellow pages map, you must update yp's database of maps (ypmaps) to include the new one. Suppose your new map is called ''ishmael'', and ''home_domain'' is the sub-directory where the maps are kept. Then, on the yp master server, you would type:

```
ypmaster%  cd /etc/yp
ypmaster%  (makedbm -u home_domain/ypmaps;\
echo ishmael) | makedbm - tmpmap
```

Note that we display the second command above on two lines. You may type it in as one long command (even if the line wraps on your screen), or you may escape the return and newline with a backslash, as shown here. However, you cannot simply type in half the command, hit return, and type the second half.

makedbm makes two files tmpmap.dir and tmpmap.pag. Move those two as shown below:

```
ypmaster%  mv tmpmap.dir home_domain/ypmaps.dir
ypmaster%  mv tmpmap.pag home_domain/ypmaps.pag
ypmaster%  yppush ypmaps
```

A temporary file must be used here so the second invocation of makedbm doesn't clobber the ypmaps before the first invocation gets to disassemble it.

Once you have made changes to an existing database or created a new one, you should edit /etc/yp/Makefile to permanently incorporate the procedure.

3.3.7. How To Propagate A New yp Database

A short time after a database has been newly created or altered, the yp system will find out about it and will propagate the changes. If you want to hurry things up, you can run yppush(8) to inform the master server that a new version of the map has been placed in the domain. You must update the map named ''ypmaps'' to include a new map named ''foo'', and then run yppush for ''ypmaps'' before trying to run yppush for ''foo''.

The master will try to tell its slaves about new versions of maps, but if a slave is overloaded or unreachable, that slave may not find out about the change. Further, the master may be overloaded when the slave tries to get the new version from the master. This means that yppush may not work. You can use yppoll(8) to find out which version is where. If you believe that the correct version is present at the master, and that the master knows about it, but that it is not yet at a slave in which you have special interest, you may run yppull(8) to ''kick'' that slave into trying to get the map from the master. Run yppush first if you plan on running yppull at all, so that the master can correctly report his current version to his peers. Here again, if the master is overloaded, the peer may not be able to get the new version from the master; you can tell by using yppoll.

It is very important to note there is a delay between the time you run `yppush` or `yppull` and the time the database transfer completes. Generally it takes a minute or two. However, if the map hasn't been moved within 5 minutes, it's probably not going to make it. Make sure that both the master and the slave are up, and try again.

3.3.8. How To Add A New `yp` Server Not In The Original Set Of `yp` Servers

To add a new `yp` slave server start by modifying some maps on the master `yp` server. If the new server is a host which has not been a `yp` server before, the host's name must be added to the map ''ypservers'' in the domain ''yp_private'' and in the default domain. The sequence for adding a server named ''ypslave'' to domain ''home_domain'' is:

```
ypmaster% cd /etc/yp
ypmaster% (makedbm -u yp_private/ypservers;\
echo ypslave ypslave)|makedbm - tmpmap
ypmaster% mv tmpmap.dir yp_private/ypservers.dir
ypmaster% mv tmpmap.pag yp_private/ypservers.pag
ypmaster% (makedbm -u home_domain/ypservers;\
echo ypslave ypslave)|makedbm - tmpmap
ypmaster% mv tmpmap.dir home_domain/ypservers.dir
ypmaster% mv tmpmap.pag home_domain/ypservers.pag
```

Note that we display some commands above on two lines. You may type these as one long command (even if the line wraps on your screen), or you may escape the return and newline with a backslash, as shown here. However, you cannot simply type in half the command, hit return, and type the second half.

The host's address should be in ''hosts.byname''. If that's not true, edit `/etc/hosts` and run `make`. In this case the commands should be:

```
ypmaster%  <edit /etc/hosts here>
ypmaster% cd /etc/yp
ypmaster% make NOPUSH=1 hosts
ypmaster% cp home_domain/hosts.byname.* yp_private
```

After updating the databases, they should be `yppush`ed and `yppull`ed around as described in the section above on modifying an individual database. The new slave `yp` server's databases should be set up by copying the databases from `yp` master server ''ypmaster''. Remote login to the new `yp` slave, and use `ypinit` (8) in the following way:

```
ypslave% cd /etc/yp
ypslave% ypinit -s ypmaster
```

Then complete the steps described above in the section *How To Set Up A Slave yp Server*.

