4

Disk Shares

In the previous three chapters, we showed you how to install Samba on a Unix server and set up Windows clients to use a simple disk share. This chapter will show you how Samba can assume more productive roles on your network.

Samba's daemons, *smbd* and *nmbd*, are controlled through a single ASCII file, *smb.conf*, that can contain over 200 unique options. These options define how Samba reacts to the network around it, including everything from simple permissions to encrypted connections and NT domains. The next five chapters are designed to help you get familiar with this file and its options. Some of these options you will use and change frequently; others you may never use—it all depends on how much functionality you want Samba to offer its clients.

This chapter introduces the structure of the Samba configuration file and shows you how to use these options to create and modify disk shares. Subsequent chapters will discuss browsing, how to configure users, security, domains, and printers, and a host of other myriad topics that you can implement with Samba on your network.

Learning the Samba Configuration File

Here is an example of a Samba configuration file. If you have worked with a Windows .INI file, the structure of the *smb.conf* file should look very familiar:

```
[global]
    log level = 1
    max log size = 1000
    socket options = TCP_NODELAY IPTOS_LOWDELAY
    guest ok = no
[homes]
    browseable = no
    map archive = yes
```

```
[printers]
    path = /usr/tmp
    guest ok = yes
    printable = yes
    min print space = 2000
[test]
    browseable = yes
    read only = yes
    guest ok = yes
    path = /export/samba/test
```

Although you may not understand the contents yet, this is a good configuration file to grab if you're in a hurry. (If you're not, we'll create a new one from scratch shortly.) In a nutshell, this configuration file sets up basic debug logging in a default log file not to exceed 1MB, optimizes TCP/IP socket connections between the Samba server and any SMB clients, and allows Samba to create a disk share for each user that has a standard Unix account on the server. In addition, each of the printers registered on the server will be publicly available, as will a single read-only share that maps to the */export/samba/test* directory. The last part of this file is similar to the disk share you used to test Samba in Chapter 2, *Installing Samba on a Unix System*.

Configuration File Structure

Let's take another look at this configuration file, this time from a higher level:

```
[global]
    ...
[homes]
    ...
[printers]
    ...
[test]
    ...
```

The names inside the square brackets delineate unique sections of the *smb.conf* file; each section names the *share* (or service) that the section refers to. For example, the `[test]` and `[homes]` sections are each unique disk shares; they contain options that map to specific directories on the Samba server. The `[printers]` share contains options that map to various printers on the server. All the sections defined in the *smb.conf* file, with the exception of the `[global]` section, will be available as a disk or printer share to clients connecting to the Samba server.

The remaining lines are individual configuration options unique to that share. These options will continue until a new bracketed section is encountered, or until the end of the file is reached. Each configuration option follows a simple format:

```
option = value
```

Options in the *smb.conf* file are set by assigning a value to them. We should warn you up front that some of the option names in Samba are poorly chosen. For example, `read only` is self-explanatory, and is typical of many recent Samba options. `public` is an older option, and is vague; it now has a less-confusing synonym `guest ok` (may be accessed by guests). We describe some of the more common historical names in this chapter in sections that highlight each major task. In addition, Appendix C, *Samba Configuration Option Quick Reference*, contains an alphabetical index of all the configuration options and their meanings.

Whitespaces, quotes, and commas

An important item to remember about configuration options is that all whitespaces in the *`value`* are significant. For example, consider the following option:

```
volume = The Big Bad Hard Drive Number 3543
```

Samba strips away the spaces between the final `e` in `volume` and the first `T` in `The`. These whitespaces are insignificant. The rest of the whitespaces are significant and will be recognized and preserved by Samba when reading in the file. Space is not significant in option names (such as `guest ok`), but we recommend you follow convention and keep spaces between the words of options.

If you feel safer including quotation marks at the beginning and ending of a configuration option's value, you may do so. Samba will ignore these quotation marks when it encounters them. Never use quotation marks around an option itself; Samba will treat this as an error.

Finally, you can use whitespaces to separate a series of values in a list, or you can use commas. These two options are equivalent:

```
netbios aliases = sales, accounting, payroll
netbios aliases = sales accounting payroll
```

In some values, however, you must use one form of separation—spaces in some cases, commas in others.

Capitalization

Capitalization is not important in the Samba configuration file except in locations where it would confuse the underlying operating system. For example, let's assume that you included the following option in a share that pointed to */export/samba/simple*:

```
PATH = /EXPORT/SAMBA/SIMPLE
```

Samba would have no problem with the `path` configuration option appearing entirely in capital letters. However, when it tries to connect to the given directory,

it would be unsuccessful because the Unix filesystem in the underlying operating system *is* case sensitive. Consequently, the path listed would not be found and clients would be unable to connect to the share.

Line continuation

You can continue a line in the Samba configuration file using the backslash, as follows:

```
comment = The first share that has the primary copies \
          of the new Teamworks software product.
```

Because of the backslash, these two lines will be treated as one line by Samba. The second line begins at the first non-whitespace character that Samba encounters; in this case, the `o` in `of`.

Comments

You can insert comments in the *smb.conf* configuration file by preceding a line with either a hash mark (#) or a semicolon (;). Both characters are equivalent. For example, the first three lines in the following example would be considered comments:

```
#  This is the printers section. We have given a minimum print
;  space of 2000 to prevent some errors that we've seen when
;  the spooler runs out of space.

[printers]
    public = yes
    min print space = 2000
```

Samba will ignore all comment lines in its configuration file; there are no limitations to what can be placed on a comment line after the initial hash mark or semicolon. Note that the line continuation character (\) will *not* be honored on a commented line. Like the rest of the line, it is ignored.

Changes at runtime

You can modify the *smb.conf* configuration file and any of its options at any time while the Samba daemons are running. By default, Samba checks the configuration file every 60 seconds for changes. If it finds any, the changes are immediately put into effect. If you don't wish to wait that long, you can force a reload by either sending a SIGHUP signal to the *smbd* and *nmbd* processes, or simply restarting the daemons.

For example, if the *smbd* process was 893, you could force it to reread the configuration file with the following command:

```
# kill -SIGHUP 893
```

Not all changes will be immediately recognized by clients. For example, changes to a share that is currently in use will not be registered until the client disconnects and reconnects to that share. In addition, server-specific parameters such as the workgroup or NetBIOS name of the server will not register immediately either. This keeps active clients from being suddenly disconnected or encountering unexpected access problems while a session is open.

Variables

Samba includes a complete set of variables for determining characteristics of the Samba server and the clients to which it connects. Each of these variables begins with a percent sign, followed by a single uppercase or lowercase letter, and can be used only on the right side of a configuration option (e.g., after the equal sign):

```
[pub]
    path = /home/ftp/pub/%a
```

The `%a` stands for the client machine's architecture (e.g., `WinNT` for Windows NT, `Win95` for Windows 95 or 98, or `WfWg` for Windows for Workgroups). Because of this, Samba will assign a unique path for the `[pub]` share to client machines running Windows NT, a different path for client machines running Windows 95, and another path for Windows for Workgroups. In other words, the paths that each client would see as its share differ according to the client's architecture, as follows:

```
/home/ftp/pub/WinNT
/home/ftp/pub/Win95
/home/ftp/pub/WfWg
```

Using variables in this manner comes in handy if you wish to have different users run custom configurations based on their own unique characteristics or conditions. Samba has 19 variables, as shown in Table 4-1.

