

CHAPTER 3

Basic Samba Configuration

All major Linux distributions ship with Samba, the Server Message Block/Common Internet File System (SMB/CIFS) server for Unix-like systems. This server package enables Linux to serve files and printers to Windows clients, providing a reliable and low-cost platform to fill this role. In fact, despite some fundamental differences between the Linux/Unix and Windows platforms, Samba handles its duties so well that Samba servers are often more trouble-free than their Windows counterparts, so network administrators have sometimes gone to great lengths to deploy Linux running Samba rather than Windows in this role.

This chapter describes basic Samba configuration, starting with installing the server. Other topics include the configuration file format, how you identify the server to other computers on the network, minimal options to help Samba get along with other systems in terms of its browsing features, and setting password options. You must set these basic features before you can move on to the next topic, configuring file and printer shares; that topic is covered in Chapter 4.



An experienced Samba administrator who's familiar with the local network can set all the options described in this chapter in just a minute or two. Many of these options require some time to fully describe because of changes in SMB/CIFS over time and because of peculiarities of integrating SMB/CIFS with Linux's traditional networking tools, but you'll change only a handful of Samba configuration file options. If you're impatient to get started, pay particular attention to the sections "Workgroup Name Options" and "Setting Password Options."

Installing Samba

Samba isn't a single server; rather, it's a family of servers that together provide the full functionality of the package. (Nonetheless, references to "the Samba server" or similar phrases are common.) Four daemons provide the most important Samba features.

smbd

This daemon handles the file- and printer-serving functions per se. Clients connect to it using TCP port 139 or 445 to request the transfer of files.

nmbd

This daemon handles most of the SMB/CIFS functionality not provided by *smbd*, including NetBIOS name resolution (as described in the section “Identifying the Server”) and browsing features (as described briefly in the section “Setting Master Browser Options” and in more detail in Chapter 5). If you run *smbd*, chances are you’ll also run *nmbd*. This server binds to UDP ports 137 and 138.

SWAT

The Samba Web Administration Tool (SWAT) provides a web-based GUI administration tool for Samba. Running it isn’t necessary, and I don’t describe it further in this book. It can be a handy tool for new Samba administrators, though, and it provides some functions that can help ordinary users, such as an interface to change their passwords. It usually runs on TCP port 901.

Winbind

This daemon, which is also known as *winbindd*, provides a way for Linux to access NetBIOS name and Windows NT domain information. The main upshot is that a system that runs Winbind can authenticate its local users against the Windows domain’s user database, as described in Chapter 7. Although Winbind is a daemon, it isn’t a server for other computers; it enables extra functionality solely for the computer on which it runs.

In addition to these daemons, Samba provides a number of support utilities and client programs. These include the *smbclient* client program, which provides FTP-like access to SMB/CIFS shares; the *smbmount* utilities, which helps you mount SMB/CIFS shares in Linux; and the *smbpasswd* utility for handling Samba passwords. Some of these tools are described in this chapter, but others are covered elsewhere in this book.

Most Linux distributions deliver these programs in one or more packages. Typically, a base package is called *samba* or *samba-common*. Additional functionality often ships in other packages, such as *samba-clients* or *swat*. Consult your distribution’s package list and descriptions to learn what you need to install for the functionality you require. Alternatively, you can download and install Samba from its own web site, <http://www.samba.org>. This site’s download area provides links to binaries for many distributions and to a source code tarball that should compile on any Linux distribution. (Just one source tarball contains all the major Samba components described here.)

Samba (or at least the *smbd* and *nmbd* daemons) is typically launched through SysV startup scripts, and these usually install from the distribution’s main Samba package. If you installed Samba from a source tarball, though, you’ll need to create your own SysV startup script, run Samba from a local startup script, or launch Samba

manually on an as-needed basis. (The *packaging* subdirectory of the Samba source package includes sample SysV startup scripts for several distributions.) Although it's possible to run Samba from a super server such as *inetd* or *xinetd*, doing so is uncommon and isn't recommended. In fact, *nmbd* tends to be a bit difficult to run in this way.

A few features related to SMB/CIFS aren't part of the main Samba package. Most notably, the ability to mount SMB/CIFS shares on a Linux system is built into the Linux kernel, although it relies on the external *smbmount* command, which is part of the Samba package. Some GUI SMB/CIFS network browsers are also available separately. Many of these tools nonetheless rely on the basic Samba configuration described in this chapter for certain default values.

The Samba Configuration File Format

Before delving into Samba configuration, you should understand the Samba configuration file format. This file is called *smb.conf*, and it's typically located in */etc/samba*, although a few distributions (particularly old ones) place it in some variant location, such as */etc/samba.d* or */etc*. When you compile from source code, it goes in */usr/local/samba/lib* unless you change a configuration option.

Wherever it's located, the *smb.conf* file is broken into several distinct sections, each of which has its purposes. Within each section, lines have a simple structure consisting of a *parameter* that's to be set and one or more *values* to be assigned to the parameter, or they may be comment lines. You should also understand the use of Samba *variables*, which enable you to set a parameter to a value you may not know when creating the configuration file.

Configuration File Sections

Example 3-1 shows a short but complete *smb.conf* configuration file. In this file, the section names appear between square brackets ([]). In this example, the section names are [global], [homes], and [freefiles].