3.3.9. How To Change The Master Server

When changing master servers, you will follow one of two procedures: the first when the master is out of service; the second when the master is running and will convert to running as a slave.

Master Out Of Service — When the old master is down, begin on an operating slave `yp` server. The slave must have a complete set of ASCII files (`/etc/passwd`, and the others) so that updates can be done there when it becomes the new master. Ideally, these files have been restored from a recent backup of the old master, so that recent changes are reflected.

To change the slave to a master, the map ‘‘ypmaps’’ must be changed in both domain ‘‘yp_private’’ and the default domain. There is no intermediate ASCII form for this map. Probably the easiest way to modify ‘‘ypmaps’’ so that each map mentioned in it points at the new master is to use `makedbm`(8) to create a temporary ASCII file, edit that, and then push the temp file through `makedbm` again to update the maps. At this point, if it is running, kill `ypserv` on the slave. (Re)start `ypserv`. That server now believes that it is the master server. Now we have to get the other `yp` servers to think so too. Run `yppush`(8) for both the default domain and for ‘‘yp_private’’ for the map ‘‘ypmaps’’, and also for ‘‘ypservers’’ and/or ‘‘hosts.byname’’ and ‘‘hosts.byaddr’’ if they have been modified. You can also run `yppull`(8) to kick any slave servers which might not be immediately reachable.

If the old master does come back up and is still supposed to be a `yp` server, the easiest way to make it consistent with the new world is to run **`ypinit`**(8) with the **`-s`**, configuring the old master as a slave and drawing the current database from the new master. This is a sort of blunt-instrument approach, but is easier than the other available alternatives.

Master Converts To Slave — It is easier to change from one master server to another when the old master will continue to run. Make whatever changes are required to ‘‘ypmaps’’ and ‘‘ypservers’’ on the old master, and run `yppush`(8) on the old master. This will let the old master know about the new versions, but once it finds out about the change of its own status from the new copy of ‘‘ypmaps’’, it will not contact its peers to tell them to get new versions of the maps from it. Then run **`ypinit`**(8) with a **`-s`** switch at the new master to get the latest version of everything from the old master to the new master. Also make sure the ‘‘sources’’ like `/etc/passwd`, are copied to the new master. Kill and restart `ypserv` at the new master. It will then know it is the master. You should then run `yppush`(8) at the new master to get the rest of slaves to know about the new master.

3.4. Debugging A Yellow Pages Client

We divide this debugging section into two parts — first those problems seen on a `yp` client, and then those problems seen on a `yp` server.

Before trying to debug a yellow pages client, read the earlier section in this chapter on how the `yp` works.

3.4.1. On Client: Commands Hang

The most common problem at a `yp` client node is for a command to hang and generate console messages which say:

```
yp: server not responding for domain <wigwam>. Still trying
```

Sometimes many commands will begin to hang, even though the system as a whole seems okay and you can run new commands.

The message above indicates that `ypbind` on the local machine is unable to communicate with `ypserv` in the domain ‘‘wigwam’’. This often happens when machines which run `ypserv` have crashed. It may also occur if the net or the `yp` server machine is so overloaded that `ypserv` can’t get a response back to your `ypbind` within the timeout period. Under these circumstances, all the other `yp` client nodes on your net will show the same or similar problems. The condition is temporary in most cases, and the messages will usually go away when the `yp` server machine reboots and `ypserv` gets back in business; or when the load on the `yp` server nodes and/or the Ethernet decreases.

However, in the circumstances described below, the situation will never get better.

