5

Browsing and Advanced Disk Shares

This chapter continues our discussion of disk shares from the previous chapter. Here, we will discuss various differences between the Windows and Unix filesystems—and how Samba works to bridge the gap. There are a surprising number of inconsistencies between a DOS filesystem and a Unix filesystem. In addition, we will talk briefly about name mangling, file locking, and a relatively new feature for Samba: opportunistic locking, or oplocks. However, before we move into that territory, we should first discuss the somewhat arcane topic of browsing with Samba.

Browsing

Browsing is the ability to examine the servers and shares that are currently available on your network. On a Windows NT 4.0 or 95/98 client, a user can browse network servers through the Network Neighborhood folder. By double-clicking the icon representing the server, the user should be able to see the printer and disk share resources available on that machine as well. (If you have Windows NT 3.*x*, you can use the Disk-Connect Network Drive menu in the File Manager to display the available shares on a server.)

From the Windows command line, you can also use the `net view` option to see which servers are currently on the network. Here is an example of the `net view` command in action:

```
C:\>net view
Servers available in workgroup SIMPLE
Server name            Remark
-------------------------------------------------------------
\\CHIMAERA             Windows NT 4.0
\\HYDRA                Samba 2.0.4 on (hydra)
\\PHOENIX              Windows 98
```

Preventing Browsing

You can restrict a share from being in a browse list by using the `browseable` option. This boolean option prevents a share from being seen in the Network Neighborhood at all. For example, to prevent the `[data]` share from the previous chapter from being visible, we could write:

```
[data]
    path = /home/samba/data
    browseable = no
    guest ok = yes
    comment = Data Drive
    volume = Sample-Data-Drive
    writeable = yes
```

Although you typically don't want to do this to an ordinary disk share, the browseable option is useful in the event that you need to create a share with contents that you do not want others to see, such as a `[netlogin]` share for storing logon scripts for Windows domain control (see Chapter 6, *Users, Security, and Domains* for more information on logon scripts).

Another example is the `[homes]` share. This share is often marked non-browsable so that a share named `[homes]` won't appear when its machine's resources are browsed. However, if a user `alice` logs on and looks at the machine's shares, an `[alice]` share will appear under the machine. What if we wanted to make sure `alice`'s share appeared to everyone before she logs in? This could be done with the global `auto services` option. This option preloads shares into the browse list to ensure that they are always visible:

```
[global]
    ...
    auto services = alice
    ...
```

Default Services

In the event that a user cannot successfully connect to a share, you can specify a default share to which they can connect. Since you do not know who will default to this share at any time, you will probably want to set the `guest ok` option to `yes` for this share. Specifying a `default service` can be useful when sending the utterly befuddled to a directory of help files. For example:

```
[global]
    ...
    default service = helpshare
    ...

[helpshare]
    path = /home/samba/helpshare/%S
    browseable = yes
```

```
    guest ok = yes
    comment = Default Share for Unsuccessful Connections
    volume = Sample-Data-Drive
    writeable = no
```

Note that we used the `%S` variable in the `path` option. If you use the `%S` variable, it will refer to the requested nonexistent share (the original share requested by the user), not the name of the resulting default share. This allows us to create different paths with the names of each server, which can provide more customized help files for users. In addition, any underscores (_) specified in the requested share will be converted to slashes (/) when the `%S` variable is used.

Browsing Elections

As mentioned in Chapter 1, *Learning the Samba*, one machine in each subnet always keeps a list of the currently active machines. This list is called the *browse list* and the server that maintains it is called the *local master browser*. As machines come on and off the network, the local master browser continually updates the information in the browse list and provides it to any machine that requests it.

A computer becomes a local master browser by holding a browsing election on the local subnet. Browsing elections can be called at any time. Samba can rig a browsing election for a variety of outcomes, including always becoming the local master browser of the subnet or never becoming it. For example, the following options, which we've added to the configuration file from Chapter 4, *Disk Shares*, will ensure that Samba always wins the election for local master browser no matter which machines are also present:

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

    #  Browsing election options
    os level = 34
    local master = yes

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

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

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

However, what if we didn't always want to win the election? What if we wanted to yield browsing to a Windows NT Server if present? In order to do that, we need to learn how browsing elections work. As you already know, each machine that takes place in the election must broadcast information about itself. This information includes the following:

- The version of the election protocol used
- The operating system on the machine
- The amount of time the client has been on the network
- The hostname of the client

Here is how the election is decided. Operating systems are assigned a binary value according to their version, as shown in Table 5-1.

