Chapter 6. The Samba Configuration File

In previous 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 300 unique options (also called parameters). Some of these options you will use and change frequently; others you might never use, depending 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 options to create and modify disk shares. Subsequent chapters will discuss browsing, how to configure users, security, printing, and other topics related to implementing Samba on your network.

The Samba Configuration File

The Samba configuration file, called *smb.conf* by default, uses the same format as Windows *.ini* files. If you have ever worked with a *.ini* file, you will find *smb.conf* easy to create and modify. Even if you haven't, you will find the format to be simple and easy to learn. Here is an example of a Samba configuration file:

```
[global]
    workgroup = METRAN
    encrypt passwords = yes
    wins support = yes
    log level = 1
    max log size = 1000
    read only = no
[homes]
    browsable = no
    map archive = yes
[printers]
    path = /var/tmp
    printable = yes
    min print space = 2000
[test]
    browsable = yes
    read only = yes
    path = /usr/local/samba/tmp
```

This configuration file is based on the one we created in Chapter 2 and sets up a workgroup in which Samba authenticates users using encrypted passwords and the default user-level security method. Samba is providing WINS name server support. We've configured very basic event logging to use a log file not to exceed 1MB in size. The `[homes]` share has been added to allow Samba to create a disk share for the home directory of each user who has a standard Unix account on the server. In addition, each printer registered on the server will be publicly available, as will a single read-only share that maps to the */usr/local/samba/tmp* directory.

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) to which the section refers. For example, the `[test]` and `[homes]` sections are 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 for that share. These options will continue until a new 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. The `public` option is an older option and is vague. It now has a less-confusing synonym `guest ok` (meaning it can be accessed by guests). *Appendix B* contains an alphabetical index of all the configuration options and their meanings.

Whitespace, quotes, and commas

An important item to remember about configuration options is that all whitespace within the *value* is significant. For example, consider the following option:

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

Samba strips away the spaces up to 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 `read only`), 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 end of a configuration option's value, you can do so. Samba will ignore these quotation marks when it encounters them. Never use quotation marks around an option name; Samba will treat this as an error.

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

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

In some cases, you must use one form of separation—sometimes spaces are required, and sometimes commas.

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 *is* case-sensitive. Consequently, the path listed would not be found, and clients could not connect to the share.

Line continuation

You can continue a line in the Samba configuration file using the backslash, like this:

```
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 nonwhitespace character that Samba encounters; in this case, the `o` in `of`.

Comments

You can insert comments in the *smb.conf* configuration file by starting a line with either a hash (`#`) or a semicolon (`;`). For this purpose, 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.

WARNING

Samba does not allow mixing of comment lines and parameters. Be careful not to put comments on the same line as anything else, such as:

```
path = /d # server's data partition
```

Errors such as this, where the parameter value is defined with a string, can be tricky to notice. The *testparm* program won't complain, and the only clues you'll receive are that *testparm* reports the `path` parameter set to `/d # server's data partition`, and the failures that result when clients attempt to access the share.

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. If it finds any changes, they are immediately put into effect.

TIP

Having Samba check the configuration file automatically can be convenient, but it also means that if you edit *smb.conf* directly, you might be immediately changing your network's configuration every time you save the file. If you're making anything more than a minor change, it may be wiser to copy *smb.conf* to a temporary file, edit that, run `testparm` *filename* to check it, and then copy the temporary file back to *smb.conf*. That way, you can be sure to put all your changes into effect at once, and only after you are confident that you have created the exact configuration you wish to implement.

If you don't want to wait for the configuration file to be reloaded automatically, you can force a reload either by sending a hangup signal to the *smbd* and *nmbd* processes or simply by restarting the daemons. Actually, it can be a good idea to restart the daemons because it forces the clients to disconnect and reconnect, ensuring that the new configuration is applied to all clients. We showed you how to restart the daemons in Chapter 2, and sending them a hangup (HUP) signal is very similar. On Linux, it can be done with the command:

```
# killall -HUP smbd nmbd
```

In this case, 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 go into effect immediately either. (This behavior was implemented intentionally because it keeps active clients from being suddenly disconnected or encountering unexpected access problems while a session is open.)

Variables

Because a new copy of the *smbd* daemon is created for each connecting client, it is possible for each client to have its own customized configuration file. Samba allows a limited, yet useful, form of variable substitution in the configuration file to allow information about the Samba server and the client to be included in the configuration at the time the client connects. Inside the configuration file, a variable begins with a percent sign (`%`), followed by a single upper- or lowercase letter, and can be used only on the right side of a configuration option (i.e., after the equal sign). An example is:

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

The `%a` stands for the client system's architecture and will be replaced as shown in Table 6-1.

Table 6-1. %a substitution

Client operating system ("architecture")	Replacement string
Windows for Workgroups	`WfWg`
Windows 95 and Windows 98	`Win95`
Windows NT	`WinNT`
Windows 2000 and Windows XP	`Win2K`
Samba	`Samba`

Any OS not listed earlier	UNKNOWN

In this example, Samba will assign a unique path for the [pub] share to client systems based on what operating system they are running. The paths that each client would see as its share differ according to the client's architecture:

```
/home/ftp/pub/WfwG
/home/ftp/pub/Win95
/home/ftp/pub/WinNT
/home/ftp/pub/Win2K
/home/ftp/pub/Samba
/home/ftp/pub/UNKNOWN
```

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 20 variables, as shown in Table 6-2.

Table 6-2. Samba variables

Variable	Definition
Client variables	
%a	Client's architecture (see Table 6-1)
%I	Client's IP address (e.g., 172.16.1.2)
%m	Client's NetBIOS name
%M	Client's DNS name
User variables	
%u	Current Unix username
%U	Requested client username (not always used by Samba)
%H	Home directory of %u
%g	Primary group of %u
%G	Primary group of %U
Share variables	
%S	Current share's name
%P	Current share's root directory
%p	Automounter's path to the share's root directory, if different from %P
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
%$*var*	The value of environment variable var

Here's another example of using variables: let's say there are five clients on your network, but one client, maya, requires a slightly different [homes] configuration. With Samba, it's simple to handle this:

```
[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 system is maya, and if a *smb.conf.maya* file exists in the */usr/local/samba/lib* directory, Samba will insert that configuration file into the default one. If any configuration options are restated in *smb.conf.maya*, 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.

If the file specified by the include parameter does not exist, 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 maya when using this strategy, instead of one for each client that is on the network.

Client-specific configuration files can be used to customize particular clients. They also can be used to make debugging Samba easier. For example, 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.

You can use the variables in Table 6-2 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 every configuration option listed here; we'll go over each of them in the upcoming chapters.

The [global] Section

The [global] section appears in virtually every Samba configuration file, even though it is not mandatory. There are two purposes for the [global] section. Server-wide settings are defined here, and any options that

apply to shares will be used as a default in all share definitions, unless overridden within the share definition.

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

```
[global]
    workgroup = METRAN
    encrypt passwords = yes
    wins support = yes
    log level = 1
    max log size = 1000
    read only = no
[homes]
    browsable = no
    map archive = yes
[printers]
    path = /var/tmp
    printable = yes
    min print space = 2000
[test]
    browsable = yes
    read only = yes
    path = /usr/local/samba/tmp
```

When a client connects to the `[test]` share, Samba first reads the `[global]` section and sets the option `read only = no` as the global default for each share it encounters throughout the configuration file. This includes the `[homes]` and `[test]` shares. When it reads the definition of the `[test]` share, it then finds the configuration option `read only = yes` and overrides the default from the `[global]` section with the value `yes`.

Any option that appears before the first marked section is assumed to be a global option. This means that the `[global]` section heading is not absolutely required; however, we suggest you always include it for clarity and to ensure future compatibility.

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 a `[homes]` share exists, the unresolved share name is assumed to be a Unix username. If that username appears in the password database on the Samba server, Samba assumes the client is a Unix user trying to connect to her home directory on the server.

For example, assume a client system is connecting to the Samba server `toltec` 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 nonencrypted passwords or with Samba's *smbpasswd* file if encrypted passwords are in use. If the passwords match, 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, with the share's path set to `alice`'s home directory.

The process of using the `[homes]` section to create users (and dealing with their passwords) is discussed in more detail in Chapter 9.

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.[1] If it does, Samba creates a share named after the printer.

This means that as with `[homes]`, you don't have to maintain a share for each system printer in the *smb.conf* file. Instead, Samba honors the Unix printer registry if you ask it to, and it provides the registered printers to the client systems. However, there is a potential difficulty: 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 10.