- If the `yp` client has not set, or incorrectly set, `domainname` on the machine. Clients must use a domain name that the `yp` servers know. Use `domainname`(1) to see the client domain name. Compare that with the domain name set on the `yp` servers. The domain name should be set in `/etc/rc.local`. When `/etc/rc.local` fails to set, or incorrectly sets, `domainname` you will have to reboot your system; reboot it single-user (type `b -s` to the prom monitor). Then edit `/etc/rc.local` to set the name correctly. Type `<ctrl d>` to come up multi-user, and the problem should be gone.
- If your domain name is correct, make sure your local net has at least one `yp` server machine. You can only bind to a `ypserv` process on your local net, not on another accessible net. There must be at least one `yp` server for your machine’s domain running on your local net. Two or more `yp` servers will improve availability and reponse characteristics for `yp` services.

- If your local net has a `yp` server, make sure it is up and running. Check other machines on your local net. If several client machines have problems simultaneously, suspect a server problem. Find a client machine behaving normally, and try the `ypwhich` command. If `ypwhich` never returns an answer, kill it and go to a terminal on the `yp` server machine. Type:

  ```
  % ps ax | grep yp
  ```

  and look for `ypserv` and `ypbind` processes. If the server's `ypbind` daemon is not running, start it up by typing:

  ```
  % /etc/ypbind
  ```

  If there is a `ypserv` process running, do a `ypwhich` on the `yp` server machine. If `ypwhich` returns no answer, `ypserv` has probably hung and should be restarted. Kill the existing `ypserv` process (you must be logged on as root), and start /etc/ypserv:

  ```
  % kill -9 [some pid # from ps]
  % /etc/ypserv
  ```

  If `ps` shows no `ypserv` process running, start one up.

3.4.2. On Client: `yp` Service Unavailable

When other machines on the network appear to be okay, but `yp` service becomes unavailable on your machine, many different symptoms may show up. Among them: some commands appear to operate correctly while others terminate printing an error message about the unavailability of `yp`; some commands limp along in a backup-strategy-mode particular to the program involved; and some commands or daemons crash with obscure messages or no message at all. For example, things like the following may show up:

```
my_machine% ypcat myfile
ypcat: can't bind to yp server for domain <wigwam>.
        Reason: can't communicate with ypbind.

my_machine% /etc/yp/yppoll myfile
Sorry, I can't make use of the yellow pages. I give up.
```

When symptoms like those above occur, try

```
my_machine% ls -l
```

on a directory containing files owned by many users, including users not in the local machine's `/etc/passwd` file — for example `/usr2`. If the `ls -l` reports file owners not in the local machine's `/etc/passwd` file as numbers, rather than names, it is one more symptom that `yp` service is not working.

These symptoms usually indicate that your `ypbind` process is not running. You can do a `ps ax` to check for one. If it you do not find it, type:

```
my_machine% /etc/ypbind
```

to start it. `yp` problems should disappear.

3.4.3. On Client: `ypbind` Crashes

If `ypbind` crashes almost immediately each time it is started, you should look for a problem in some other part of the system. Check for the presence of the `portmap` daemon by typing:

```
my_machine%  ps ax | grep portmap
```

If you don't find it running, reboot.

If `portmap` itself will not stay up or behaves strangely, look for more fundamental problems. Check the network software in the ways suggested in the section on *Ethernet Debugging* in the *Communications* chapter of this manual.

You may be able to talk to the `portmap` on your machine from a machine operating normally. From such a machine, type:

```
flipper% rpcinfo -p your_machine_name
```

If your `portmap` is okay, the output should look like:

```
[program, version, protocol, port]:

[100005, 1, 17, 1046]
[100001, 2, 17, 1055]
[100001, 1, 17, 1055]
[100002, 1, 17, 1052]
[100008, 1, 17, 1049]
[100007, 1, 17, 1027]
[100007, 1, 6, 1026]
```

On your machine the port numbers will be different. The two entries that represent the `ypbind` process are:

```
[100007, 1, 17, port_#]
[100007, 1, 6, port_#]
```

If they are not there, `ypbind` has been unable to register its services. Reboot the machine. If they are there and they change each time you try to restart `/etc/ypbind`, reboot the system, even if the `portmap` is up. If the situation persists after reboot, call for help.