Example 3-1. A short smb.conf file

```
[global]
workgroup = GREENHOUSE
netbios name = MANDRAKE
server string = Free files for all
encrypt passwords = Yes
security = User
os level = 2
domain master = No
preferred master = No
domain logons = No
```

Example 3-1. A short smb.conf file (continued)

```
[homes]
  browseable = No
  writeable = Yes

# Put all our public files in a logical place....
[freefiles]
  path = /usr/share/samba/public
  browseable = Yes
  writeable = No
```

The [global] section of *smb.conf* is the only section that's really required. It sets *global-level parameters* that affect the operation of the server as a whole, such as setting its NetBIOS name and password encryption settings. In addition, you can place most *share-level parameters* in the [global] section, in which case the parameter effectively changes the default behavior. For instance, the *writeable* parameter is share-level, meaning that you can set it differently for each share. If placed in the [global] section, though, this parameter sets the default for the rest of the shares. This can be handy if you have many shares that use similar options; rather than set the same parameter in all the shares, you can set it just once, in the [global] section.

Sections after the [global] section—the [homes] and [freefiles] sections in Example 3-1—all define individual Samba shares. Each share definition begins with its name and ends with the next share definition or the end of the file. All the parameters in a share definition must be share-level parameters.

Frequently, the share names are not indented, while parameters belonging to a share are indented. This practice makes it easy to locate the parameters you want to adjust, but it's not required; Samba ignores most whitespace in *smb.conf*, including indentation of configuration lines.

Parameters, Values, and Comments

If you examine Example 3-1, you'll quickly discern the basic form of an *smb.conf* parameter line:

```
parameter = Value
```

The *parameter* is a keyword that holds particular meaning to Samba. Some Samba functions can be accessed through multiple parameter names; for instance, *writeable* is synonymous with *writable* and *write ok*, and *read only* is an antonym for these. In other words, *writeable = Yes* has the same effect as *read only = No*.

The *Value* is the value that's assigned to the parameter. Several different types of values exist:

Boolean values

Many Samba parameters require Boolean options. For these, *Yes*, *True*, and *1* are all synonymous, while *No*, *False*, and *0* are their opposites. A few Booleans also

accept other options to set a feature automatically or have some other parameter-specific effect.

Numeric values

Some parameters take numeric values, such as a time in seconds or a file size in bytes or kilobytes. Both integral and real numeric values are possible, although some parameters expect one type or the other. Some parameters take values that are special numbers or sets of numbers, such as IP addresses.

String values

You can provide strings to some parameters, such as the values of the `workgroup`, `netbios name`, `server string`, and `path` parameters in Example 3-1. Sometimes these strings can be almost anything you like, as in `server string`. Other strings must be constrained in some way, though; for instance, `path` is a local Linux pathname. When a string value contains spaces, you do not normally need to enclose it in quotes, although you can do so if you prefer. Quotes may also be necessary with lists of string items that contain spaces.

Delimited values

Some parameters accept a limited range of strings as values. For instance, Example 3-1 shows the security parameter, which accepts just a handful of values.

Lists

Many parameters accept multiple values as options, such as several IP addresses or hostnames. Lists are normally delimited by commas or spaces, although a few parameters use other characters as delimiters.

For the most part, Samba doesn't care about the case of its parameters or values; `domain master = No` has the same effect as `DOMAIN MASTER = no` or any other variant. Some values, though, are case-sensitive for reasons other than Samba. For instance, a Linux filename provided as a value is case-sensitive because the underlying Linux filesystem is case-sensitive.

Similarly, parameters aren't sensitive to whitespace; you can insert or remove spaces from parameters without causing problems. For instance, `server string = Free files for all` is identical to `serverstring = Free files for all`. Whitespace may be important to parameters' values, though.

If a configuration line is very long, you can break it across multiple lines by ending the first line (and any subsequent nonterminal lines) with a backslash (`\`):

```
hosts allow = daisy.greenhouse.example.com, 172.24.21.27, \  
             192.168.7.107
```

This example sets the `hosts allow` parameter to three values—a hostname and two IP addresses.

Instead of or in addition to a parameter and value, an *smb.conf* line may hold a comment. These are denoted by a hash mark (`#`) or a semicolon (`;`); Samba ignores lines that begin with one of these characters. (Whitespace before comments is ignored.)

Many sample *smb.conf* files contain numerous comments describing the function of each configuration line in the file.



Samba provides a parameter called *comment*. This is not to be confused with a comment! The *comment* parameter sets a free-form string that's associated with a share for the benefit of users.

Variables and Their Uses

In most cases, you can set a Samba parameter to a constant value. All the parameters in Example 3-1 do this. Samba also supports variables as parameter values. A *variable* is a placeholder, denoted by a leading percent symbol (%), that can take on a particular value depending upon the machine on which Samba is running, the Samba version, the username of the person accessing the share, and so on. Table 3-1 summarizes Samba's variables. Note that variable identifiers are case-sensitive; for instance, %d and %D are distinct variables.

Table 3-1. Samba variables

Variable	Meaning
%a	The client's OS. Possible values are OS2 (OS/2), Samba, UNKNOWN, wfwg (DOS or Windows for Workgroups), win2K (Windows 2000), win95 (Windows 9x/Me), or winNT (Windows NT).
%c	A print job's length in pages, if known.
%d	The daemon's process ID number.
%D	The client's workgroup or NT domain name, if known.
%f	The sender of a WinPopUp message.
%g	The primary group of %u.
%G	The primary group of %U.
%h	The server's DNS hostname, if known.
%H	The home directory of %u
%I	The client's IP address.
%J	A print job's name.
%L	The server's NetBIOS name.
%m	The client's NetBIOS name, if known.
%M	The client's DNS hostname, if known.
%N	The NIS home directory server.
%p	The path to an automounted share's root directory.
%P	The path to the share's root directory.
%R	The level of the SMB protocol in use. Legal values are CORE, COREPLUS, LANMAN1, LANMAN2, and NT1.
%s	A filename. In printer shares, this identifies the file passed by the client to be printed. It can also refer to a file that holds a WinPopUp message.
%S	The share's name.