Table 5-1 . Operating System Values in an Election

Operating System	Value
Windows NT Server 4.0	33
Windows NT Server 3.51	32
Windows NT Workstation 4.0	17
Windows NT Workstation 3.51	16
Windows 98	2
Windows 95	1
Windows 3.1 for Workgroups	1

Following that, each computer on the network is assigned a separate value according to its role, as shown in Table 5-2.

Table 5-2. Computer Role Settings in an Election

Role	Value
Primary Domain Controller	128
WINS Client	32
Preferred Master Browser	8
Active Master Browser	4
Standby Browser	2
Active Backup Browser	1

Elections are decided in the following order:

1. The machine with the highest version of the election protocol will win. (So far, this is meaningless, as all Windows clients have version 1 of the election protocol.)
2. The machine with the highest operating system value wins the election.
3. If there is a tie, the machine with the setting of Preferred Master Browser (role 8) wins the election.
4. If there is still a tie, the client who has been online the longest wins the election.
5. And finally, if there is still a tie, the client name that comes first alphabetically wins.
6. The machine that is the "runner-up" can become a backup browser.

As a result, if you want Samba to take the role of a local master browser, but only if there isn't a Windows NT Server (4.0 or 3.51) on the network, you could change the `os level` parameter in the previous example to:

```
os level = 31
```

This will cause Samba to immediately lose the election to a Windows NT 4.0 or Windows NT 3.5 Server, both of which have a higher operating systems level. On the other hand, if you wanted to decide the local master browser on the basis of the network role, such as which machine is the primary domain controller, you could set the `os level` to match the highest type of operating system on the network and let the election protocol fall down to the next level.

How can you can tell if a machine is a local master browser? By using the `nbtstat` command. Place the NetBIOS name of the machine you wish to check after the `-a` option:

```
C:\>nbtstat -a hydra

      NetBIOS Remote Machine Name Table

 Name                           Type         Status
---------------------------------------------------------------
 HYDRA                    <00>  UNIQUE       Registered
 HYDRA                    <03>  UNIQUE       Registered
 HYDRA                    <20>  UNIQUE       Registered
 ..__ _MSBROWSE_ _.       <01>  GROUP        Registered
 SIMPLE                   <00>  GROUP        Registered
 SIMPLE                   <1D>  UNIQUE       Registered
 SIMPLE                   <1E>  GROUP        Registered

 MAC Address = 00-00-00-00-00-00
```

The resource entry that you're looking for is the `.._ _MSBROWSE_ _.<01>`. This indicates that the server is currently acting as the local master browser for the current subnet. In addition, if the machine is a Samba server, you can check the Samba *nmbd* log file for an entry such as:

```
nmbd/nmbd_become_lmb.c:become_local_master_stage2(406)
*****
Samba name server HYDRA is now a local master browser for
workgroup SIMPLE on subnet 192.168.220.100
****
```

Finally, Windows NT servers serving as primary domain controllers contain a sneak that allows them to assume the role of the local master browser in certain conditions; this is called the *preferred master browser* bit. Earlier, we mentioned that Samba could set this bit on itself as well. You can enable it with the `preferred master` option:

```
#  Browsing election options
os level = 33
local master = yes
preferred master = yes
```

If the preferred master bit is set, the machine will force a browsing election at startup. Of course, this is needed only if you set the `os level` option to match the Windows NT machine. We recommend that you don't use this option if another machine also has the role of preferred master, such as an NT server.

Domain Master Browser

In the opening chapter, we mentioned that in order for a Windows workgroup or domain to extend into multiple subnets, one machine would have to take the role of the *domain master browser*. The domain master browser propagates browse lists across each of the subnets in the workgroup. This works because each local master browser periodically synchronizes its browse list with the domain master browser. During this synchronization, the local master browser passes on any server that the domain master browser does not have in its browse list, and vice versa. In a perfect world, each local master browser would eventually have the browse list for the entire domain.

Unlike the local master browser, there is no election to determine which machine assumes the role of the domain master browser. Instead, the administrator has to set it manually. By Microsoft design, however, the domain master browser and the primary domain controller (PDC) both register a resource type of <1B>, so the roles—and the machines—are inseparable.

If you have a Windows NT server on the network acting as a PDC, we recommend that you do not use Samba to become the domain master browser. The reverse is true as well: if Samba is taking on the responsibilities of a PDC, we

recommend making it the domain master browser as well. Although it is possible to split the roles with Samba, this is not a good idea. Using two different machines to serve as the PDC and the domain master browser can cause random errors to occur on a Windows workgroup.