Configuration Options

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

Global options

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 options

Share options can appear in share definitions, the `[global]` section, or both. 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, configuration options can take three kinds of values. 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`, `1`, or `0`. The values are case-insensitive: `YES` is the same as `yes`.

Numeric

This is a decimal, hexadecimal, or octal number. The standard `0x`*nn* syntax is used for hexadecimal and `0`*nnn* for octal.

String

This is a string of case-sensitive characters, such as a filename or a username.

Configuration File Options

You can instruct Samba to include or replace configuration options as it is processing them. The options to do this are summarized in Table 6-3.

Table 6-3. Configuration file options

Option	Parameters	Function	Default	Scope

`config file`	string (name of file)	Sets the location of a configuration file to use instead of the current one	None	Global
`include`	string (name of file)	Specifies an additional set of configuration options to be included 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 encountered so far, will be discarded, and Samba will configure itself entirely with the options in the new file. Variables can be used with the `config file` option, which is useful in the event that you want to use a special configuration file based on the NetBIOS machine name or user of the client that is 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:

```
[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. This allows a default configuration file to serve most clients, while providing for exceptions with customized configuration files.

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 6-1. This option also can be used with variables. 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. Options in the include file override any option specified previously, but not options that are specified later. In Figure 6-1, all three options will override their previous values.

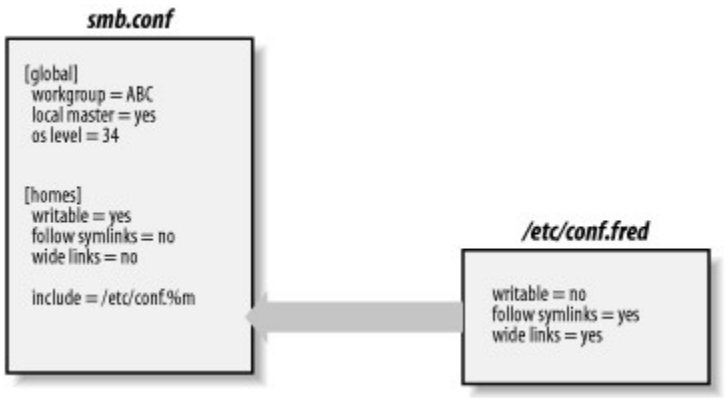

Figure 6-1. The include option in a Samba configuration file

The `include` option does not work with the variables `%u` (user), `%P` (current share's root directory), or `%S` (current share's name) because they are not set at the time the `include` parameter is processed.

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, jay

[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

We will now start from scratch and build a configuration file for our Samba server. First we will introduce three basic configuration options that can appear in the `[global]` section of the *smb.conf* file:

```
[global]
    #  Server configuration parameters
    netbios name = toltec
    server string = Samba %v on %L
    workgroup = METRAN
    encrypt passwords = yes
```

This configuration file is pretty simple; it advertises the Samba server under the NetBIOS name `toltec`. In addition, it places the system in the METRAN workgroup and displays a description to clients that includes the Samba version number, as well as the NetBIOS name of the Samba server.

> **TIP**
>
> If you used the line `encrypt passwords = yes` in your earlier configuration file, you should do so here as well.

If you like, you can go ahead and try this configuration file. Create a file named *smb.conf* under the */usr/local/samba/lib* directory with the text listed earlier. Then restart the Samba server and use a Windows client to verify the results. Be sure that your Windows clients are in the METRAN workgroup as well. After double-clicking the Network Neighborhood on a Windows client, you should see a window similar to Figure 6-2. (In this figure, `Mixtec` is another Samba server, and `Zapotec` is a Windows client.)

Figure 6-2. Network Neighborhood showing Toltec, the Samba server

You can verify the `server string` by listing the details of the Network Neighborhood window (select Details in the View menu). You should see a window similar to Figure 6-3.

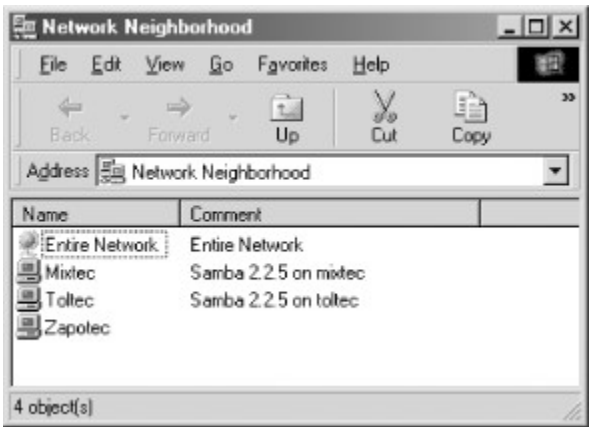

Figure 6-3. Network Neighborhood details listing

If you were to click the *toltec* 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 6-4 summarizes the server configuration options introduced previously. All three of these options are global in scope, so they must appear in the `[global]` section of the configuration file.

Table 6-4. Server configuration options

Option	Parameters	Function	Default	Scope
`netbios name`	string	NetBIOS name of the Samba server	Server's unqualified DNS hostname	Global
`workgroup`	string	NetBIOS group to which the server belongs	Defined at compile time	Global
`server string`	string	Descriptive string for the Samba server	`Samba %v`	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 fully qualified domain name. For example, a system 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 system'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 outlined in Chapter 1.

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 system 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 system. 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 server that's taking over for it. Users won't have to change their drive mappings to a different server; new connections to `SALES` will simply go to the new server.

workgroup

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

The default option for this parameter is set at compile time to `WORKGROUP`. Because this is the default workgroup name of every unconfigured Windows and Samba system, we recommend that you always set your workgroup name in the Samba configuration file. When choosing your workgroup name, try to avoid making it the same name as a server or user. This will avoid possible problems with WINS name resolution.

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 view) and the comment entry of the Microsoft Windows printer manager.[2]

You can use 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
```