Table 3-1. Samba variables (continued)

Variable	Meaning
%t	A WinPopUp message's destination.
%T	The current date and time.
%u	The effective Linux username. This may not be the same as %U.
%U	The username sent by the client.
%v	Samba's version number.
%z	A print job's size in bytes.
%%\$(<i>envvar</i>)	The value of the environment variable <i>envvar</i> .

You can use a variable much as you'd use any other value in a parameter. It will be expanded to its full replacement value when Samba needs to do so. You can even combine variables with regular text or with other variables. For instance, consider the following parameter:

```
log file = /var/log/samba/log.%m
```

A line like this is a common sight in the global sections of *smb.conf* files. If the client's NetBIOS name is *DAISY*, Samba logs information on accesses by this client in */var/log/samba/log.daisy*. (Samba usually converts NetBIOS names to lowercase.) If Samba doesn't know the client's NetBIOS name, the IP address is substituted for the NetBIOS name. Separating logfiles in this way can be handy when debugging problems or tracing usage patterns for the server.

Some environment variables aren't guaranteed to be available. For instance, %L is only available if the client uses the NetBIOS over TCP/IP (NBT) method of connecting to the server, using TCP port 139. This variable is meaningless or will return an IP address for a client that uses the newer "raw" SMB/CIFS over on TCP port 445. Similarly, %h and %M work correctly only if your network's DNS server is working correctly. Variables that convert IP addresses to DNS names also require you to set the `hostname lookups = Yes` parameter to work correctly. Some parameters have meaning only in particular contexts; for instance, %S is meaningless when used with global parameters because a share name can apply only to an individual share and not to the system as a whole.

The include Parameter

Normally, a Samba server uses a single *smb.conf* configuration file; however, you can use the `include` parameter to merge in multiple files. This parameter takes a filename as an option. Samba reads the specified file and uses its contents as if they were part of the main *smb.conf* file, at the location of the `include` parameter.

Typically, you pass a variable as part of the filename that you give to `include`. You can use this ability to provide customized configurations for different client

computers, client OSs, users, and so on. For instance, you can set options that adjust the server's delivery of filenames to clients (as described in Chapter 4) based on the client OS:

```
include = /etc/samba/smb-%a.conf
```

You then create files called *smb-Win95.conf*, *smb-Samba.conf*, or other appropriate values, and place OS-specific options in each file. You can place such a call in the [global] section or in a share definition. In fact, you can even place entire share definitions in an included configuration file. This type of configuration can be useful when one OS works better with one set of options than another. For instance, you might want to set different case-sensitivity options depending on the client OS's capabilities.

Identifying the Server

The first task you must undertake when configuring a Samba server is setting various identification options. SMB/CIFS was designed for non-TCP/IP networks and includes server identification tools that are independent of common TCP/IP naming systems, such as DNS hostnames. SMB/CIFS machines are identified by NetBIOS names, and computers belong to workgroups or NT domains (an NT domain is simply a workgroup with some extra features). Although most recent SMB/CIFS clients can contact servers using DNS hostnames or raw IP addresses rather than NetBIOS names, you must give your Samba server a NetBIOS name and a workgroup (or NT domain) name for interaction with older clients, such as DOS and Windows 9x systems. You may also want to adjust a few additional identification options, which tell the system what operating system to pretend to be, among other things.

NetBIOS Name Options

A NetBIOS name is similar to a computer's DNS hostname (without the domain name component). It's a string of up to 15 characters that can contain letters, numbers, and various punctuation marks. (Using punctuation can be confusing, though, and so is usually best avoided.) NetBIOS names are case-insensitive, although I generally present them in all-uppercase in this book to distinguish them from DNS hostnames, which I present in lowercase.



Technically, the NetBIOS name as just described is only the *base* of the NetBIOS name. The full NetBIOS name includes a one-byte code that identifies the type of service available under the name; for instance, a NetBIOS name might end with a hexadecimal 0x20 to signify a file or print service. A single computer is likely to register several NetBIOS names using a single NetBIOS base name and different type codes. Samba handles this automatically; you just give it the NetBIOS base name, and it registers the names required based on other *smb.conf* options.

You set your computer's NetBIOS name with the global `netbios name` parameter:

```
netbios name = MANDRAKE
```

If you don't include this parameter in the *smb.conf* file, the default is to use your computer's DNS hostname, minus the domain component. For instance, if your computer is called *mandrake.greenhouse.example.com*, Samba registers the NetBIOS name *MANDRAKE*. This default is usually reasonable, assuming your DNS hostname is set correctly; however, you may want to set the NetBIOS name explicitly just to be sure. (When you do so, this setting overrides the DNS hostname for NetBIOS purposes but not for other TCP/IP protocols.) In most cases, you shouldn't try to use different NetBIOS and DNS names on a single computer because it will most likely confuse your users.

Occasionally, you may want to give a computer multiple NetBIOS names. Samba supports this option via the global `netbios aliases` parameter, which enables you to specify names to be registered *in addition to* the name provided with `netbios name` (or the DNS hostname, if you omit `netbios name` from your *smb.conf* file). For instance, suppose that *MANDRAKE* should also be known as *MANDRAGORA* and *MANDRAGORIN*. You can do so by using the following line in addition to the `netbios name` line shown earlier:

```
netbios aliases = MANDRAGORA MANDRAGORIN
```

You can use this parameter to give a system multiple NetBIOS names if it also has multiple DNS hostnames. You can also use it to consolidate several servers in one. For instance, if you replace two old file server computers with one new server, you can have the new server appear under both names by assigning one name with `netbios name` and the second with `netbios aliases`. You'll need to define file or printer shares to match those found on both original servers, though.