Table 4-1. Samba Variables

Variable	Definition
Client variables	
`%a`	Client's architecture (e.g., Samba, WfWg, WinNT, Win95, or UNKNOWN)
`%I`	Client's IP address (e.g., 192.168.220.100)
`%m`	Client's NetBIOS name
`%M`	Client's DNS name
User variables	
`%g`	Primary group of `%u`
`%G`	Primary group of `%U`
`%H`	Home directory of `%u`
`%u`	Current Unix username
`%U`	Requested client username (not always used by Samba)

Table 4-1. Samba Variables (continued)

Variable	Definition
Share variables	
`%p`	Automounter's path to the share's root directory, if different from `%P`
`%P`	Current share's root directory
`%S`	Current share's name
Server variables	
`%d`	Current server process ID
`%h`	Samba server's DNS hostname
`%L`	Samba server's NetBIOS name
`%N`	Home directory server, from the automount map
`%v`	Samba version
Miscellaneous variables	
`%R`	The SMB protocol level that was negotiated
`%T`	The current date and time

Here's another example of using variables: let's say that there are five clients on your network, but one client, `fred`, requires a slightly different `[homes]` configuration loaded when it connects to the Samba server. With Samba, it's simple to attack such a problem:

```
[homes]
    ...
    include = /usr/local/samba/lib/smb.conf.%m
    ...
```

The `include` option here causes a separate configuration file for each particular NetBIOS machine (`%m`) to be read in addition to the current file. If the hostname of the client machine is `fred`, and if a *smb.conf.fred* file exists in the *`samba_dir/lib/`* directory (or whatever directory you've specified for your configuration files), Samba will insert that configuration file into the default one. If any configuration options are restated in *smb.conf.fred*, those values will override any options previously encountered in that share. Note that we say "previously." If any options are restated in the main configuration file after the `include` option, Samba will honor those restated values for the share in which they are defined.

Here's the important part: if there is no such file, Samba will not generate an error. In fact, it won't do anything at all. This allows you to create only one extra configuration file for `fred` when using this strategy, instead of one for each NetBIOS machine that is on the network.

Machine-specific configuration files can be used both to customize particular clients and to make debugging Samba easier. Consider the latter; if we have one

client with a problem, we can use this approach to give it a private log file with a more verbose logging level. This allows us to see what Samba is doing without slowing down all the other clients or overflowing the disk with useless logs. Remember, with large networks you may not always have the option to restart the Samba server to perform debugging!

You can use each of the variables in Table 4-1 to give custom values to a variety of Samba options. We will highlight several of these options as we move through the next few chapters.

Special Sections

Now that we've gotten our feet wet with variables, there are a few special sections of the Samba configuration file that we should talk about. Again, don't worry if you do not understand each and every configuration options listed below; we'll go over each of them over the course of the upcoming chapters.

The [globals] Section

The `[globals]` section appears in virtually every Samba configuration file, even though it is not mandatory to define one. Any option set in this section of the file will apply to all the other shares, as if the contents of the section were copied into the share itself. There is one catch: other sections can list the same option in their section with a new value; this has the effect of overriding the value specified in the `[globals]` section.

To illustrate this, let's again look at the opening example of the chapter:

```
[global]
    log level = 1
    max log size = 1000
    socket options = TCP_NODELAY IPTOS_LOWDELAY
    guest ok = no
[homes]
    browseable = no
    map archive = yes
[printers]
    path = /usr/tmp
    guest ok = yes
    printable = yes
    min print space = 2000
[test]
    browseable = yes
    read only = yes
    guest ok = yes
    path = /export/samba/test
```

In the previous example, if we were going to connect a client to the `[test]` share, Samba would first read in the `[globals]` section. At that point, it would set the option `guest ok = no` as the global default for each share it encounters throughout the configuration file. This includes the `[homes]` and `[printers]` shares. When it reads in the `[test]` share, however, it would then find the configuration option `guest ok = yes`, and override the default from the `[globals]` section with the value `yes` in the context of the `[pub]` share.

Any option that appears outside of a section (before the first marked section) is also assumed to be a global option.

The [homes] Section

If a client attempts to connect to a share that doesn't appear in the *smb.conf* file, Samba will search for a `[homes]` share in the configuration file. If one exists, the unidentified share name is assumed to be a Unix username, which is queried in the password database of the Samba server. If that username appears, Samba assumes the client is a Unix user trying to connect to his or her home directory on the server.

For example, assume a client machine is connecting to the Samba server `hydra` for the first time, and tries to connect to a share named `[alice]`. There is no `[alice]` share defined in the *smb.conf* file, but there is a `[homes]`, so Samba searches the password database file and finds an `alice` user account is present on the system. Samba then checks the password provided by the client against user `alice`'s Unix password—either with the password database file if it's using non-encrypted passwords, or Samba's *smbpasswd* file if encrypted passwords are in use. If the passwords match, then Samba knows it has guessed right: the user `alice` is trying to connect to her home directory. Samba will then create a share called `[alice]` for her.

The process of using the `[homes]` section to create users (and dealing with their passwords) is discussed in more detail in the Chapter 6, *Users, Security, and Domains*.

The [printers] Section

The third special section is called `[printers]` and is similar to `[homes]`. If a client attempts to connect to a share that isn't in the *smb.conf* file, and its name can't be found in the password file, Samba will check to see if it is a printer share. Samba does this by reading the printer capabilities file (usually */etc/printcap*) to

see if the share name appears there.* If it does, Samba creates a share named after the printer.

Like `[homes]`, this means you don't have to maintain a share for each of your system printers in the *smb.conf* file. Instead, Samba honors the Unix printer registry if you request it to, and provides the registered printers to the client machines. There is, however, an obvious limitation: if you have an account named `fred` and a printer named `fred`, Samba will always find the user account first, even if the client really needed to connect to the printer.

The process of setting up the `[printers]` share is discussed in more detail in Chapter 7, *Printing and Name Resolution*.

Configuration Options

Options in the Samba configuration files fall into one of two categories: *global* or *share*. Each category dictates where an option can appear in the configuration file.

Global
: Global options *must* appear in the `[global]` section and nowhere else. These are options that typically apply to the behavior of the Samba server itself, and not to any of its shares.

Share
: Share options can appear in specific shares, or they can appear in the `[global]` section. If they appear in the `[global]` section, they will define a default behavior for all shares, unless a share overrides the option with a value of its own.