Samba can assume the role of a domain master browser for all subnets in the workgroup with the following option:

```
domain master = yes
```

You can verify that a Samba machine is in fact the domain master browser by checking the *nmbd* log file:

```
nmbd/nmbd_become_dmb.c:become_domain_master_stage2(118)
*****
Samba name server HYDRA is now a domain master browser for
workgroup SIMPLE on subnet 192.168.220.100
*****
```

Or you can use the `nmblookup` command that comes with the Samba distribution to query for a unique <1B> resource type in the workgroup:

```
# nmblookup SIMPLE#1B
Sending queries to 192.168.220.255
192.168.220.100 SIMPLE<1b>
```

Multiple subnets

There are three rules that you must remember when creating a workgroup/domain that spans more than one subnet:

- You must have either a Windows NT or Samba machine acting as a local master browser on each subnet in the workgroup/domain. (If you have a domain master browser in a subnet, a local master browser is not needed.)
- You must have a Windows NT Server or a Samba machine acting as a domain master browser somewhere in the workgroup.
- Each local master browser must be instructed to synchronize with the domain master browser.

Samba has a few other features in this arena in the event that you don't have or want a domain master browser on your network. Consider the subnets shown in Figure 5-1.

First, a Samba server that is a local master browser can use the `remote announce` configuration option to make sure that computers in different subnets are sent broadcast announcements about the server. This has the effect of ensuring that the Samba server appears in the browse lists of foreign subnets. To achieve this, however, the directed broadcasts must reach the local master browser on the other subnet. Be aware that many routers do not allow directed broadcasts by default;

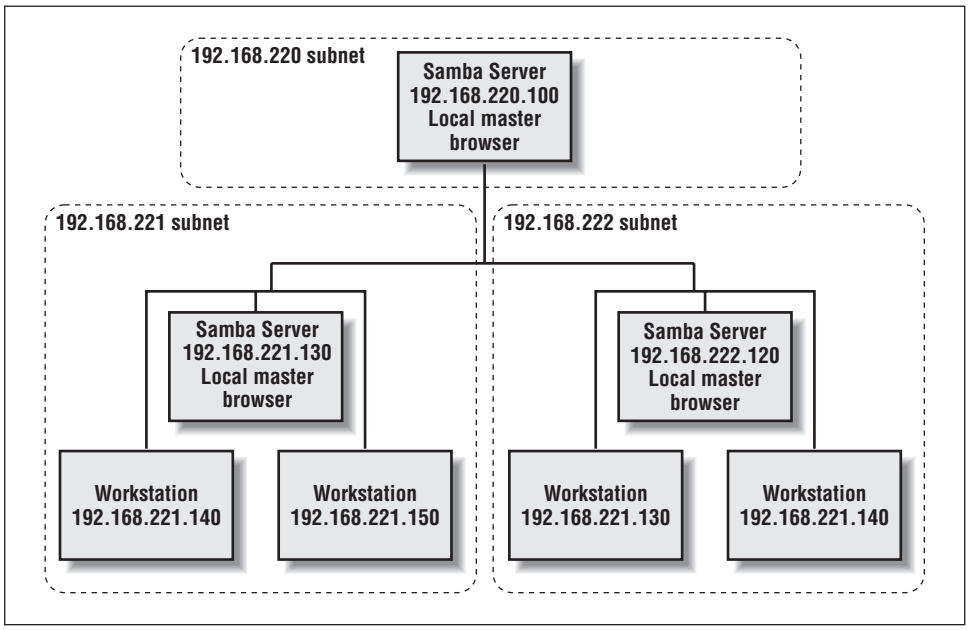

Figure 5-1. Multiple subnets with Samba servers

you may have to change this setting on the router for the directed broadcasts to get through to its subnet.

With the `remote announce` option, list the subnets and the workgroup that should receive the broadcast. For example, to ensure that machines in the 192.168.221 and 192.168.222 subnets and SIMPLE workgroup are sent broadcast information from our Samba server, we could specify the following:

```
#  Browsing election options
os level = 34
local master = yes
remote announce = 192.168.221.255/SIMPLE \
    192.168.222.255/SIMPLE
```

In addition, you are allowed to specify the exact address to send broadcasts to if the local master browser on the foreign subnet is guaranteed to always have a fixed IP address.

A Samba local master browser can synchronize its browse list directly with another Samba server acting as a local master browser on a different subnet. For example, let's assume that Samba is configured as a local master browser, and Samba local master browsers exist at 192.168.221.130 and 192.168.222.120. We can use the `remote browse sync` option to sync directly with the Samba servers, as follows:

```
#  Browsing election options
os level = 34
```

```
local master = yes
remote browse sync = 192.168.221.130 192.168.222.120
```

In order for this to work, the other Samba machines must also be local master browsers. You can also use directed broadcasts with this option if you do not know specific IP addresses of local master browsers.