If you use the `%L` variable as part of a filename in an `include` parameter, you can load different shares depending on which NetBIOS name a client uses to address the server. This can help minimize user confusion should you want to consolidate many servers into one; to users, your single server can look like the two old ones, complete with different shares available under each name. Be aware, though, that many newer clients, including Windows 2000 and XP, no longer use NetBIOS names by default, so this trick may not be useful on all networks. Specifying `smb ports = 139` limits Samba to using port 139, and hence NetBIOS and its naming conventions. This forces the desired behavior even with most newer clients.

NetBIOS name resolution can work in any of several ways. The most common methods are broadcast name resolution and a NetBIOS Name Server (NBNS) computer, a.k.a. a Windows Internet Name Service (WINS) system. In broadcast name resolution, a client sends a broadcast that contains the name of the system it wants to contact, and that system responds to the broadcasts. Broadcast name resolution is easy to configure (no special Samba parameters are required), but it doesn't work well in networks with multiple subnets.

If your network includes an NBNS system, you should point Samba at it with the global `wins server` parameter, which requires the IP address (or DNS hostname, if you also set `hostname lookups = Yes`) of the NBNS system:

```
wins server = 172.24.21.1
```

Samba 3.0 and later supports multiple NBNS systems (separated by spaces or commas on the `wins server` line).

Conceptually, you can consider an NBNS system to be much like a DNS server; clients contact it to turn names into IP addresses. Unlike a DNS server, though, an NBNS system requires no explicit configuration to add hostnames to it. Instead, clients contact the NBNS system when they start up and at various times thereafter in order to register their configured names. The `wins server` parameter has the dual effect of telling Samba (or `nmbd`, to be more precise) to register with the NBNS system and to use that system for NBNS lookups, when they're required.

While you're setting the `wins server` option, you should check to be sure that `wins support` is set to `No`. If this value is `Yes`, Samba attempts to operate as an NBNS system. This is likely to cause confusion if your network has an existing NBNS system. Of course, if you really want your computer to take on these duties, you should set `wins support = Yes`, but, in this case, you should omit the `wins server` parameter; Samba knows to refer to itself for this function when it's configured as an NBNS system.

You can tell Samba which name lookup methods to use with the `name resolve order` parameter, which takes an ordered list of one to four values:

lmhosts

This option tells Samba to use an *lmhosts* file, which is conceptually and structurally similar to an */etc/hosts* file: it's a list of IP addresses and associated NetBIOS names, one per line. The file is typically stored in the same directory as *smb.conf*.

host

This option refers to lookups using the computer's normal TCP/IP hostname lookup mechanisms—typically */etc/hosts* and DNS. This lookup method doesn't work for some service types, so you shouldn't rely on it exclusively.

wins

This option refers to NBNS-based lookups; it requires that you set wins server (or wins support = Yes).

bcast

You can have Samba perform broadcast name resolution with this option.

The default name resolution order is *lmhosts host wins bcast*, but you can remove options or change their order by specifying them with *name resolve order*:

```
name resolve order = wins bcast lmhosts
```

This example causes NBNS lookups to be tried first, followed by broadcasts, followed by *lmhosts* lookups. In this example, ordinary TCP/IP hostname lookups are not attempted by Samba.

Workgroup Name Options

The NetBIOS naming system is basically flat; all computers on a network have names in the same namespace, with no hierarchical structure. This contrasts with DNS names, which provide for an arbitrary number of domains and subdomains. NetBIOS avoids name conflicts primarily by restricting the scope of the network; NetBIOS name broadcasts don't normally pass over routers, and NBNS computers typically serve just one organization's computers.

NetBIOS does provide the *illusion* of a two-tiered structure, though, through the use of workgroups and NT domains. On a conceptual level, a workgroup is a collection of computers that are related in some way, such as those in a single department. On a technical level, workgroups are implemented by having members of the workgroup register NetBIOS names based on the workgroup name and using particular service type codes.

In any event, you must tell Samba the name of the workgroup to which it belongs. You do this with the global *workgroup* parameter, which takes a workgroup name as its value. These names follow the same naming rules as NetBIOS machine names, but because the computer's DNS domain name is less likely to be a suitable

substitute, it's not used as the default value. Instead, the default if you omit the workgroup parameter is a compile-time option, but it's usually *WORKGROUP*.

If you fail to set the computer's workgroup correctly, you may not be able to browse to the server from Windows clients, or the server may appear under its own unique workgroup in the clients' browsers. Thus, it's important that you set this option appropriately for your network. If you're configuring a new network, select a workgroup name as you see fit. Perhaps your organization's domain name will work, or maybe a subdomain name will be more appropriate. In some cases, you might even use something unrelated, but to avoid confusion, it's usually best to employ a DNS domain or subdomain name as the workgroup name.



Windows NT domains are just workgroups with a special server, the domain controller, which handles centralized logons and typically some other tasks. If you use a domain configuration, you set the NT domain name using the workgroup parameter.

Miscellaneous Identification Options

In addition to setting the NetBIOS name, workgroup name, and related options, you may need to attend to a few miscellaneous identification parameters. These can affect how other systems interact with your Samba server:

server string

This parameter sets a free-form string that appears along with the NetBIOS name in many operating systems' network browsers. In fact, in Windows XP, this string is more prominent than the NetBIOS name.

protocol

This parameter sets the maximum protocol level that Samba uses. (The %R entry in Table 3-1 describes the values that this parameter accepts.) Chances are you won't need to change this value, but it's conceivable that downgrading will help when dealing with very old clients.

announce as

You can tell Samba to announce itself as any number of different Windows OSs with this parameter. Legal values are NT Server (the default), NT (a synonym for NT Server), NT Workstation, Win95, and WfW. As with `protocol`, chances are you won't need to adjust this parameter except perhaps with some very old clients, which might not be able to cope with newer settings.

announce version

This parameter sets an OS version number that goes along with the `announce as` value. The default value for recent versions of Samba is 4.9, and this should almost never be changed.