In addition, the values that a configuration option can take can be divided into four categories. They are as follows:

Boolean
: These are simply yes or no values, but can be represented by any of the following: `yes`, `no`, `true`, `false`, `0`, `1`. The values are case insensitive: `YES` is the same as `yes`.

Numerical
: An integer, hexidecimal, or octal number. The standard `0x`*nn* syntax is used for hexadecimal and `0`*nnn* for octal.

String
: A string of case-sensitive characters, such as a filename or a username.

---

* Depending on your system, this file may not be */etc/printcap*. You can use the *testparm* command that comes with Samba to determine the value of the `printcap name` configuration option; this was the default value chosen when Samba was compiled.

Enumerated list

A finite list of known values. In effect, a boolean is an enumerated list with only two values.

Configuration File Options

Samba has well over 200 configuration options at its disposal. So let's start off easy by introducing some of the options you can use to modify the configuration file itself.

As we hinted earlier in the chapter, configuration files are by no means static. You can instruct Samba to include or even replace configuration options as it is processing them. The options to do this are summarized in Table 4-2.

Table 4-2. Configuration File Options

Option	Parameters	Function	Default	Scope
`config file`	string (fully-qualified name)	Sets the location of a configuration file to use instead of the current one.	None	Global
`include`	string (fully-qualified name)	Specifies an additional segment of configuration options to be included at this point in the configuration file.	None	Global
`copy`	string (name of share)	Allows you to clone the configuration options of another share in the current share.	None	Share

config file

The global `config file` option specifies a replacement configuration file that will be loaded when the option is encountered. If the target file exists, the remainder of the current configuration file, as well as the options encounter so far, will be discarded; Samba will configure itself entirely with the options in the new file. The `config file` option takes advantage of the variables above, which is useful in the event that you want load a special configuration file based on the machine name or user of the client that it connecting.

For example, the following line instructs Samba to use a configuration file specified by the NetBIOS name of the client connecting, if such a file exists. If it does, options specified in the original configuration file are ignored. The following example attempts to lead a new configuration file based on the client's NetBIOS name:

```
[global]
    config file = /usr/local/samba/lib/smb.conf.%m
```

If the configuration file specified does not exist, the option is ignored and Samba will continue to configure itself based on the current file.

include

This option, discussed in greater detail earlier, copies the target file into the current configuration file at the point specified, as shown in Figure 4-1. This option also takes advantage of the variables specified earlier in the chapter, which is useful in the event that you want load configuration options based on the machine name or user of the client that it connecting. You can use this option as follows:

```
[global]
    include = /usr/local/samba/lib/smb.conf.%m
```

If the configuration file specified does not exist, the option is ignored. Remember that any option specified previously is overridden. In Figure 4-1, all three options will override their previous values.

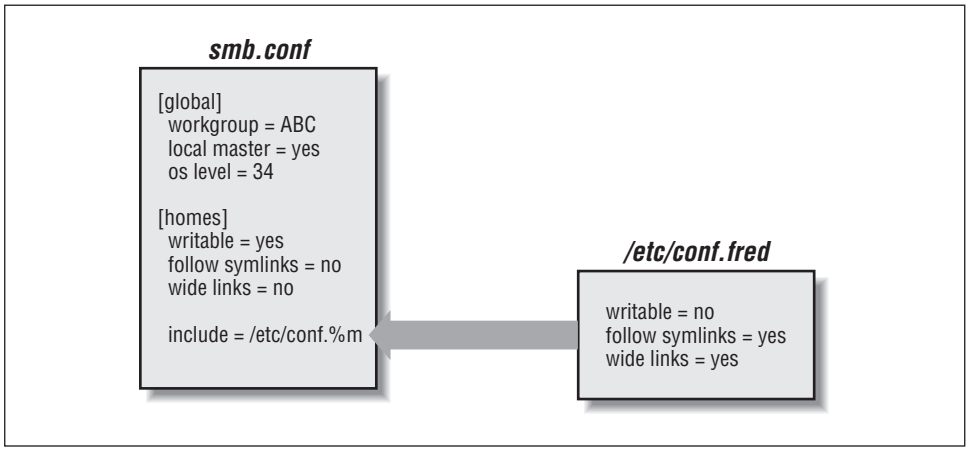

Figure 4-1. The include option in a Samba configuration file

The `include` option cannot understand the variables `%u` (user), `%p` (current share's rout directory), or `%s` (current share) because they are not set at the time the file is read.

copy

The `copy` configuration option allows you to clone the configuration options of the share name that you specify in the current share. The target share must appear earlier in the configuration file than the share that is performing the copy. For example:

```
[template]
    writable = yes
```

```
        browsable = yes
        valid users = andy, dave, peter

    [data]
        path = /usr/local/samba
        copy = template
```

Note that any options in the share that invoked the `copy` directive will override those in the cloned share; it does not matter whether they appear before or after the `copy` directive.

Server Configuration

Now it's time to begin configuring your Samba server. Let's introduce three basic configuration options that can appear in the `[global]` section of your *smb.conf* file:

```
[global]
    #  Server configuration parameters
    netbios name = HYDRA
    server string = Samba %v on (%L)
    workgroup = SIMPLE
```

This configuration file is pretty simple; it advertises the Samba server on a NBT network under the NetBIOS name `hydra`. In addition, the machine belongs to the workgroup SIMPLE and displays a description to clients that includes the Samba version number as well as the NetBIOS name of the Samba server.

If you had to enter `encrypt passwords=yes` in your earlier configuration file, you should do so here as well.

Go ahead and try this configuration file. Create a file named *smb.conf* under the */usr/local/samba/lib* directory with the text listed above. Then reset the Samba server and use a Windows client to verify the results. Be sure that your Windows clients are in the SIMPLE workgroup as well. After clicking on the Network Neighborhood on a Windows client, you should see a window similar to Figure 4-2. (In this figure, `phoenix` and `chimaera` are our Windows clients.)

You can verify the `server string` by listing the details of the Network Neighborhood window (select the Details menu item under the View menu), at which point you should see a window similar to Figure 4-3.

Figure 4-2. Network Neighborhood showing the Samba server

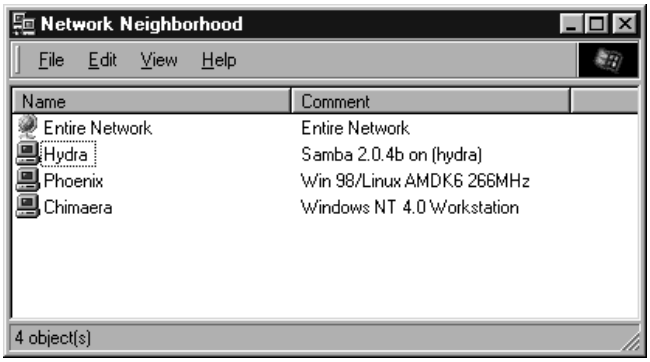

Figure 4-3. Network Neighborhood details listing

If you were to click on the Hydra icon, a window should appear that shows the services that it provides. In this case, the window would be completely empty because there are no shares on the server yet.