Browsing Options

Table 5-3 shows 14 options that define how Samba handles browsing tasks. We recommend the defaults for a site that prefers to be easy on its users with respect to locating shares and printers.

Table 5-3. Browsing Configuration Options

Option	Parameters	Function	Default	Scope
`announce as`	`NT` or `Win95` or `WfW`	Sets the operating system that Samba will announce itself as.	`NT`	Global
`announce version`	numerical	Sets the version of the operating system that Samba will announce itself as.	`4.2`	Global
`browseable (browsable)`	boolean	Allows share to be displayed in list of machine resources.	`yes`	Share
`browse list`	boolean	If `yes`, Samba will provide a browse list on this server.	`yes`	Global
`auto services (preload)`	string (share list)	Sets a list of shares that will always appear in the browse list.	None	Global
`default service (default)`	string (share name)	Names a share (service) that will be provided if the client requests a share not listed in *smb.conf*.	None	Global
`local master`	boolean	If `yes`, Samba will try to become a master browser on the local subnet.	`yes`	Global
`lm announce`	`yes` or `no` or `auto`	Enables or disables LAN Manager style host announcements.	`auto`	Global
`lm interval`	numerical	Specifies the frequency in seconds that LAN Manager announcements will be made if activated.	`60`	Global
`preferred master (prefered master)`	boolean	If `yes`, Samba will use the preferred master browser bit to attempt to become the local master browser.	`no`	Global
`domain master`	boolean	If `yes`, Samba will try to become the main browser master for the workgroup.	`no`	Global

Table 5-3. Browsing Configuration Options (continued)

Option	Parameters	Function	Default	Scope
`os level`	numerical	Sets the operating system level of Samba in an election for local master browser.	`0`	Global
`remote browse sync`	string (list of IP addresses)	Lists Samba servers to synchronize browse lists with.	None	Global
`remote announce`	string (IP address/ workgroup pairs)	Lists subnets and workgroups to send directed broadcast packets to, allowing Samba to appear to browse lists.	None	Global

announce as

This global configuration option specifies the type of operating system that Samba will announce to other machines on the network. The default value for this option is `NT`, which represents a Windows NT operating system. Other possible values are `Win95`, which represents a Windows 95 operating system, and `WfW` for a Windows for Workgroup operating system. You can override the default value with the following:

```
[global]
    announce as = Win95
```

We recommend against changing the default value of this configuration option.

announce version

This global option is frequently used with the `announce as` configuration option; it specifies the version of the operating system that Samba will announce to other machines on the network. The default value of this options is 4.2, which places itself above the current Windows NT version of 4.0. You can specify a new value with a global entry such as the following:

```
[global]
    announce version = 4.3
```

We recommend against changing the default value of this configuration option.

browseable

The `browseable` option (also spelled `browsable`) indicates whether the share referenced should appear in the list of available resources of the machine on which it resides. This option is always set to `yes` by default. If you wish to prevent the share from being seen in a client's browser, you can reset this option to `no`.

Note that this does not prevent someone from accessing the share using other means, such as specifying a UNC location (`//server/accounting`) in Windows

Explorer. It only prevents the share from being listed under the machine's resources when being browsed.

browse list

You should never need to change this parameter from its default value of `yes`. If your Samba server is acting as a local master browser (i.e., it has won the browsing election), you can use the global `browse list` option to instruct Samba to provide or withhold its browse list to all clients. By default, Samba always provides a browse list. You can withhold this information by specifying the following:

```
[global]
    browse list = no
```

If you disable the browse list, clients cannot browse the names of other machines, their services, and other domains currently available on the network. Note that this won't make any particular machine inaccessible; if someone knows a valid machine name/address and a share on that machine, they can still connect to it explicitly using NET USE or by mapping a drive letter to it using Windows Explorer. It simply prevents information in the browse list from being retrieved by any client that requests it.

auto services

The global `auto services` option, which is also called `preload`, ensures that the specified shares are always visible in the browse list. One common use for this option is to advertise specific user or printer shares that are created by the `[homes]` or `[printers]` shares, but are not otherwise browsable.