3.4.4. On Client: `ypwhich` Inconsistent

When you use `ypwhich` several times at the same client node, the answer you get back varies — the `yp` server changes. This is normal. The binding of `yp` client to `yp` server will change over time on a busy net, and when the `yp` servers are busy. Whenever possible, the system stabilizes at a point where all clients get acceptable response time from the `yp` servers. As long as your client machine gets `yp` service, it doesn't matter where the service comes from. Often a `yp` server machine gets its own `yp` services from another `yp` server on the net.

3.5. Debugging A Yellow Pages Server

Before trying to debug a yellow pages server, read the earlier section in this chapter on how the `yp` works.

3.5.1. On Server: Different Versions Of A `yp` Map

Since `yp` works by propagating maps between servers, you will commonly find different versions of a map at different servers on the network. This version skew is normal, if transient, and abnormal otherwise. How do we define transient? Let's look for a minute at the `yp` update-propagation algorithms.

Each map must be updated at its master `yp` server. Typically, all maps have the same master `yp` server — `ypinit`(8) assumes this standard.

`yppush`(8) should always be run on the master server after you update a database. When you use the Makefile in `/etc/yp`, `yppush` runs automatically. `yppush` tells the `ypserv` process at the master about the new copy of the map. `ypserv` then reinitializes its information about the map and contacts its non-master peer `ypserv` processes (in other words, the `yp` slave servers) with news of the update. After this contact, the new map is transferred from master to slave.

During the process there will, of course, be a temporary version skew. Sometimes this skew will drag out for a long time. For example, a heavily loaded slave server, or a slave on a busy net, may not get the update message; or, the `ypserv` process may be swapped out with many previous requests in its queue; or, the slave server's `ypserv` process may be down, or the slave's machine may be down. In these cases, the message sent from the master to the slave may time out or get dumped by the system. The master doesn't keep track of who got the message and who didn't. So if a slave misses the first try, it misses it altogether.

How does the slave get the updated map in the circumstances described above? Each slave server continuously scans its list of known maps, and tries to find some other `yp` server with a more recent version. If it finds one, it gets that more recent version from its peer. It always first attempts to communicate with the map's master, but if the master is unavailable it chooses another peer at random. The combination of the master server ''pushing'' and the slave servers ''scanning for'' updated files will eventually bring map versions into sync. A `yp` server traverses its lists of all known maps in somewhat less than half-an-hour. In addition, a user may run `yppull`(8) to tell a slave to seek a new version of a map. It is similar to `yppush`, since it hurries the system up, but it does not guarantee success - it's a ''hint'' to the system.

A master server scans its list of known maps in a similar manner. But instead of communicating with its peers to find a more recent version, it rechecks the disk files which implement the maps, and tries to find version changes by itself. It also detects when new maps come into existence. In both cases, when it finds an updated or new map, it contacts its slaves exactly as if `yppush` had been run.

Therefore, if you detect a map version skew between two `yp` servers (using `yppoll`(8), for example) run `yppull`(8), and check back on in about ten minutes. Run it a second time if you don't see results. If you try a few times and nothing happens you should suspect something is broken.

Propagation failures can be caused by errors in the databases `yp` uses to run itself. Make sure that all `yp` servers in question are mentioned in the map ‘‘ypservers’’, both within the default domain and the domain ‘‘yp_private’’. Also make sure there are entries in the map ‘‘hosts.byname’’ within both domains.

If none of these solves the problem, you can work around it by using `rcp` or `tftp` to copy those `dbm` files that implement the map, from the master to the out-of-sync slave. Make sure that the `dbm` files end up with ‘‘root’’ as the owner.