Server Configuration Options

Table 4-3 summarizes the server configuration options introduced previously. Note that all three of these options are global in scope; in other words, they must appear in the `[global]` section of the configuration file.

Table 4-3. Server Configuration Options

Option	Parameters	Function	Default	Scope
`netbios name`	string	Sets the primary NetBIOS name of the Samba server.	Server DNS hostname	Global

Table 4-3. Server Configuration Options (continued)

Option	Parameters	Function	Default	Scope
`server string`	string	Sets a descriptive string for the Samba server.	`Samba %v`	Global
`workgroup`	string	Sets the NetBIOS group of machines that the server belongs to.	Defined at compile time	Global

netbios name

The `netbios name` option allows you to set the NetBIOS name of the server. For example:

```
netbios name = YORKVM1
```

The default value for this configuration option is the server's hostname; that is, the first part of its complete DNS machine name. For example, a machine with the DNS name `ruby.ora.com` would be given the NetBIOS name `RUBY` by default. While you can use this option to restate the machine's NetBIOS name in the configuration file (as we did previously), it is more commonly used to assign the Samba server a NetBIOS name other than its current DNS name. Remember that the name given must follow the rules for valid NetBIOS machine names as outlines in Chapter 1, *Learning the Samba*.

Changing the NetBIOS name of the server is not recommended unless you have a good reason. One such reason might be if the hostname of the machine is not unique because the LAN is divided over two or more DNS domains. For example, YORKVM1 is a good NetBIOS candidate for *vm1.york.example.com* to differentiate it from *vm1.falkirk.example.com*, which has the same hostname but resides in a different DNS domain.

Another use of this option is for relocating SMB services from a dead or retired machine. For example, if `SALES` is the SMB server for the department, and it suddenly dies, you could immediately reset `netbios name = SALES` on a backup Samba machine that's taking over for it. Users won't have to change their drive mappings to a different machine; new connections to `SALES` will simply go to the new machine.

server string

The `server string` parameter defines a comment string that will appear next to the server name in both the Network Neighborhood (when shown with the Details menu) and the comment entry of the Microsoft Windows print manager. You can use the standard variables to provide information in the description. For example, our entry earlier was:

```
[global]
    server string = Samba %v on (%h)
```

The default for this option simply presents the current version of Samba and is equivalent to:

```
server string = Samba %v
```

workgroup

The `workgroup` parameter sets the current workgroup where the Samba server will advertise itself. Clients that wish to access shares on the Samba server should be on the same NetBIOS workgroup. Remember that workgroups are really just NetBIOS group names, and must follow the standard NetBIOS naming conventions outlined in Chapter 1. For example:

```
[global]
    workgroup = SIMPLE
```

The default option for this parameter is set at compile time. If the entry is not changed in the makefile, it will be `WORKGROUP`. Because this tends to be the workgroup name of every unconfigured NetBIOS network, we recommend that you always set your workgroup name in the Samba configuration file.*

Disk Share Configuration

We mentioned in the previous section that there were no disk shares on the `hydra` server. Let's continue with the configuration file and create an empty disk share called [`data`]. Here are the additions that will do it:

```
[global]
    netbios name = HYDRA
    server string = Samba %v on (%L)
    workgroup = SIMPLE

[data]
    path = /export/samba/data
    comment = Data Drive
    volume = Sample-Data-Drive
    writeable = yes
    guest ok = yes
```

The `[data]` share is typical for a Samba disk share. The share maps to a directory on the Samba server: */export/samba/data*. We've also provided a comment that describes the share as a `Data Drive`, as well as a volume name for the share itself.

The share is set to writeable so that users can write data to it; the default with Samba is to create a read-only share. As a result, this option needs to be explicitly set for each disk share you wish to make writeable.

---

* We should also mention that it is an inherently bad idea to have a workgroup that shares the same name as a server.

You may have noticed that we set the `guest ok` parameter to `yes`. While this isn't very security-conscious, there are some password issues that we need to understand before setting up individual users and authentication. For the moment, this will sidestep those issues and let anyone connect to the share.

Go ahead and make these additions to your configuration file. In addition, create the */export/samba/data* directory as root on your Samba machine with the following commands:

```
# mkdir /export/samba/data
# chmod 777 /export/samba/data
```

Now, if you connect to the `hydra` server again (you can do this by clicking on its icon in the Windows Network Neighborhood), you should see a single share listed entitled `data`, as shown in Figure 4-4. This share should also have read/write access to it. Try creating or copying a file into the share. Or, if you're really feeling adventurous, you can even try mapping a network drive to it!

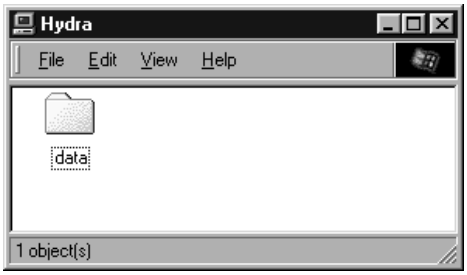

Figure 4-4. The initial data share on the Samba server

Disk Share Configuration Options

The basic Samba configuration options for disk shares previously introduced are listed in Table 4-4.

Table 4-4. Basic Share Configuration Options

Option	Parameters	Function	Default	Scope
`path (directory)`	string (fully-qualified pathname)	Sets the Unix directory that will be provided for a disk share or used for spooling by a printer share	`/tmp`	Share
`guest ok (public)`	boolean	If set to `yes`, authentication is not needed to access this share	`no`	Share
`comment`	string	Sets the comment that appears with the share	None	Share

Table 4-4. Basic Share Configuration Options (continued)

Option	Parameters	Function	Default	Scope
`volume`	string	Sets the volume name: the DOS name of the physical drive	Share name	Share
`read only`	boolean	If `yes`, allows read only access to a share.	`yes`	Share
`writeable (write ok)`	boolean	If `no`, allows read only access to a share.	`no`	Share

path

This option, which has the synonym `directory`, indicates the pathname at the root of the file or printing share. You can choose any path on the Samba server, so long as the owner of the Samba process that is connecting has read and write access to that directory. If the path is for a printing share, it should point to a temporary directory where files can be written on the server before being spooled to the target printer (*/tmp* and */var/spool* are popular choices). If this path is for a disk share, the contents of the folder representing the share name on the client will match the content of the directory on the Samba server. For example, if we have the following disk share listed in our configuration file:

```
[network]
    path = /export/samba/network
    writable = yes
    guest ok = yes
```

And the contents of the directory */usr/local/network* on the Unix side are:

```
$ ls -al /export/samba/network
drwxrwxrwx  9  root   nobody  1024  Feb 16 17:17  .
drwxr-xr-x  9  nobody nobody  1024  Feb 16 17:17  ..
drwxr-xr-x  9  nobody nobody  1024  Feb 16 17:17  quicken
drwxr-xr-x  9  nobody nobody  1024  Feb 16 17:17  tax98
drwxr-xr-x  9  nobody nobody  1024  Feb 16 17:17  taxdocuments
```