Disk Share Configuration

We mentioned in the previous section that there were no disk shares on the `toltec` server. Let's continue

building the configuration file and create an empty disk share called `[data]`. Here are the additions that will do it:

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

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

Samba's default is to create a read-only share. As a result, the `writable` option needs to be explicitly set for each disk share you wish to make writable.

We will also need to create the */export/samba/data* directory on the Samba server with the following commands:

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

Now, if we connect to the `toltec` server again by double-clicking its icon in the Windows Network Neighborhood, we will see a single share entitled `data`, as shown in Figure 6-4. This share has read/write access, so files can be copied to or from it.

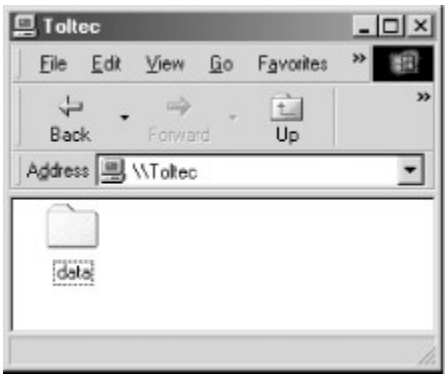

Figure 6-4. The initial data share on the Samba server

Disk Share Configuration Options

The basic Samba configuration options for disk shares previously introduced are listed in Table 6-5.

Table 6-5. Basic share configuration options

Option	Parameters	Function	Default	Scope
`path (directory)`	string (directory name)	Sets the Unix directory that will be provided for a disk share or used for spooling by a printer share.	`/tmp`	Share
`comment`	string	Sets the comment that appears with the share.	None	Share
`volume`	string	Sets the MS-DOS volume name for the share.	Share name	Share

`read only`	boolean	If `yes`, allows read-only access to a share.	`yes`	Share
`writable` (`write ok` or `writeable`)	boolean	If `no`, allows read-only access to a share. If `yes`, both reading and writing are allowed.	`no`	Share

path

This option, which has the synonym `directory`, indicates the pathname for the root of the shared directory or printer. You can choose any directory 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 contents of the directory on the Samba server.

The directory specified as the value for `path` can be given as a relative path, in which case it will be relative to the directory specified by the `root directory` parameter. Because `root directory` defaults to root (*/*), it is generally a good idea to use absolute paths for the `path` parameter, unless `root directory` has been set to something other than the default.

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 using the Details view on the share folder or with the *net view* command at an MS-DOS prompt. For example, here is how you might insert a comment for a share:

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

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.

volume

This option allows you to specify the volume name of the share, which would otherwise default to the name of the share given in the *smb.conf* file.