3.5.2. On Server: `ypserv` Crashes

When the `ypserv` process crashes almost immediately, and won't stay up even with repeated activations, the debug process is virtually identical to that described above in the section *On Client: ypbind Crashes*. Check for the `portmap` daemon:

```
ypserver% ps ax | grep portmap
```

Reboot the server if you do not find it. If it is there, type:

```
ypserver% /usr/etc/rpcinfo -p ypserver_name
```

and look for output to the screen like:

```
[program, version, protocol, port]:

[100001, 2, 17, 1062]
[100001, 1, 17, 1062]
[100002, 1, 17, 1060]
[100008, 1, 17, 1058]
[100005, 1, 17, 1056]
[100007, 1, 17, 1032]
[100007, 1, 6, 1027]
[100004, 1, 6, 1026]
[100004, 1, 17, 1024]
```

On your machine, the port numbers will be different. The two entries that represent the `ypserv` process are:

```
[100004, 1, 6, port_#]
[100004, 1, 17, port_#]
```

If they are not there, `ypserv` has been unable to register its services. Reboot the machine. If they are there, and they change each time you try to restart `/etc/ypserv`, reboot the machine. If the situation persists after reboot, call for help.

3.6. Yellow Pages Policies

Here are the policies set by the C-library routines when they access the following files on a system running the yellow pages.

`/etc/passwd`
: Always consulted. If there are `+` or `-` entries, the `yp` password map is consulted, otherwise `yp` is unused.

`/etc/group`
: Always consulted. If there are `+` or `-` entries, the `yp` group map is consulted, otherwise `yp` is unused.

`/etc/hosts.equiv`
: (And similarly for `.rhosts`) Always consulted, though neither of these files is in the yellow pages database. (See the section below *How Security Is Changed With The Yellow Pages*, for a fuller explanation of these two files.) If there are `+` or `-` entries, whose arguments are netgroups, the `yp` netgroup map is consulted, otherwise `yp` is unused.

`/etc/services`
: Never consulted. The data that was formerly read from this file now comes from the `yp` services database.

`/etc/protocols`
: Never consulted. The data that was formerly read from this file now comes from the `yp` protocols database.

`/etc/networks`
: Never consulted. The data that was formerly read from this file now comes from the `yp` networks database.

`/etc/netgroup`
: Never consulted. The data that was formerly read from this file now comes from the `yp` netgroup database.

`/etc/hosts`
: Consulted only when booting (by the `ifconfig` command in the `/etc/rc.local` file). After that the `yp` is used instead.

3.7. How Security Is Changed With The Yellow Pages

Read the section above on `yp` accessing policies to better understand `yp` security issues.

3.7.1. Global And Local `yp` Database Files

There are seven files now in the yellow pages database. Six were formerly in `/etc`: `/etc/passwd`, `/etc/group`, `/etc/hosts`, `/etc/networks`, `/etc/services`, and `/etc/protocols`. In addition, there is a new file `/etc/netgroup`. (Note that a site may add database files of its own.) We divide the yellow pages into local and global file types. A local file is first checked for on your own machine, then in the yellow pages. A global file is only checked for in the yellow pages. `/etc/passwd` and `/etc/group` are the local files in the yellow pages database. The other five yellow pages files are global.

For example, a program that calls `/etc/passwd` (a local file) will first look in the password file on your machine; the yellow pages password file will only be consulted if your machine's password file contains ''+'' (plus sign) entries. The `/etc/passwd` file is local so that you can control the entries for your own machine. The only other local file is `/etc/group`. To repeat, local files are consulted first on your own machine, before looking in the yellow pages.

The remaining yellow pages files (`hosts`, `networks`, `services`, `protocols`, and `netgroup`) are global files. The information in these files is network wide data, and in a perfect world would only be accessed from the yellow pages. However, there are two situations when it is necessary to consult a copy of these files on your own machine. The first assists in the changeover to the yellow pages. If someone installs 2.0 software, but a yellow pages server has not yet been started up, then no network data will be available. Second, when booting, each machine needs an entry in `/etc/hosts` for itself. In summary, if yellow pages is running, global files are only checked in the yellow pages; a file on your local machine is not consulted.