Then we should see the equivalent of Figure 4-5 on the client side.

guest ok

This option (which has an older synonym `public`) allows or prohibits guest access to a share. The default value is `no`. If set to `yes`, it means that no username or password will be needed to connect to the share. When a user connects, the access rights will be equivalent to the designated guest user. The default account to which Samba offers the share is `nobody`. However, this can be reset with the

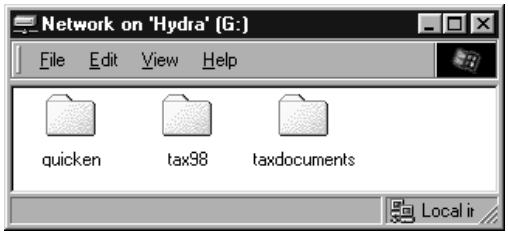

Figure 4-5. Windows client view of a network filesystem specified by path

`guest account` configuration option. For example, the following lines allow guest user access to the `[accounting]` share with the permissions of the *ftp* account:

```
[global]
    guest account = ftp
[accounting]
    path = /usr/local/account
    guest ok = yes
```

Note that users can still connect to the share using a valid username/password combination. If successful, they will hold the access rights granted by their own account and not the guest account. If a user attempts to log in and fails, however, he or she will default to the access rights of the guest account. You can mandate that every user who attaches to the share will be using the guest account (and will have the permissions of the guest) by setting the option `guest only = yes`.

comment

The `comment` option allows you to enter a comment that will be sent to the client when it attempts to browse the share. The user can see the comment by listing Details on the share folder under the appropriate computer in the Windows Network Neighborhood, or type the command `NET VIEW` at an MS-DOS prompt. For example, here is how you might insert a comment for a `[network]` share:

```
[network]
    comment = Network Drive
    path = /export/samba/network
```

This yields a folder similar to Figure 4-6 on the client side. Note that with the current configuration of Windows, this comment will not be shown once a share is mapped to a Windows network drive.

Be sure not to confuse the `comment` option, which documents a Samba server's shares, with the `server string` option, which documents the server itself.

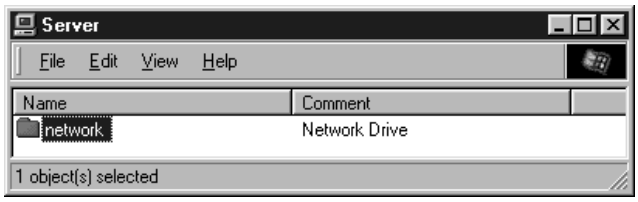

Figure 4-6. Windows client view of a share comment

volume

This option allows you to specify the volume name of the share as reported by SMB. This normally resolves to the name of the share given in the *smb.conf* file. However, if you wish to name it something else (for whatever reason) you can do so with this option.

For example, an installer program may check the volume name of a CD-ROM to make sure the right CD-ROM is in the drive before attempting to install it. If you copy the contents of the CD-ROM into a network share, and wish to install from there, you can use this option to get around the issue:

```
[network]
    comment = Network Drive
    volume = ASVP-102-RTYUIKA
    path = /home/samba/network
```

read only and writeable

The options `read only` and `writeable` (or `write ok`) are really two ways of saying the same thing, but approached from opposite ends. For example, you can set either of the following options in the `[global]` section or in an individual share:

```
read only = yes
writeable = no
```

If either option is set as shown, data can be read from a share, but cannot be written to it. You might think you would need this option only if you were creating a read-only share. However, note that this read-only behavior is the *default* action for shares; if you want to be able to write data to a share, you must explicitly specify one of the following options in the configuration file for each share:

```
read only = no
writeable = yes
```

Note that if you specify more than one occurrence of either option, Samba will adhere to the last value it encounters for the share.

Networking Options with Samba

If you're running Samba on a multi-homed machine (that is, one on multiple subnets), or even if you want to implement a security policy on your own subnet, you should take a close look at the networking configuration options:

For the purposes of this exercise, let's assume that our Samba server is connected to a network with more than one subnet. Specifically, the machine can access both the 192.168.220.* and 134.213.233.* subnets. Here are our additions to the ongoing configuration file for the networking configuration options:

```
[global]
    netbios name = HYDRA
    server string = Samba %v on (%L)
    workgroup = SIMPLE

    #  Networking configuration options
    hosts allow = 192.168.220. 134.213.233. localhost
    hosts deny = 192.168.220.102
    interfaces = 192.168.220.100/255.255.255.0 \
                    134.213.233.110/255.255.255.0
    bind interfaces only = yes

[data]
    path = /home/samba/data
    guest ok = yes
    comment = Data Drive
    volume = Sample-Data-Drive
    writeable = yes
```

Let's first talk about the `hosts allow` and `hosts deny` options. If these options sound familiar, you're probably thinking of the *hosts.allow* and *hosts.deny* files that are found in the */etc* directories of many Unix systems. The purpose of these options is identical to those files; they provide a means of security by allowing or denying the connections of other hosts based on their IP addresses. Why not just use the *hosts.allow* and *hosts.deny* files themselves? Because there may be services on the server that you want others to access without giving them access Samba's disk or printer shares

With the `hosts allow` option above, we've specified a cropped IP address: 192.168.220. (Note that there is still a third period; it's just missing the fourth number.) This is equivalent to saying: "All hosts on the 192.168.220 subnet." However, we've explicitly specified in a hosts deny line that 192.168.220.102 is not to be allowed access.

You might be wondering: why will 192.168.220.102 be denied even though it is still in the subnet matched by the `hosts allow` option? Here is how Samba sorts out the rules specified by `hosts allow` and `hosts deny`:

1. If there are no `allow` or `deny` options defined anywhere in *smb.conf*, Samba will allow connections from any machine allowed by the system itself.
2. If there are `hosts allow` or `hosts deny` options defined in the `[global]` section of *smb.conf*, they will apply to all shares, even if the shares have an overriding option defined.
3. If there is only a `hosts allow` option defined for a share, only the hosts listed will be allowed to use the share. All others will be denied.
4. If there is only a `hosts deny` option defined for a share, any machine which is not on the list will be able to use the share.
5. If both a `hosts allow` and `hosts deny` option are defined, a host must appear in the allow list and not appear in the deny list (in any form) in order to access the share. Otherwise, the host will not be allowed.

Take care that you don't explicitly allow a host to access a share, but then deny access to the entire subnet of which the host is part.

Let's look at another example of that final item. Consider the following options:

```
hosts allow = 111.222.
hosts deny = 111.222.333.
```

In this case, only the hosts that belong to the subnet 111.222.*.* will be allowed access to the Samba shares. However, if a client belongs to the 111.222.333.* subnet, it will be denied access, even though it still matches the qualifications outlined by `hosts allow`. The client must appear on the `hosts allow` list and *must not* appear on the `hosts deny` list in order to gain access to a Samba share. If a computer attempts to access a share to which it is not allowed access, it will receive an error message.