Chances are you'll only want to set the server string parameter, which has a direct effect on clients. This is shown in Figure 3-1, which depicts a Windows XP computer's view of the servers on a network using the My Network Places browsing tool. In most cases, the NetBIOS name appears in parentheses after the value of the server string variable. (*HALRLOPRILLALAR* in Figure 3-1 is an exception because it lacks the server string value, or rather, it lacks its equivalent because this computer is the Windows computer used to take the screen shot.) Many default *smb.conf* files place the *%v* variable in the server string parameter, which has the effect of displaying the Samba version number to clients, as in the *TEELA* server in Figure 3-1. This information, though, can be useful to miscreants wanting to break into the computer. To be sure, they can discover the version in some other way, but there's no point in making it easy for them; I recommend not using *%v* in your server string parameter.

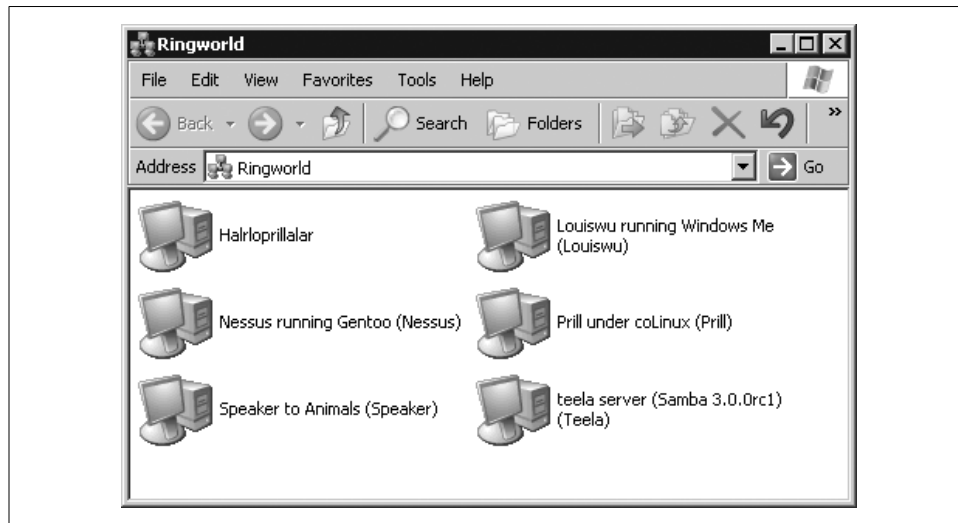


Figure 3-1. Windows displays the value of the server string variable alongside the NetBIOS name in its network browser

Setting Master Browser Options

In order to handle local network browsers like the one shown in Figure 3-1, SMB/CIFS requires one computer to be designated a *master browser*. This computer collects data on the computers on the network and provides it to any computer that asks for the information. The clients then present the data to users in one form or another (Figure 3-1 being one example).



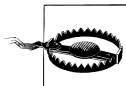
In the context of SMB/CIFS, a network browser is a tool that provides information about, and typically a way to access, SMB/CIFS file and printer shares. Typically, it's integrated into the OS's local file manager. In Windows, it's called either *My Network Places* or *Network Neighborhood*, depending on the version of Windows. SMB/CIFS browsers use different protocols from web browsers. Although some programs, such as the K Desktop Environment's (KDE's) Konqueror, can serve as both SMB/CIFS and web browsers, the two protocols are entirely unrelated.

In fact, two types of SMB/CIFS master browsers exist. A *local master browser* handles browsing tasks on a single subnet. A *domain master browser* helps integrate multiple subnets. The local master browser is selected automatically by the computers on a network using a process known as an *election*. Samba provides options that influence how it participates in elections; you can “rig” an election so that Samba wins or loses it, as you see fit. Domain master browser status is acquired based on server configuration, and Samba provides options to control this process, as well. Chapter 5 describes these parameters in more detail. For the moment, though, if you don't want Samba to acquire either type of master browser status, you should add the following parameters to your *smb.conf* file's [global] section:

```
domain master = No
local master = No
os level = 0
preferred master = No
```

In fact, the first two options should be sufficient to keep Samba from acquiring master browser status; the last two options simply provide added insurance, should you accidentally mis-set the *local master* parameter. Of course, on some networks you might want Samba to acquire local master browser status; to do so, set the following options:

```
local master = Yes
os level = 33
preferred master = Yes
```



Configuring Samba to become a local master browser on a network on which browsing works fine may cause problems. If Samba acquires master browser duties over a working master browser, the result can be disruptions should that old system try to reacquire master browser status periodically. Each such attempt to reacquire master browser status will result in a temporary browsing outage. Likewise, network topology and other issues can cause problems when changing a master browser. If in doubt, configure Samba to *not* try to take on these duties.

Setting Password Options



New Samba installations are frequently plagued by two problems: incorrectly set workgroup names and password encryption difficulties. The first problem is easily corrected by changing the `workgroup` parameter, as described earlier. Password problems are harder to overcome because they may require changing more than just one or two Samba parameters. To address these issues, you must first understand them. You must then decide whether to use unencrypted or encrypted passwords. On some networks, you may need to decide whether to use a password server for authentication, as well.

Password Issues

Samba password issues can be complicated. SMB/CIFS provides several different ways to encode passwords, to authenticate clients using passwords, and to store them. In fact, some of these issues are negotiated between client and server, with no need for explicit configuration, but others require your attention.