Some software installation programs check the volume name of the distribution CD-ROM to make sure the correct CD-ROM is in the drive before attempting to install from 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 make sure the installation program sees the correct volume name:

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

read only, writable

The options `read only` and `writable` (also called `writeable` or `write ok`) are really two ways of saying the same thing, but they are 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
writable = 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
writable = yes
```

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 multihomed system (on multiple subnets), you will need to configure Samba to use all the network interfaces. Another use for the options presented in this section is to implement better security by allowing or disallowing connections on the specified interfaces.

Let's assume that our Samba server can access both the subnets 192.168.220.* and 134.213.233.*. Here are our additions to the configuration file to add the networking configuration options:

```
[global]
    #  Networking configuration options
    hosts allow = 192.168.220. 134.213.233.
    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
```

Take a look at 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. We could use the *hosts.allow* and *hosts.deny* files, but we are using this method instead because there might be services on the server that we want others to access without also giving them access to Samba's disk or printer shares.

With the `hosts allow` option, we've specified a 192.168.220 IP address, which 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 192.168.220.102 will be denied even though it is still in the subnet matched by the `hosts allow` option. It is important to understand how Samba sorts out the rules specified by `hosts allow` and `hosts deny` :

1. If no `allow` or `deny` options are defined anywhere in *smb.conf*, Samba will allow connections from any system.

2. If `hosts allow` or `hosts deny` options are defined in the `[global]` section of *smb.conf*, they will apply to all shares, even if either option is defined in one or more of the shares.

3. If only a `hosts allow` option is defined for a share, only the hosts listed will be allowed to use the share. All others will be denied.

4. If only a `hosts deny` option is defined for a share, any client 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) to access the share. Otherwise, the host will not be allowed.

   WARNING

   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 to gain access to a Samba share.

The other two options that we've specified are `interfaces` and `bind interface only`. 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 system 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 on the subnets specified with the `interfaces` option. This is different from the `hosts allow` and `hosts deny` options, which prevent clients from making connections to services, but not from receiving broadcast messages. Using the `bind interfaces only` option is a way to shut out all datagrams from foreign subnets. 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 earlier are summarized in Table 6-6.

Table 6-6. Networking configuration options

Option	Parameters	Function	Default	Scope
`hosts allow` (`allow hosts`)	string (list of hostnames)	Client systems that can connect to Samba.	None	Share
`hosts deny` (`deny hosts`)	string (list of hostnames)	Client systems that cannot connect to Samba.	None	Share
`interfaces`	string (list of IP/netmask	Network interfaces Samba will respond to. Allows correcting defaults.	System-dependent	Global

	combinations)			
`bind interfaces only`	boolean	If set to `yes`, Samba will bind only to those interfaces specified by the `interfaces` option.	`no`	Global

hosts allow

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

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

- Hostnames, such as `ftp.example.com` .
- IP addresses, such as `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 systems within the *ora.com* domain.
- Netgroups, which start with an at sign (`@`), such as `@printerhosts`. Netgroups are usually available only on systems running NIS or NIS+. 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 systems whose IP addresses begin with 130.63.9.
- The keyword `ALL`, which allows any client access.
- The keyword `EXCEPT` followed by 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 to include the trailing dot).

Using the `ALL` keyword by itself is almost always a bad idea because it means that crackers on any network can access your Samba server.

The hostname `localhost`, for the loopback address 127.0.0.1, is included in the `hosts allow` list by default and does not need to be listed explicitly unless you have specified the `bind interfaces only` parameter. This address is required for Samba to work properly.

Other than that, there is no default value for the `hosts allow` configuration option. The default course of action in the event that neither the `hosts allow` or `hosts deny` option is specified in *smb.conf* is to allow access from all sources.

> **TIP**
>
> If you specify `hosts allow` in the `[global]` section, that definition will override any `hosts allow` lines in the share definitions. This is the opposite of the usual behavior, which is for parameters set in share definitions to override default values set in the `[global]` section.

hosts deny

The `hosts deny` option (synonymous with `deny hosts`) specifies client systems that do not have permission to access a share, written as a comma- or space-separated list of hostnames or their IP addresses. Use the same format for specifying clients as the `hosts allow` option earlier. For example, to restrict access to the server from everywhere but `example.com`, you could write:

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

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 access to specific shares, omit both the `hosts allow` and `hosts deny` options from the `[global]` section of the configuration file.

NOTE

Never include the loopback address (`localhost` at IP address 127.0.0.1) in the `hosts deny` list. The *smbpasswd* program needs to connect through the loopback address to the Samba server as a client to change a user's encrypted password. If the loopback address is disabled, the locally generated packets requesting the change of the encrypted password will be discarded by Samba.

In addition, both local browsing propagation and some functions of SWAT require access to the Samba server through the loopback address and will not work correctly if this address is disabled.

interfaces

The `interfaces` option specifies the networks that you want the Samba server to recognize and respond to. 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, as in 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, like this:

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

The number after the slash specifies the number of bits that will be set in the netmask. For example, the number 24 means that the first 24 (of 32) bits will be set in the bitmask, which is the same as specifying 255.255.255.0 as the netmask. Likewise, 16 would be equivalent to a netmask of 255.255.0.0, and 8 would be the same as a netmask of 255.0.0.0.

WARNING

This option might 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 respond only to 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 Samba to ignore any packets (including broadcast packets) whose source address does not correspond to any of the network interfaces specified by 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 be used only by experts.

> **TIP**
>
> If you set `bind interfaces only` to `yes` , add the local host 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, local browse list propagation will fail, and some functions of swat will not work properly.

Virtual Servers

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

For example, the accounting department might have an `accounting` server, and clients of it would see just the accounting disks and printers. The marketing department could have its own server, `marketing`, with its own reports, and so on. However, all the services would be provided by one medium-size Unix server (and one relaxed administrator) instead of having one small server per department.

Virtual Server Configuration Options

Samba will allow a server to use more than one NetBIOS name with the `netbios aliases` option. See Table 6-7.

Table 6-7. Virtual server configuration options

Option	Parameters	Function	Default	Scope
`netbios aliases`	string (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 Windows clients. When a connection is requested to any of the servers, 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 larger and faster disks and are retiring or reallocating the old Windows NT/2000 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 6-5 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 to which it's trying to connect, which is made available in the configuration file through the `%L` variable. If the requested server is `sales`, Samba will include the file */usr/local/samba/lib/smb.conf.sales*. 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

[sales2003]
    path = /usr/local/samba/sales/sales2003/
...
```