This option works best with disk shares. If you wish to force each of your system printers (i.e., those listed in the printer capabilities file) into the browse list using this option, we recommend using the `load printers` option instead. Any shares listed with the `auto services` option will not be displayed if the `browse list` option is set to `no`.

default service

The global `default service` option (sometimes called `default`) names a "last-ditch" share. If set to an existing share name, and a client requests a nonexistent disk or printer share, Samba will attempt to connect the user to the share specified by this option instead. The option is specified as follows:

```
default service = helpshare
```

Note that there are no braces surrounding the share name `helpshare`, even though the definition of the share later in the Samba configuration file will have

braces. Also, if you use the `%S` variable in the share specified by this option, it will represent the requested, nonexistent share, not the default service. Any underscores (`_`) specified in the request share will be converted to slashes (`/`) when the variable is used.

local master

This global option specifies whether Samba will attempt to become the local master browser for the subnet when it starts up. If this option is set to `yes`, Samba will take place in elections. However, setting this option by itself does not guarantee victory. (Other parameters, such as `preferred master` and `os level` help Samba win browsing elections.) If this option is set to `no`, Samba will lose all browsing elections, no matter which values are specified by the other configuration options. The default value is `yes`.

lm announce

The global `lm announce` option tells Samba's *nmbd* whether or not to send LAN Manager host announcements on behalf of the server. These host announcements may be required by older clients, such as IBM's OS/2 operating system. This announcement allows the server to be added to the browse lists of the client. If activated, Samba will announce itself repetitively at the number of seconds specified by the `lm interval` option.

This configuration option takes the standard boolean values, `yes` and `no`, which engage or disengage LAN Manager announcements, respectively. In addition, there is a third option, `auto`, which causes *nmbd* to passively listen for LAN Manager announcements, but not send any of its own initially. If LAN Manager announcements are detected for another machine on the network, *nmbd* will start sending its own LAN Manager announcements to ensure that it is visible. You can specify the option as follows:

```
[global]
    lm announce = yes
```

The default value is `auto`. You probably won't need to change this value from its default.

lm interval

This option, which is used in conjunction with `lm announce`, indicates the number of seconds *nmbd* will wait before repeatedly broadcasting LAN Manager-style announcements. Remember that LAN Manager announcements must be activated in order for this option to be used. The default value is 60 seconds. If you set this value to 0, Samba will not send any LAN Manager host announcements, no matter

what the value of the `lm announce` option. You can reset the value of this option as follows:

```
[global]
    lm interval = 90
```

preferred master

The `preferred master` option requests that Samba set the preferred master bit when participating in an election. This gives the server a higher preferred status in the workgroup than other machines at the same operating system level. If you are configuring your Samba machine to become the local master browser, it is wise to set the following value:

```
[global]
    preferred master = yes
```

Otherwise, you should leave it set to its default, `no`. If Samba is configured as a preferred master browser, it will force an election when it first comes online.

os level

The global `os level` option dictates the operating system level at which Samba will masquerade during a browser election. If you wish to have Samba win an election and become the master browser, you can set the level above that of the operating system on your network with the highest current value. The values are shown in Table 5-1. The default level is 0, which means that Samba will lose all elections. If you wish Samba to win all elections, you can reset its value as follows:

```
os level = 34
```

This means that the server will vote for itself 34 times each time an election is called, which ensures a victory.

domain master

If Samba is the primary domain controller for your workgroup or NT domain, it should also be the domain master browser. The domain master browser is a special machine that has the NetBIOS resource type <1B> and is used to propagate browse lists to and from each of the local master browsers in individual subnets across the domain. To force Samba to become the domain master browser, set the following in the `[global]` section of the *smb.conf*:

```
[global]
    domain master = yes
```

If you have a Windows NT server on the network acting as a primary domain controller (PDC), we recommend that you do not use Samba to become the domain

master browser. The reverse is true as well: if Samba is taking on the responsibilities of a PDC, we recommend making it the domain master browser. Splitting the PDC and the domain master browser will cause unpredictable errors to occur on the network.

remote browse sync

The global `remote browse sync` option specifies that Samba should synchronize its browse lists with local master browsers in other subnets. However, the synchronization can occur only with other Samba servers, and not with Windows computers. For example, if your Samba server was a master browser on the subnet 192.168.235, and Samba local master browsers existed on other subnets at 192.168.234.92 and 192.168.236.2, you could specify the following:

```
remote browse sync = 192.168.234.92 192.168.236.2
```

The Samba server would then directly contact the other machines on the address list and synchronize browse lists. You can also say:

```
remote browse sync = 192.168.234.255 192.168.236.255
```

This forces Samba to broadcast queries to determine the IP addresses of the local master browser on each subnet, with which it will then synchronize browse lists. This only works, however, if your router doesn't block directed broadcast requests ending in 255.

remote announce

Samba servers are capable of providing browse lists to foreign subnets with the `remote announce` option. This is typically sent to the local master browser of the foreign subnet in question. However, if you do not know the address of the local master browser, you can do the following:

```
[global]
    remote announce = 192.168.234.255/ACCOUNTING \
                      192.168.236.255/ACCOUNTING
```

With this, Samba will broadcast host announcements to all machines on subnets 192.168.234 and 192.168.236, which will hopefully reach the local master browser of the subnet. You can also specify exact IP addresses, if they are known.