The simplest case of Samba password handling, at least from the point of view of Samba administration, is to have Samba accept unencrypted (or *cleartext*) passwords from clients and authenticate users against the local Linux account database. Conceptually, this works much like Linux authentication for FTP, Telnet, SSH, or other servers that use the Linux account database. Unfortunately, this approach has some problems. Most importantly, exchanging passwords in cleartext makes them vulnerable to *sniffing*—interception by unauthorized third parties who have physical access to your network wires. (In an Internet exchange, sniffing can also occur on intervening routers or their networks.) Thus, unencrypted passwords are undesirable from a security point of view. (On the other hand, the password encryption systems used by some versions of SMB/CIFS are not much better than cleartext, so you shouldn't consider encrypted passwords to be proof against sniffing.) In terms of practicality, cleartext passwords are also a problem because recent versions of Windows use encrypted passwords by default and don't drop back to cleartext passwords. Although you can reconfigure Windows clients to use cleartext passwords, doing so on a large network can be tedious.

So, what about encrypted passwords? Unfortunately, the password encryption systems used by SMB/CIFS aren't compatible with the encrypted form of Linux passwords used in a standard Linux password database (*/etc/passwd* or, more commonly, */etc/shadow*). Therefore, in order to support SMB/CIFS encrypted passwords, Samba must maintain its own password database. Typically, this database is stored in a file called *smbpasswd* and is located in the same directory as *smb.conf* or a subdirectory of that directory. Other methods of storing this database exist but are beyond the scope of this book. If you want to use encrypted passwords, you must not only configure Samba to use them but create the encrypted password file, populate it with account information,



and assign passwords to users. Because the Linux passwords are stored in a *hash* (basically, a one-way encryption system), they can't be decrypted, and you'll need to either assign random Samba passwords to users or have them enter passwords in some way. This task can be tedious on a large network.

A third approach to handling passwords is to defer to another computer. For instance, if your network is configured as a Windows NT domain or an AD domain, you can have Samba defer to the domain controller. This approach greatly simplifies Samba setup because you don't need to configure a local password database. Samba provides several options for how to defer to a remote system.

No matter what method you use, each user of your system must have a local account. (Using guest accounts can relax this restriction, but this topic is beyond the scope of this book.) Thus, you must still create local Linux accounts even if you use a Windows domain controller for authentication. If this task is tedious because you have many users, you may want to consult Chapter 7, which describes joining a Linux system to an NT domain in a way that enables the underlying Linux accounts to mirror the NT domain's accounts. Although this configuration can be a bit tricky to set up, it can greatly simplify account maintenance on a large network that has an NT domain controller (either a Windows system or a Samba server).



Using Cleartext Passwords



From a Samba configuration perspective, the simplest authentication method is to use cleartext passwords. You can do so by setting `encrypt passwords = No` in the `[global]` section of `smb.conf`. This configuration is the default in Samba versions prior to 3.0; however, with Version 3.0, the default setting changed to `Yes`. To avoid confusion, I recommend setting the value explicitly, whatever version of Samba you're using. When configured to use cleartext passwords, Samba doesn't attempt to negotiate an encrypted password exchange with clients; it does attempt to authenticate users against the passwords stored in the local Linux password database. Thus, users must have valid local Linux passwords, not just valid accounts. (With encrypted passwords, Samba users' accounts could conceivably exist but have disabled local passwords.)

Windows versions since Windows 95 OEM Service Release 2 (OSR2) and Windows NT 4.0 Service Pack 3 (SP3) require the use of encrypted passwords by default. Thus, these OSs will not work with a Samba server configured to use cleartext passwords unless you change a Windows Registry entry. One relatively painless way to do so is to use a `.reg` file that ships with Samba. In fact, several such files exist, one for each version of Windows. The filename takes the form `WinVer_PlanPassword.reg`, where `Ver` is the Windows version. For instance, `Win2000_PlanPassword.reg` is the file for Windows 2000, and `WinXP_PlanPassword.reg` does the job for Windows XP. Some distributions deliver these files in compressed form, so `.gz` may be tacked onto the end; if so, you'll need to uncompress the file with `gunzip` before you use it. Precisely

where you can find these files also varies. Most place them in the *Registry* subdirectory of the Samba documentation directory, as in */usr/share/doc/samba-3.0.2a/full_docs/Registry*, but the precise path varies.



Try using your distribution's package management tools to locate these files. For instance, on a computer that uses the RPM Package Manager (RPM), you could type `rpm -ql samba | grep PlainPassword` to locate the files in the *samba* package that contain the string *PlainPassword* in their names.

Once you've located these files, copy the ones you need to a floppy disk, put them on an FTP site, send them via email, or otherwise make them accessible to clients. On a Windows system, double-click the file from the file manager to install the changes in the Registry. You then need to reboot the computer for the changes to take effect. In a small office, you should be able to apply the patch to all the Windows clients in a few minutes by walking from one system to another with a floppy disk in your hand. Alternatively, you can make the changes manually using a Windows Registry editor; however, applying the changes automatically is almost certain to be both faster and more reliable.