The other two options that we've specified are the `interfaces` and the `bind interface only` address. Let's look at the `interfaces` option first. Samba, by default, sends data only from the primary network interface, which in our example is the 192.168.220.100 subnet. If we would like it to send data to more than that one interface, we need to specify the complete list with the `interfaces` option. In the previous example, we've bound Samba to interface with both subnets (192.168.220 and 134.213.233) on which the machine is operating by specifying the other network interface address: 134.213.233.100. If you have more than one interface on your computer, you should always set this option as there is no guarantee that the primary interface that Samba chooses will be the right one.

Finally, the `bind interfaces only` option instructs the *nmbd* process not to accept any broadcast messages other than those subnets specified with the `interfaces` option. Note that this is different from the `hosts allow` and `hosts deny` options, which prevent machines from making connections to services, but not from receiving broadcast messages. Using the `bind interfaces only` option is a way to shut out even datagrams from foreign subnets from being received by the Samba server. In addition, it instructs the *smbd* process to bind to only the interface list given by the *interfaces* option. This restricts the networks that Samba will serve.

Networking Options

The networking options we introduced above are summarized in Table 4-5.

Table 4-5. Networking Configuration Options

Option	Parameters	Function	Default	Scope
`hosts allow` (`allow hosts`)	string (list of hostnames)	Specifies the machines that can connect to Samba.	none	Share
`hosts deny` (`deny hosts`)	string (list of hostnames)	Specifies the machines that cannot connect to Samba.	none	Share
`interfaces`	string (list of IP/netmask combinations)	Sets the network interfaces Samba will respond to. Allows correcting defaults.	system-dependent	Global
`bind interfaces only`	boolean	If set to `yes`, Samba will bind only to those interfaces specified by the `interfaces` option.	`no`	Global
`socket address`	string (IP address)	Sets IP address to listen on, for use with multiple virtual interfaces on a server.	none	Global

hosts allow

The `hosts allow` option (sometimes written as `allow hosts`) specifies the machines that have permission to access shares on the Samba server, written as a comma- or space-separated list of names of machines or their IP addresses. You can gain quite a bit of security by simply placing your LAN's subnet address in this option. For example, we specified the following in our example:

```
hosts allow = 192.168.220. localhost
```

Note that we placed `localhost` after the subnet address. One of the most common mistakes when attempting to use the `hosts allow` option is to accidentally disallow the Samba server from communicating with itself. The *smbpasswd* program will occasionally need to connect to the Samba server as a client in order to change a user's encrypted password. In addition, local browsing propagation requires local host access. If this option is enabled and the localhost address is not specified, the locally-generated packets requesting the change of the encrypted password will be discarded by Samba, and browsing propagation will not work properly. To avoid this, explicitly allow the loopback address (either `localhost` or `127.0.0.1`) to be used.*

You can specify any of the following formats for this option:

- Hostnames, such as `ftp.example.com`.
- IP addresses, like `130.63.9.252`.
- Domain names, which can be differentiated from individual hostnames because they start with a dot. For example, `.ora.com` represents all machines within the *ora.com* domain.
- Netgroups, which start with an at-sign, such as `@printerhosts`. Netgroups are available on systems running yellow pages/NIS or NIS+, but rarely otherwise. If netgroups are supported on your system, there should be a `netgroups` manual page that describes them in more detail.
- Subnets, which end with a dot. For example, `130.63.9.` means all the machines whose IP addresses begin with 130.63.9.
- The keyword `ALL`, which allows any client access.
- The keyword `EXCEPT` followed by more one or more names, IP addresses, domain names, netgroups, or subnets. For example, you could specify that Samba allow all hosts except those on the 192.168.110 subnet with `hosts allow = ALL EXCEPT 192.168.110.` (remember the trailing dot).

Using the `ALL` keyword is almost always a bad idea, since it means that anyone on any network can browse your files if they guess the name of your server.

Note that there is no default value for the `hosts allow` configuration option, although the default course of action in the event that neither option is specified is to allow access from all sources. In addition, if you specify this option in the `[global]` section of the configuration file, it will override any `hosts allow` options defined shares.

* Starting with Samba 2.0.5, `localhost` will automatically be allowed unless it is explicitly denied.

hosts deny

The `hosts deny` option (also `deny hosts`) specifies machines that do not have permission to access a share, written as a comma- or space-separated list of machine names or their IP addresses. Use the same format as specifying clients as the `hosts allow` option above. For example, to restrict access to the server from everywhere but *example.com*, you could write:

```
hosts deny = ALL EXCEPT .example.com
```

Like `hosts allow`, there is no default value for the `hosts deny` configuration option, although the default course of action in the event that neither option is specified is to allow access from all sources. Also, if you specify this option in the `[global]` section of the configuration file, it will override any `hosts deny` options defined in shares. If you wish to deny *hosts* access to specific shares, omit both the `hosts allow` and `hosts deny` options in the `[global]` section of the configuration file.

interfaces

The `interfaces` option outlines the network addresses to which you want the Samba server to recognize and respond. This option is handy if you have a computer that resides on more than one network subnet. If this option is not set, Samba searches for the primary network interface of the server (typically the first Ethernet card) upon startup and configures itself to operate on only that subnet. If the server is configured for more than one subnet and you do not specify this option, Samba will only work on the first subnet it encounters. You must use this option to force Samba to serve the other subnets on your network.

The value of this option is one or more sets of IP address/netmask pairs, such as the following:

```
interfaces = 192.168.220.100/255.255.255.0 192.168.210.30/255.255.255.0
```

You can optionally specify a CIDR format bitmask, as follows:

```
interfaces = 192.168.220.100/24 192.168.210.30/24
```

The bitmask number specifies the first number of bits that will be turned on in the netmask. For example, the number 24 means that the first 24 (of 32) bits will be activated in the bit mask, which is the same as saying 255.255.255.0. Likewise, 16 would be equal to 255.255.0.0, and 8 would be equal to 255.0.0.0.

This option may not work correctly if you are using DHCP.

bind interfaces only

The `bind interfaces only` option can be used to force the *smbd* and *nmbd* processes to serve SMB requests to only those addresses specified by the `interfaces` option. The *nmbd* process normally binds to the all addresses interface (0.0.0.0.) on ports 137 and 138, allowing it to receive broadcasts from anywhere. However, you can override this behavior with the following:

```
bind interfaces only = yes
```

This will cause both Samba processes to ignore any packets whose origination address does not match the broadcast address(es) specified by the `interfaces` option, including broadcast packets. With *smbd*, this option will cause Samba to not serve file requests to subnets other than those listed in the `interfaces` option. You should avoid using this option if you want to allow temporary network connections, such as those created through SLIP or PPP. It's very rare that this option is needed, and it should only be used by experts.