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.

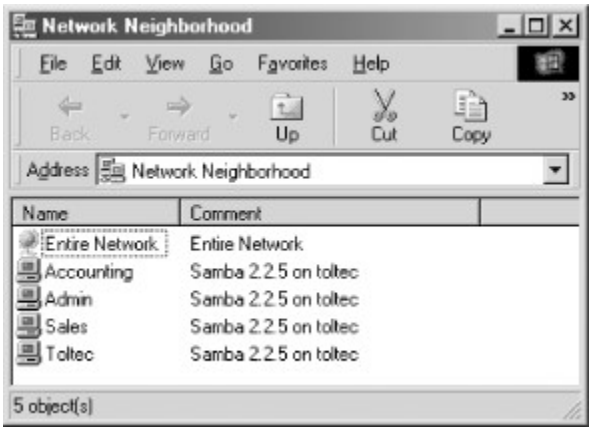

Figure 6-5. Using NetBIOS aliases for a Samba server

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:

```
[2002/07/21 13:23:25, 3] smbd/service.c:close_cnum(514)
  maya (172.16.1.6) closed connection to service IPC$
[2002/07/21 13:23:25, 3] smbd/connection.c:yield_connection(40)
  Yielding connection to IPC$
```

```
[2002/07/21 13:23:25, 3] smbd/process.c:process_smb(615)
  Transaction 923 of length 49
[2002/07/21 13:23:25, 3] smbd/process.c:switch_message(448)
  switch message SMBread (pid 467)
[2002/07/21 13:23:25, 3] lib/doscalls.c:dos_ChDir(336)
  dos_ChDir to /home/samba
[2002/07/21 13:23:25, 3] smbd/reply.c:reply_read(2199)
  read fnum=4207 num=2820 nread=2820
[2002/07/21 13:23:25, 3] smbd/process.c:process_smb(615)
  Transaction 924 of length 55
[2002/07/21 13:23:25, 3] smbd/process.c:switch_message(448)
  switch message SMBreadbraw (pid 467)
[2002/07/21 13:23:25, 3] smbd/reply.c:reply_readbraw(2053)
  readbraw fnum=4207 start=130820 max=1276 min=0 nread=1276
[2002/07/21 13:23:25, 3] smbd/process.c:process_smb(615)
  Transaction 925 of length 55
[2002/07/21 13:23:25, 3] smbd/process.c:switch_message(448)
  switch message SMBreadbraw (pid 467)
```

Much of this information is of use only to Samba programmers. However, we will go over the meaning of some of these entries in more detail in Chapter 12.