Once you've configured the Samba server and your clients to use cleartext passwords, the clients should be able to access the server, assuming appropriate accounts with valid passwords exist on the server. However, a few additional parameters can affect access:

password level

Linux's local passwords are case-sensitive, but many SMB/CIFS clients assume passwords will be treated in a case-insensitive way. For instance, Windows 9x/Me converts all passwords to uppercase when using certain SMB/CIFS protocol levels. In order to work around this problem, the global `password level` parameter tells Samba to try case variants. The default value for this parameter is 0, which causes Samba to try the password as delivered and the password converted to all-lowercase if it was sent in all-uppercase. Higher values cause Samba to convert the password to lowercase and then to convert the specified number of letters to uppercase. For instance, if `password level = 1` and if a client gives RHUMBA as the password, Samba tries to authenticate the user with passwords of RHUMBA, rhumba, Rhumba, rHumba, and so on. Using high numbers as the `password level` parameter can therefore improve the odds of a successful login using a valid password that's been corrupted by the client. These attempts increase the time for Samba to confirm that a password is invalid, though, and, in some cases, to verify a valid password. They also increase the odds of a successful break-in by effectively eliminating case as a security feature in your local passwords.

username level

This global parameter is similar to `password level`, but it applies to usernames rather than passwords. One other minor difference is that when this parameter is set to its default value of 0, Samba tests the username converted to lowercase followed by the username converted to lowercase but with an initial capital letter. If you give higher values, Samba tries up to the number of letters converted to uppercase that you specify, just as with `password level`.

username

This share-level parameter specifies a list of usernames against which to test a password. This is necessary when using some very old clients (such as some antiquated DOS clients) that don't send usernames, just passwords. Samba tries the password with each of the usernames specified. Ordinarily, this parameter isn't needed because all modern clients deliver usernames by default. This parameter can be used with encrypted passwords as well as with cleartext ones, but because the clients most likely to force its use employ unencrypted passwords, I've described it here.

On the whole, using cleartext passwords is normally undesirable on modern networks. Between the increased risks of password sniffing with cleartext passwords, and the fact that most modern SMB/CIFS clients don't use them by default, you're usually better off switching to an encrypted password system or to a password server. Encrypted passwords can be more of a hassle to configure on the Samba server, but they're easier to configure on the clients.

Using Encrypted Passwords

Because all modern versions of Windows use encrypted passwords by default, this approach is the easiest one from a client configuration point of view. You will, though, need to take some steps to get encrypted passwords working on the server.



From a client perspective, the difference between using encrypted passwords on the file or print server and using a password server is nil. In both cases, the client engages in a challenge-response authentication exchange with the file or print server; only the configuration of that file or print server differs.

Only one `smb.conf` entry needs changing to use encrypted passwords: `encrypt passwords = Yes`. This is the default value for Samba 3.0 and later, but earlier versions used `No` as the default, so I recommend setting it explicitly to avoid confusion. A few other parameters can influence how Samba treats encrypted passwords, but they probably don't need adjustment:

smb passwd file

You can tell Samba what file to use for holding encrypted passwords with this global parameter. Ordinarily, Samba uses a file called *smbpasswd*, which is usually located in the same directory as *smb.conf* or a subdirectory thereof.

passwd backend

This parameter tells Samba how to store its password database. The usual value, *smbpasswd*, specifies that the *smbpasswd* file (or another file specified by *smb passwd file*) be used. Other options tell Samba to use more specialized types of databases, such as an LDAP directory. Fully describing these alternatives is beyond the scope of this book.

lanman auth

The LANMAN hashing scheme is one of several SMB/CIFS encrypted authentication systems. This global Boolean parameter enables or disables support for this protocol. The default value is *Yes*, and the parameter must be set to this value to support Windows 9x/Me systems.

ntlm auth

The NT LANMAN (NTLM) hash is an improvement on LANMAN authentication, and this parameter controls whether Samba accepts this authentication method. The default value is *Yes*. If this option and *lanman auth* are both set to *No*, only the newest NTLMv2 protocol will work; however, this protocol was added only to Windows NT 4.0 SP4; older clients (and many non-Microsoft clients) don't support NTLMv2.

In addition to setting *encrypt passwords = Yes* and, if desired, any ancillary password-related parameters, you must prepare a local Samba password database. This database is maintained using the *smbpasswd* command. In particular, you add users to the database using the *-a* parameter (which can only be used as *root*):

```
# smbpasswd -a linnaeus
New SMB password:
Retype new SMB password:
Added user linnaeus
```

This command adds a Samba password entry for the user *linnaeus*. If your system lacks a current *smbpasswd* file or its equivalent, you'll see an error message to the effect that it doesn't exist; but don't fear, the *smbpasswd* utility creates the file and adds the user to it. In any event, you should repeat this command for every user on your system. Note that Samba requires the username to match an existing Linux account, so if you're configuring a new system, you should create a Linux account first, and then create a Samba password database entry for it.

Running *smbpasswd* in this way isn't very difficult for a network with just a few users, but on a larger network, it can be quite tedious. If you want to script the operation, you can deliver a password to the utility within a script by appending the password to the username, separated by a percent symbol (%):

```
smbpasswd -a linnaeus%apassword
```

Of course, your script will need to generate passwords in some (preferably random) way. You'll then need to either communicate this information to your users or help them enter their passwords later. Another option is to use a script called *mksmbpasswd*, *mksmbpasswd.sh*, or something similar. These scripts create a new *smbpasswd* file from your regular Linux *passwd* database. Such scripts used to ship with Samba packages, but they're less common today, perhaps because they save very little time. The scripts can't convert Linux passwords to a form that SMB/CIFS can use, so they deliberately generate accounts with invalid passwords. Thus, you must still help users enter their encrypted passwords manually.

In theory, the global update *encrypted* parameter can help you enter encrypted passwords. When set to Yes, this parameter causes Samba to set a user's encrypted password to the value of an unencrypted password that a client computer sends when the user logs on. Unfortunately, this requires you to configure your clients to send unencrypted passwords. Thus, although *update encrypted* might help you convert a network from cleartext to encrypted passwords, it won't be of much help when adding a new Linux system to an existing network that already uses encrypted passwords.

The case options described earlier, in the section "Using Cleartext Passwords," are inapplicable to encrypted passwords. Depending on the hash chosen by the client, passwords may be case-sensitive or -insensitive, and Samba provides the same case sensitivity as clients. Thus, in this respect encrypted passwords are simpler than unencrypted passwords.