3.7.2. Two Other Files `yp` Consults

The files `/etc/hosts.equiv` and `/.rhosts` are not in the yellow pages database. Each machine has its own unique copy. However it is possible to put entries in your `/etc/hosts.equiv` file that refer to the yellow pages. For example a line consisting of

```
+@engineering
```

will include all members of `engineering` as it is defined in the local file `/etc/netgroup` or in the `yp` database. A line consisting only of ''+'' (a plus sign) will include everyone in your `/etc/hosts.equiv` file.

3.7.3. Security Implications

Recall that to be able to log into a machine without having a password, you need to be in both the `/etc/hosts.equiv` file and the `/etc/passwd` file. By having a ''+'' entry in `/etc/hosts.equiv`, you effectively bypass this check, and anyone in your `/etc/passwd` file will be allowed to rlogin to your machine without restriction.

An /etc/passwd file and /etc/group file may also have ‘‘+’’ entries. A line in an /etc/passwd file such as

```
+nb::::Napoleon Bonaparte:/usr2/nb:/bin/csh
```

pulls in an entry for nb from the yellow pages. It gets the uid, gid and password from the yellow pages, and gets the gecos, home directory and default shell from the ‘‘+’’ entry itself. On the other hand, an /etc/passwd entry such as

```
+nb:
```

gets all information from the yellow pages.

Finally, notice that

```
+nb::1189:10:Napoleon Bonaparte:/usr2/nb:/bin/csh
```

is quite different from

```
nb::1189:10:Napoleon Bonaparte:/usr2/nb:/bin/csh
```

In the first of the two examples the password field is obtained from the yellow pages, in the second user nb has no password. Also, if there is no entry for nb in the yellow pages, then the effect of the first example is as if no entry for nb was present at all. That is quite different from the second example.

3.7.4. Special yp Password Change

When you change your password with the passwd(1) command, you will change the entry explicitly given in your own /etc/passwd file. If your password is not given explicitly, but rather is pulled in from the yellow pages with a ‘‘+’’ entry, then the passwd command will print the error message Not in passwd file. To change your passwd in the yellow pages, you must use the new yppasswd(1) command. In order to enable this service, the system administrator must start up the daemon yppasswdd(8C) server on the machine serving as the master for the yellow pages password file.

3.7.5. Manual Pages Covering Security Issues

For more details, see the following man pages: yppasswd(1), hosts.equiv(5), passwd(5), group(5), netgroup(5), yppasswdd(8C).

3.8. What If You Do Not Use The Yellow Pages?

If you choose not to use the yellow pages, the procedure for bypassing the software implementation is quite simple. In the file /etc/rc.local find the lines that look like:

```
if [ -f /etc/ypbind ]; then
        /etc/ypbind; echo -n ’ ypbind’  >/dev/console
fi
```

And comment them out, like:

```
# if [ -f /etc/ypbind ]; then
#         /etc/ypbind; echo -n ’ ypbind’        >/dev/console
# fi
```

4. Adding A New User To A Machine

To add a new user, add an entry to the password file and create a home directory on the new user's machine as described in the steps below. To add a new client machine, see *Adding A Client Machine To An* `nd` *Server, below.*

4.1. Edit The Master `yp` Server's `/etc/passwd` File

Typically, for a new user, add a password file entry to every machine on the local network. You must be superuser to do it, and you should begin on the master `yp` server machine. First edit the master `yp` server's `/etc/passwd` file. Later the password file entry for the user will be copied to the `/etc/passwd` file on the new client's partition; without an entry in it, the person administering the new client machine would not be able to login should the yellow pages fail.

On the master `yp` server add a new line to the password file with the `vipw` command; `vipw` brings the password file into the `vi` editor, and prevents anyone else from editing it until you are done:

```
# /etc/vipw
```

`/etc/passwd` is a readable ASCII file with a one line entry for each valid user on the system. Each entry is separated into fields by colons (:); there are seven fields on each line and some fields may be left blank by placing two colons back to back. Avoid using the characters for single and double quotes (' ' "), and backslashes (\) in the password file. See `passwd`(5) for more about the file format.