If you set `bind interfaces only` to `yes`, you should add the localhost address (127.0.01) to the "interfaces" list. Otherwise, *smbpasswd* will be unable to connect to the server using its default mode in order to change a password.

socket address

The `socket address` option dictates which of the addresses specified with the `interfaces` parameter Samba should listen on for connections. Samba accepts connections on all addresses specified by default. When used in an *smb.conf* file, this option will force Samba to listen on only one IP address. For example:

```
interfaces = 192.168.220.100/24 192.168.210.30/24
socket address = 192.168.210.30
```

This option is a programmer's tool and we recommend that you do not use it.

Virtual Servers

Virtual servers are a technique for creating the illusion of multiple NetBIOS servers on the network, when in reality there is only one. The technique is simple to implement: a machine simply registers more than one NetBIOS name in association with its IP address. There are tangible benefits to doing this.

The accounting department, for example, might have an `accounting` server, and clients of it would see just the accounting disks and printers. The marketing

department could have their own server, `marketing`, with their own reports, and so on. However, all the services would be provided by one medium-sized Unix workstation (and one relaxed administrator), instead of having one small server and one administrator per department.

Samba will allow a Unix server to use more than one NetBIOS name with the `netbios aliases` option. See Table 4-6.

Table 4-6. Virtual Server Configuration Options

Option	Parameters	Function	Default	Scope
`netbios aliases`	List of NetBIOS names	Additional NetBIOS names to respond to, for use with multiple "virtual" Samba servers.	None	Global

netbios aliases

The `netbios aliases` option can be used to give the Samba server more than one NetBIOS name. Each NetBIOS name listed as a value will be displayed in the Network Neighborhood of a browsing machine. When a connection is requested to any machine, however, it will connect to the same Samba server.

This might come in handy, for example, if you're transferring three departments' data to a single Unix server with modern large disks, and are retiring or reallocating the old NT servers. If the three servers are called `sales`, `accounting`, and `admin`, you can have Samba represent all three servers with the following options:

```
[global]
    netbios aliases = sales accounting admin
    include = /usr/local/samba/lib/smb.conf.%L
```

See Figure 4-7 for what the Network Neighborhood would display from a client. When a client attempts to connect to Samba, it will specify the name of the server that it's trying to connect to, which you can access through the `%L` variable. If the requested server is `sales`, Samba will include the */usr/local/samba/lib/smb.conf.sales* file. This file might contain global and share declarations exclusively for the sales team, such as the following:

```
[global]
    workgroup = SALES
    hosts allow = 192.168.10.255

[sales1998]
    path = /usr/local/samba/sales/sales1998/
...
```

This particular example would set the workgroup to SALES as well, and set the IP address to allow connections only from the SALES subnet (192.168.10). In addition, it would offer shares specific to the sales department.

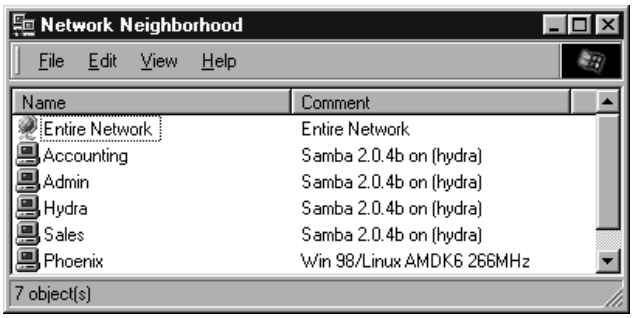

Figure 4-7. Using NetBIOS aliases for a Samba server

Logging Configuration Options

Occasionally, we need to find out what Samba is up to. This is especially true when Samba is performing an unexpected action or is not performing at all. To find out this information, we need to check Samba's log files to see exactly why it did what it did.

Samba log files can be as brief or verbose as you like. Here is an example of what a Samba log file looks like:

```
[1999/07/21 13:23:25, 3] smbd/service.c:close_cnum(514)
  phoenix (192.168.220.101) closed connection to service IPC$
[1999/07/21 13:23:25, 3] smbd/connection.c:yield_connection(40)
  Yielding connection to IPC$
[1999/07/21 13:23:25, 3] smbd/process.c:process_smb(615)
  Transaction 923 of length 49
[1999/07/21 13:23:25, 3] smbd/process.c:switch_message(448)
  switch message SMBread (pid 467)
[1999/07/21 13:23:25, 3] lib/doscalls.c:dos_ChDir(336)
  dos_ChDir to /home/samba
[1999/07/21 13:23:25, 3] smbd/reply.c:reply_read(2199)
  read fnum=4207 num=2820 nread=2820
[1999/07/21 13:23:25, 3] smbd/process.c:process_smb(615)
  Transaction 924 of length 55
[1999/07/21 13:23:25, 3] smbd/process.c:switch_message(448)
  switch message SMBreadbraw (pid 467)
[1999/07/21 13:23:25, 3] smbd/reply.c:reply_readbraw(2053)
  readbraw fnum=4207 start=130820 max=1276 min=0 nread=1276
[1999/07/21 13:23:25, 3] smbd/process.c:process_smb(615)
  Transaction 925 of length 55
[1999/07/21 13:23:25, 3] smbd/process.c:switch_message(448)
  switch message SMBreadbraw (pid 467)
```

Many of these options are of use only to Samba programmers. However, we will go over the meaning of some of these entries in more detail in Chapter 9, *Troubleshooting Samba*.

Samba contains six options that allow users to describe how and where logging information should be written. Each of these options are global options and cannot appear inside a share definition. Here is an up-to-date configuration file that covers each of the share and logging options that we've seen so far:

```
[global]
    netbios name = HYDRA
    server string = Samba %v on (%I)
    workgroup = SIMPLE

    #  Networking configuration options
    hosts allow = 192.168.220. 134.213.233. localhost
    hosts deny = 192.168.220.102
    interfaces = 192.168.220.100/255.255.255.0 \
                    134.213.233.110/255.255.255.0
    bind interfaces only = yes

    # Debug logging information
    log level = 2
    log file = /var/log/samba.log.%m
    max log size = 50
    debug timestamp = yes

[data]
    path = /home/samba/data
    browseable = yes
    guest ok = yes
    comment = Data Drive
    volume = Sample-Data-Drive
    writeable = yes
```

Here, we've added a custom log file that reports information up to debug level 2. This is a relatively light debugging level. The logging level ranges from 1 to 10, where level 1 provides only a small amount of information and level 10 provides a plethora of low-level information. Level 2 will provide us with useful debugging information without wasting disk space on our server. In practice, you should avoid using log levels greater than 3 unless you are programming Samba.

This file is located in the */var/log* directory thanks to the `log file` configuration option. However, we can use variable substitution to create log files specifically for individual users or clients, such as with the `%m` variable in the following line:

```
log file = /usr/local/logs/samba.log.%m
```

Isolating the log messages can be invaluable in tracking down a network error if you know the problem is coming from a specific machine or user.