The *smbpasswd* command can be used to change passwords for existing accounts, as well as create new ones. Type ***smbpasswd username***, where *username* is the username whose password you want to change, to do the job. Individual users can also use this utility to change their passwords, but they must have shell access to the server to do so.



If users don't need shell access to the server, you can set their login shells to */usr/bin/smbpasswd*. When users log in using Telnet or SSH, they'll enter their Linux passwords and then be prompted to change their Samba passwords. Once this is done, they're immediately logged out.

Using a Password Server

Instead of using a local password database, you can defer authentication to another computer—typically a domain controller, but perhaps some other system. In fact, Samba provides three different ways to do this. You choose the method using the *security* parameter, but depending upon the method you choose, you may need to perform some additional configuration steps.

All these methods of authentication require you to set *encrypt passwords = Yes*. Instead of maintaining the account database locally, though, you point Samba at an external server.

Setting the security mode

The security parameter tells Samba what security mode to use—that is, what rules to apply for authenticating users. This parameter takes one of five possible values:

Share

When using this security level, Samba attempts to emulate the default access control method of Windows 9x/Me, which is to assign passwords to individual shares and not use usernames. To do this, Samba tries to authenticate using the password that the client sends and a series of different accounts, such as an account used by a previous logon from the client or the name of the share itself. Share-level security is a poor match to Linux's underlying security model, though, and so it's seldom used.

User

This security model is the default, and it corresponds to the use of a local account database—either a cleartext Linux account database or an encrypted Samba account database, depending on the value of `encrypt passwords`.

Server

When using server-level security, Samba authenticates users against a remote server in much the way that Windows 9x/Me servers do. On a technical level, this authentication method is similar to a man-in-the-middle attack; the Samba server essentially passes the data on from the client as if it were making the logon request, then honors the reply from the server. This approach is easy to configure but occasionally doesn't work correctly. It can be used to authenticate against a server that's not a domain controller, but the remote server must be configured to enable remote authentication. This option is being deprecated in Samba 3.0 and later; it still works, but is likely to eventually vanish.

Domain

In a domain-level configuration, the Samba server fully joins an NT domain, much as Windows NT/200x/XP systems do. Samba can then exchange credentials with the domain controller and use the full NT domain authentication system for its users.

ADS

This is the latest authentication method. It links Samba to a Windows 200x Active Directory (AD) controller and uses native AD protocols for authentication. This system is also the most difficult to configure and isn't fully described in this book.

If you use server-, domain-, or ADS-level security, you must tell Samba where to find the password server. This task can be accomplished with the `global password server` parameter, which accepts a list of one or more names or IP addresses. If you specify a name, it's looked up in the order specified by the `name resolve order` parameter. If you use domain- or ADS-level security, the remote servers must be domain

controllers. Alternatively, you can specify an asterisk (*) to have Samba attempt to locate its domain controller automatically.



Don't use the password server parameter or domain- or ADS-level security on a system that you configure as a domain controller. Such systems should use user-level security and should omit the password server parameter.

Using server-level security

Server-level security can be a quick way to use a remote password server. This configuration requires you to enter options like this:

```
security = Server  
password server = 172.24.21.98
```

Of course, you'd adjust the IP address for your own network. Little else is required for this configuration, at least on the Samba server that users access directly. You must ensure that appropriate user accounts exist on the password server system, though. Those accounts must also match the local Linux user accounts on the Samba server you're configuring; using a remote password server doesn't obviate the need to provide local Linux accounts for Samba's use.



Windows networks frequently employ longer usernames than do Linux systems; for instance, *Carl Linnaeus* rather than *linnaeus*. If your password server uses such usernames, you can map them to conventional Linux usernames with the `username map` parameter. This parameter accepts a filename that contains mappings of Linux to Windows usernames, as in `linnaeus = "Carl Linnaeus"`. When Samba receives a logon request from *Carl Linnaeus*, it authenticates against the password server using that name but uses the local *linnaeus* account. Although you can use this parameter with user-level security, it's most frequently employed with server-, domain-, or ADS-level security.

Using domain-level security

A more complex configuration than server-level security looks nearly identical to it in `smb.conf`:

```
security = Domain  
password server = 172.24.21.98
```

However, this configuration requires joining the Samba server to the domain using the `net` command. You can accomplish this task by passing the `join` subcommand to `net`:

```
# net join member -U adminuser
```

In this example, the system is joined as a member of the domain specified by the `workgroup` parameter in `smb.conf` and controlled by the domain controller pointed to

by password server. You must specify an account to use on the domain controller for this operation with the `-U` parameter. This account must have administrative access to the domain controller's account database, because it must add a *machine trust account* for your Samba server. This machine trust account is used in the authentication process for individual user logons.

As a practical matter, domain-level security is a bit tougher to configure than is server-level security, but it's more reliable in some situations. If necessary, you can use the `username map` parameter, as described in the section "Using server-level security," to associate Linux usernames with Windows usernames.



Chapter 5 describes configuring a Samba server as a domain controller, including the domain controller configuration options required to enable joining other Samba servers as full domain members.

Summary

If you've been making changes to your Samba server's configuration as you've read this chapter, it should now be functioning in a rather minimal way on your network. You should understand the basics of the Samba configuration file format, and you should be able to make your server appear in clients' SMB/CIFS browsers. If at least one share is defined (as is common in sample *smb.conf* files), you should also be able to log on to the Samba server from clients, thanks to appropriate settings for the password options on both client and server. Of course, these tasks aren't enough; in most cases, you run a Samba server in order to share files or printers, which means you need to be able to define appropriate shares. This task is the topic of Chapter 4.