Filesystem Differences

One of the biggest issues for which Samba has to correct is the difference between Unix and non-Unix filesystems. This includes items such as handling symbolic links, hidden files, and dot files. In addition, file permissions can also be a headache if not accounted for properly. This section describes how to use Samba to

make up for some of those annoying differences, and even how to add some new functionality of its own.

Hiding and Vetoing Files

There are some cases when we need to ensure that a user cannot see or access a file at all. Other times, we don't want to keep a user from accessing a file—we just want to hide it when they view the contents of the directory. On Windows systems, an attribute of files allows them to be hidden from a folder listing. With Unix, the traditional way of hiding files in a directory is to precede them with a dot (.). This prevents items such as configuration files or defaults from being seen when performing an ordinary `ls` command. Keeping a user from accessing a file at all, however, involves working with permissions on files and or directories.

The first option we should discuss is the boolean `hide dot files`. This option does exactly what it says. When set to `yes`, the option treats files beginning with a period (.) as hidden. If set to `no`, those files are always shown. The important thing to remember is that the files are only hidden. If the user has chosen to show all hidden files while browsing (e.g., using the Folder Options menu item under the View menu in Windows 98), they will still be able to see the files, as shown in Figure 5-2.

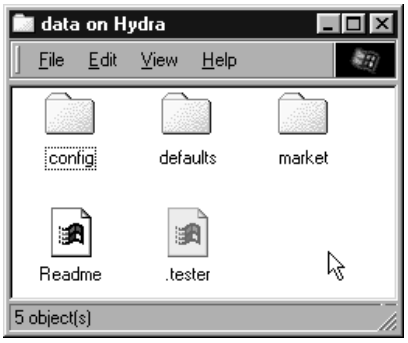

Figure 5-2. Hidden files in the [data] share

Instead of simply hiding files beginning with a dot, you can also specify a string pattern to Samba for files to hide, using the `hide files` option. For example, let's assume that we specified the following in our example `[data]` share:

```
[data]
    path = /home/samba/data
    browseable = yes
```

```
        guest ok = yes
        writeable = yes
        case sensitive = no
        hide files = /*.java/*README*/
```

Each entry for this option must begin, end, or be separated from another with a slash (/) character, even if there is only one pattern listed. This convention allows spaces to appear in filenames. In this example, the share directory would appear as shown in Figure 5-3. Again, note that we have set the Windows 98 option to view hidden files for the window.

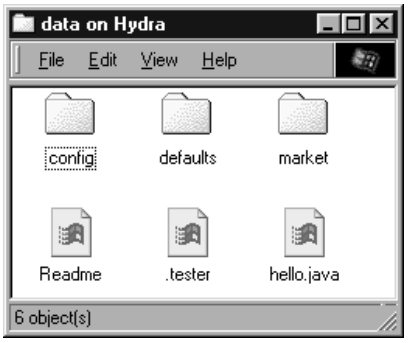

Figure 5-3. Hiding files based on filename patterns

If we want to prevent users from seeing files at all, we can instead use the `veto files` option. This option, which takes the same syntax as the `hide files` option, specifies a list of files that should never be seen by the user. For example, let's change the `[data]` share to the following:

```
[data]
    path = /home/samba/data
    browseable = yes
    guest ok = yes
    writeable = yes
    case sensitive = no
    veto files = /*.java/*README*/
```

The syntax of this option is identical to the `hide files` configuration option: each entry must begin, end, or be separated from another with a slash (/) character, even if there is only one pattern listed. By doing so, the files `hello.java` and `README` will simply disappear from the directory, and the user will not be able to access them through SMB.

There is one other question that we need to address. What happens if the user tries to delete a directory that contains vetoed files? This is where the `delete veto files` option comes in. If this boolean option is set to `yes`, the user is

allowed to delete both the regular files and the vetoed files in the directory, and the directory itself will be removed. If the option is set to `no`, the user will not be able to delete the vetoed files, and consequently the directory will not be deleted either. From the user's perspective, the directory will appear to be empty, but cannot be removed.