Samba contains six options that allow users to describe how and where logging information should be written. Each of these are global options and cannot appear inside a share definition. Here is an example of some logging options that we are adding to our configuration file:

```
[global]
    log level = 2
    log file = /var/log/samba.log.%m
    max log size = 50
    debug timestamp = 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. Levels 2 or 3 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 working on the Samba source code.

The logging 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 client system or user.

We've added a precaution to the log files: no one log file can exceed 50 KB 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 the Samba daemons.

We have decided to write the timestamps of the messages in the logs with the `debug timestamp` option, which is the default behavior. This will place a timestamp in each message written to 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. See Appendix E for more information about the `--with-syslog` option.

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 set a value for the `syslog` parameter in your Samba configuration file to specify which logging messages are to be sent to syslog. Only messages that have debug levels lower than the value of the `syslog` parameter will be sent to syslog. For example, setting the following:

```
syslog = 3
```

specifies that any logging messages with a level of 2 or below will be sent to both syslog and the Samba logging files. (The mappings to *syslog* priorities are described in the upcoming section "syslog.") To continue the example, let's assume that we have set the `log level` option to 4. Logging messages with levels of 2 and 1 will be sent to both syslog and the Samba logging files, and messages with a level of 3 or 4 will be sent to the Samba logging files, but not to syslog. If the `syslog` value exceeds the `log level` value, nothing will be sent to 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, as with the `log level` option.

Logging Configuration Options

Table 6-8 lists each logging configuration option that Samba can use.

Table 6-8. Logging configuration options

Option	Parameters	Function	Default	Scope
`log file`	string (name of file)	Name of the log file that Samba is to use. Works with all variables.	Specified in Samba makefile	Global
`log level` (`debug level`)	numeric (0-10)	Amount of log/debug messages that are sent to the log file. 0 is none; 3 is considerable.	`1`	Global
`max log size`	numeric (size in KB)	Maximum size of log file.	`5000`	Global
`debug timestamp` (`timestamp logs`)	boolean	If `no`, doesn't timestamp logs, making them easier to read during heavy debugging.	`yes`	Global

`syslog`	numeric (0-10)	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 Samba log files.	`no`	Global

log file

By default, Samba writes log information to text files in the */usr/local/samba/var* directory. The `log file` option can be used to set the name of the log file to another location. For example, to put the Samba log information in */usr/local/logs/samba.log*, you could use the following:

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

You can 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 set to 0 or 1. However, if you have a specific problem, you might 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 it slows down the server considerably. Therefore, we recommend that for normal day-to-day operation, you avoid setting this option to anything above 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 a *.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 1MB, Samba renames the log file *samba.log*. *machine-name.old*, and a new log file is generated. If there is already a file 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 quietly eat away at your available disk space. Using this option protects unwary administrators from suddenly discovering that most of the space on a disk or partition 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 timestamp information within the Samba log lines gets in the way, you can turn it off by giving either the `timestamp logs` or the synonymous `debug timestamp` option a value of `no`. For example, a regular Samba log file presents its output in the

following form:

```
12/31/01 12:03:34 toltec (172.16.1.1) connect to server network as user jay
```

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

```
toltec (172.16.1.1) connect to server network as user jay
```

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 a numeric value. 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. Debug logs greater than or equal to the `syslog` level, but less than log level, will still be sent to the standard Samba log files. 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. All messages sent to the system logger are mapped to a priority level that the syslogd daemon understands, as shown in Table 6-9. The default level is 1.

Table 6-9. 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 Section 6.8.1, earlier in this chapter.)

syslog only

The `syslog only` option tells Samba not to use its own logging files at all and to use only the system logger. To enable this, specify the following option in the global section of the Samba configuration file:

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

Footnotes

[1] Depending on your system, this file might not be */etc/printcap*. You can use the *testparm* command that comes with Samba to dump the parameter definitions and determine the value of

the `printcap name` configuration option. The value assigned to it is the default value chosen when Samba was configured and compiled, which should be correct.

[2] We are referring here to the window that opens when a printer icon in the Printers control panel is double-clicked.

TOC