We've added another precaution to the log files: no one log file can exceed 50 kilobytes in size, as specified by the `max log size` option. If a log file exceeds this size, the contents are moved to a file with the same name but with the suffix

.old appended. If the *.old* file already exists, it is overwritten and its contents are lost. The original file is cleared, waiting to receive new logging information. This prevents the hard drive from being overwhelmed with Samba log files during the life of our daemons.

For convenience, we have decided to leave the debug timestamp in the logs with the `debug timestamp` option, which is the default behavior. This will place a timestamp next to each message in the logging file. If we were not interested in this information, we could specify `no` for this option instead.

Using syslog

If you wish to use the system logger (*syslog*) in addition to or in place of the standard Samba logging file, Samba provides options for this as well. However, to use *syslog*, the first thing you will have to do is make sure that Samba was built with the `configure --with-syslog` option. See Chapter 2 for more information on configuring and compiling Samba.

Once that is done, you will need to configure your */etc/syslog.conf* to accept logging information from Samba. If there is not already a `daemon.*` entry in the */etc/syslog.conf* file, add the following:

```
daemon.*        /var/log/daemon.log
```

This specifies that any logging information from system daemons will be stored in the */var/log/daemon.log* file. This is where the Samba information will be stored as well. From there, you can specify the following global option in your configuration file:

```
syslog = 2
```

This specifies that any logging messages with a level of 1 will be sent to both the *syslog* and the Samba logging files. (The mappings to *syslog* priorities are described in the upcoming section "syslog.") Let's assume that we set the regular `log level` option above to 4. Any logging messages with a level of 2, 3, or 4 will be sent to the Samba logging files, but not to the *syslog*. Only level 1 logging messages will be sent to both. If the `syslog` value exceeds the `log level` value, nothing will be written to the *syslog*.

If you want to specify that messages be sent only to *syslog*—and not to the standard Samba logging files—you can place this option in the configuration file:

```
syslog only = yes
```

If this is the case, any logging information above the number specified in the `syslog` option will be discarded, just like the `log level` option.

Logging Configuration Options

Table 4-7 lists each of the logging configuration options that Samba can use.

Table 4-7. Global Configuration Options

Option	Parameters	Function	Default	Scope
`log file`	string (fully-qualified filename)	Sets the name and location of the log file that Samba is to use. Uses standard variables.	Specified in Samba makefile	Global
`log level (debug level)`	numerical (0–10)	Sets the amount of log/debug messages that are sent to the log file. 0 is none, 3 is considerable.	`1`	Global
`max log size`	numerical (size in KB)	Sets the maximum size of log file. After the log exceeds this size, the file will be renamed to *.bak* and a new log file started.	`5000`	Global
`debug timestamp (timestamp logs)`	boolean	If no, doesn't timestamp logs, making them easier to read during heavy debugging.	`yes`	Global
`syslog`	numerical (0–10)	Sets level of messages sent to *syslog*. Those levels below `syslog level` will be sent to the system logger.	`1`	Global
`syslog only`	boolean	If yes, uses *syslog* entirely and sends no output to the standard Samba log files.	`no`	Global

log file

On our server, Samba outputs log information to text files in the *var* subdirectory of the Samba home directory, as set by the makefile during the build. The `log file` option can be used to reset the name of the log file to another location. For example, to reset the name and location of the Samba log file to */usr/local/logs/samba.log*, you could use the following:

```
[global]
    log file = /usr/local/logs/samba.log
```

You may use variable substitution to create log files specifically for individual users or clients.

You can override the default log file location using the `-l` command-line switch when either daemon is started. However, this does not override the `log file` option. If you do specify this parameter, initial logging information will be sent to the file specified after `-l` (or the default specified in the Samba makefile) until the daemons have processed the *smb.conf* file and know to redirect it to a new log file.

log level

The `log level` option sets the amount of data to be logged. Normally this is left at 0 or 1. However, if you have a specific problem you may want to set it at 3, which provides the most useful debugging information you would need to track down a problem. Levels above 3 provide information that's primarily for the developers to use for chasing internal bugs, and slows down the server considerably. Therefore, we recommend that you avoid setting this option to anything above 3.

```
[global]
log file = /usr/local/logs/samba.log.%m
log level = 3
```

max log size

The `max log size` option sets the maximum size, in kilobytes, of the debugging log file that Samba keeps. When the log file exceeds this size, the current log file is renamed to add an *.old* extension (erasing any previous file with that name) and a new debugging log file is started with the original name. For example:

```
[global]
log file = /usr/local/logs/samba.log.%m
max log size = 1000
```

Here, if the size of any log file exceeds one megabyte in size, Samba renames the log file *samba.log.machine-name.old* and a new log file is generated. If there was a file there previously with the *.old* extension, Samba deletes it. We highly recommend setting this option in your configuration files because debug logging (even at lower levels) can covertly eat away at your available disk space. Using this option protects unwary administrators from suddenly discovering that most of their disk space has been swallowed up by a single Samba log file.

;debug timestamp or timestamp logs

If you happen to be debugging a network problem and you find that the date-stamp and timestamp information within the Samba log lines gets in the way, you can turn it off by giving either the `timestamp logs` or the `debug timestamp` option (they're synonymous) a value of `no`. For example, a regular Samba log file presents its output in the following form:

```
12/31/98 12:03:34 hydra (192.168.220.101) connect to server network as user davecb
```

With a `no` value for this option, the output would appear without the datestamp or the timestamp:

```
hydra (192.168.220.101) connect to server network as user davecb
```

syslog

The `syslog` option causes Samba log messages to be sent to the Unix system logger. The type of log information to be sent is specified as the parameter for this argument. Like the `log level` option, it can be a number from 0 to 10. Logging information with a level less than the number specified will be sent to the system logger. However, debug logs equal to or above the `syslog` level, but less than log level, will still be sent to the standard Samba log files. To get around this, use the `syslog only` option. For example:

```
[global]
    log level = 3
    syslog = 1
```

With this, all logging information with a level of 0 would be sent to the standard Samba logs and the system logger, while information with levels 1, 2, and 3 would be sent only to the standard Samba logs. Levels above 3 are not logged at all. Note that all messages sent to the system logger are mapped to a priority level that the *syslog* process understands, as shown in Table 4-8. The default level is 1.

Table 4-8. Syslog Priority Conversion

Log Level	Syslog Priority
0	LOG_ERR
1	LOG_WARNING
2	LOG_NOTICE
3	LOG_INFO
4 and above	LOG_DEBUG

If you wish to use *syslog*, you will have to run `configure --with-syslog` when compiling Samba, and you will need to configure your */etc/syslog.conf* to suit. (See the section "Using syslog," earlier in this chapter.)

syslog only

The `syslog only` option tells Samba not to use the regular logging files—the system logger only. To enable this, specify the following option in the global ection of the Samba configuration file:

```
[global]
    syslog only = yes
```