Let us suppose that your new user's name is Mr. Chimp and his account is going to be ''bonzo'', you would add a line similar to the one shown below.

```
bonzo::1947:10:Mr. Chimp:/usr/bonzo:/bin/csh
```

Notice that the second field is blank in the example. This field, when filled, contains an encrypted version of the user's password. However, when the field is blank, anyone can login simply by typing the user name — no password is required. You cannot create a password by making an entry in the `/etc/passwd` file. You must use `passwd`(1) while logged in as the user, or as superuser. Since anyone can login when a user has no password, you may provide a password for the new user and let him know it so he can log in and change it to whatever he prefers. When Mr. Chimp logs in for the first time, he can use the `passwd` utility to change his password, or `yppasswd`(1) to change it in the `yp` database.

After Mr. Chimp has a password, the entry for ''bonzo'' in the password file will look something like:

```
bonzo:Sv9wEuIq:1947:10:Mr. Chimp:/usr/bonzo:/bin/csh
```

Fields in the password file have the following meanings:

1) Login name — synonymous with user name.

2) Encrypted password. Tell all new users how to add, or change, their password with the `passwd` command and the `yppasswd` command. Remember, the system administrator can make this field empty when a user has forgotten his or her password, thereby enabling login without a password until such time as a new one is given.

3) User ID. A number unique to this user. A system knows the user by ID number associated with login name, therefore a login name must have the same user ID number on all password files of machines which are networked in a local domain. Failure to keep ID's unique will prevent moving files between directories on different machines because the system will respond as if the directories are owned by two different users. In addition, file ownership may become confused when an NFS server exports a directory to an NFS client whose password file contains users with uid's that match those of different users on the NFS server.

4) Group ID. Can be used to group users together who are working on similar projects. All system staff are in group ''10'' for historical reasons. In this example Mr. Chimp is in the system staff group. Do not put normal users in this group. If you are not sure what group to put a new client in, see `group`(5) and look in the file `/etc/group`.

5) Information about user — usually real name, phone number, etc. An ampersand (&) here is shorthand for the user's login name.

6) The user's home directory — the directory the user logs in to.

7) Initial shell to use on login. If this field is blank the default `/bin/sh` is used. We recommend placing `/bin/csh` here, as in the example above, it gives a Berkeley 4.2 C-Shell as the user's initial shell.

After you have updated the password file and created a password for the new user, be sure to update the yellow database by running `/etc/yp/make` for `/etc/passwd`:

```
# cd /etc/yp
# make /etc/passwd
```

4.2. Make A Home Directory

After adding a new entry to the password file, you should create a home directory for the new user to login to. This will be the same as the directory given in the sixth field of the password file entry. In the `/usr2` directory make a directory for the new user and change ownership to the user's login name and change group to the user's group. For example:

```
# cd /usr2
# mkdir bonzo
# chown bonzo bonzo
# chgrp 10 bonzo
```

Note that if the yellow pages databases for the password file have not yet been updated on the machine's yellow pages server, you will get the following error message when you attempt to do the `chown`:

```
unknown user id:  username
```

In that case, you can use the following set of commands:

```
# cd /usr2
# mkdir bonzo
# chown userid# bonzo
# chgrp 10 bonzo
```

You use Mr. Chimp's user id number (from the password file entry) instead of login name to change the ownership of his home directory.

4.3. The New User's Environment

Finally, you may define the new user's environment on login in several ways. For example you may give her a copy of such files as `.login` and `.cshrc` if she uses ''/bin/csh'', or `.profile` if she uses ''/bin/sh''. See the `csh`(1) and `sh`(1) pages in the *Commands Reference Manual* for discussion of these files, they can automatically set up the terminal and shell environment at each login. You may also want to give a user `.suntools` and `.mailrc` files.

If the new user is a member of any groups at your site, add her to `/etc/group` as necessary — see `group`(5) and `groups`(1). Be sure to make the changes to the `/etc/group` file on the master `yp` server if you run the yellow pages.

.ft