The `dont descend` directive specifies a list of directories whose contents Samba should not allow to be visible. Note that we say *contents*, not the directory itself. Users will be able to enter a directory marked as such, but they are prohibited from descending the directory tree any farther—they will always see an empty folder. For example, let's use this option with a more basic form of the share that we defined earlier in the chapter:

```
[data]
    path = /home/samba/data
    browseable = yes
    guest ok = yes
    writeable = yes
    case sensitive = no
    dont descend = config defaults
```

In addition, let's assume that the */home/samba/data* directory has the following contents:

```
drwxr-xr-x   6 tom      users     1024 Jun 13 09:24 .
drwxr-xr-x   8 root     root      1024 Jun 10 17:53 ..
-rw-r--r--   2 tom      users     1024 Jun  9 11:43 README
drwxr-xr-x   3 tom      users     1024 Jun 13 09:28 config
drwxr-xr-x   3 tom      users     1024 Jun 13 09:28 defaults
drwxr-xr-x   3 tom      users     1024 Jun 13 09:28 market
```

If the user then connects to the share, he or she would see the directories shown in Figure 5-4. However, the contents of the */config* and */defaults* directories would appear empty to the user, even if other folders or files existed in them. In addition, users cannot write any data to the folder (which prevents them from creating a file or folder with the same name as one that is already there but invisible). If a user attempts to do so, he or she will receive an "Access Denied" message. `dont descend` is an administrative option, not a security option, and is not a substitute for good file permissions.

Links

DOS and NT filesystems don't have symbolic links; Windows 95/98/NT systems approximate this with "shortcuts" instead. Therefore, when a client tries to open a symbolic link on a Samba server share, Samba attempts to follow the link to find

Figure 5-4. Contents of the [data] share with dont descend

the real file and let the client open it, as if he or she were on a Unix machine. If you don't want to allow this, set the `follow symlinks` option:

```
[data]
    path = /home/samba/data
    browseable = yes
    guest ok = yes
    writeable = yes
    case sensitive = no
    follow symlinks = no
```

You can test this by creating a directory on the Unix server inside the share as the user that you are logging in with. Enter the following commands:

```
% mkdir hello; cd hello
% cat "This is a test" >hello.txt
% ln -s hello.txt "Link to hello"
```

This results in the two files shown in the window in Figure 5-5. Normally, if you click on either one, you will receive a file which has the text "This is a test" inside of it. However, with the `follow symlinks` option set to `no`, you should receive an error similar to the dialog in Figure 5-5 if you click on "Link to hello."

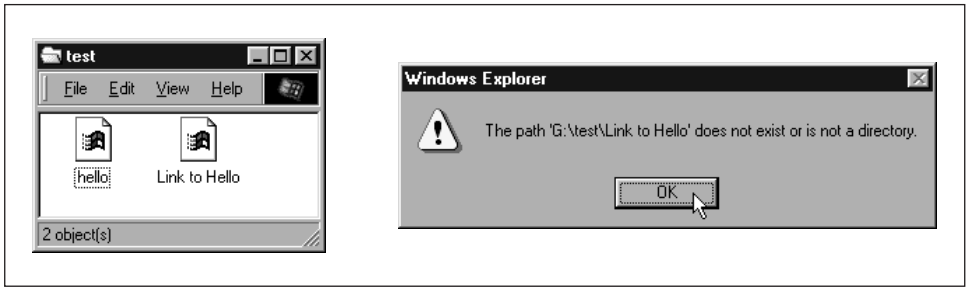

Figure 5-5. An error dialog trying to follow symbolic links when forbidden by Samba

Finally, let's discuss the `wide links` option. This option, if set to `yes`, allows the client user to follow symbolic links that point outside the shared directory tree, including files or directories at the other end of the link. For example, let's assume that we modified the `[data]` share as follows:

```
[data]
    path = /home/samba/data
    browseable = yes
    guest ok = yes
    writeable = yes
    case sensitive = no
    follow symlinks = yes
    wide links = yes
```

As long as the `follow symlinks` option is enabled, this will cause Samba to follow all symbolic links outside the current share tree. If we create a file outside the share (for example, in someone's home directory) and then create a link to it in the share as follows:

```
ln -s ~tom/datafile ./datafile
```

then you will be able to open the file in Tom's directory as per the target file's permissions.

Filesystem Options

Table 5-4 shows a breakdown of the options we discussed earlier. We recommend the defaults for most, except those listed in the following descriptions.

Table 5-4. Filesystem Configuration Options

Option	Parameters	Function	Default	Scope
`unix realname`	boolean	Provides Unix user's full name to client.	`no`	Global
`dont descend`	string (list of directories)	Indicates a list of directories whose contents Samba should make invisible to clients.	None	Share
`follow symlinks`	boolean	If set to `no`, Samba will not honor symbolic links.	`yes`	Share
`getwd cache`	boolean	If set to `yes`, Samba will use a cache for `getwd()` calls.	`yes`	Global
`wide links`	boolean	If set to `yes`, Samba will follow symbolic links outside the share.	`yes`	Share
`hide dot files`	boolean	If set to `yes`, treats Unix hidden files as hidden files in Windows.	`yes`	Share
`hide files`	string (list of files)	List of file patterns to treat as hidden.	None	Share

Table 5-4. Filesystem Configuration Options (continued)

Option	Parameters	Function	Default	Scope
`veto files`	string (list of files)	List of file patterns to never show.	None	Share
`delete veto files`	boolean	If set to `yes`, will delete files matched by `veto files` when the directory they reside in is deleted.	`no`	Share

unix realname

Some programs require a full username in order to operate. For example, a Windows email program often needs to associate a username with a given real name. If your system password file contains the real names of users in the GCOS field, the `unix realname` option instructs Samba to provide this information to clients. Without it, the name of the user will simply be his or her login ID. For example, if your Unix password file contains the following line:

```
rcollins:/KaBfco47Rer5:500:500:Robert Collins:
/home/rcollins:/bin/ksh
```

And the option in the configuration file is:

```
[global]
    unix realname = yes
```

then the name Robert Collins will be provided to any client that requests the real name of user `rcollins`. You typically don't need to bother with this option.

dont descend

The `dont descend` option can be used to specify various directories that should appear empty to the client. Note that the directory itself will still appear. However, Samba will not show any of the contents of the directory to the client user. This is not a good option to use as a security feature (a user could probably find a way around it); it really is meant only as a convenience to keep client users from browsing into directories that might have sensitive files. See our example earlier in this section.

follow symlinks

This option, which is discussed in greater detail earlier, controls whether Samba will follow a symbolic link in the Unix operating system to the target, or if it should return an error to the client user. If the option is set to `yes`, the target of the link will be interpreted as the file.

getwd cache

This global option specifies whether Samba should use a local cache for the Unix `getwd()` (get current working directory) system call. You can override the default value of `yes` as follows:

```
[global]
    getwd cache = no
```

Setting this option to `yes` can significantly increase the time it takes to resolve the working directory, especially if the `wide links` option is set to `no`. You should normally not need to alter this option.

wide links

This option specifies whether the client user can follow symbolic links that point outside the shared directory tree. This includes any files or directories at the other end of the link, as long as the permissions are correct for the user. The default value for this option is `yes`. Note that this option will not be honored if the `follow symlinks` options is set to `no`. Setting this option to `no` slows *smbd* considerably.

hide files

The `hide files` option provides one or more directory or filename patterns to Samba. Any file matching this pattern will be treated as a hidden file from the perspective of the client. Note that this simply means that the DOS hidden attribute is set, which may or may not mean that the user can actually see it while browsing.

Each entry in the list must begin, end, or be separated from another entry with a slash (`/`) character, even if there is only one pattern listed. This allows spaces to appear in the list. Asterisks can be used as a wildcard to represent zero or more characters. Questions marks can be used to represent exactly one character. For example:

```
hide files = /.jav*/README.???/
```

hide dot files

The `hide dot files` option hides any files on the server that begin with a dot (.) character, in order to mimic the functionality behind several shell commands that are present on Unix systems. Like `hide files`, those files that begin with a dot have the DOS hidden attribute set, which doesn't necessarily guarantee that a client cannot view them. The default value for this option is `yes`.

veto files

More stringent than the hidden files state is the state provided by the `veto files` configuration option. Samba won't even admit these files exist. You cannot list or

open them from the client. In reality, this isn't a trustworthy security option. It is actually a mechanism to keep PC programs from deleting special files, such as ones used to store the resource fork of a Macintosh file on a Unix filesystem. If both Windows and Macs are sharing the same files, this can prevent ill-advised power users from removing files the Mac users need.

The syntax of this option is identical to that of the `hide files` configuration option: each entry must begin, end, or be separated from another with a slash (/) character, even if only one pattern is listed. Asterisks can be used as a wildcard to represent zero or more characters. Questions marks can be used to represent exactly one character. For example:

```
veto files = /*config/*default?/
```

This option is primarily administrative—not a substitute for good file permissions.

delete veto files

This option tells Samba to delete vetoed files when a user attempts to delete the directory in which they reside. The default value is `no`. This means if a user tries to delete a directory that contains a vetoed file, the file (and the directory) will not be deleted. Instead, the directory will remain and appear to be empty from the perspective of the user. If set to `yes`, the directory and the vetoed files will be deleted.

File Permissions and Attributes on MS-DOS and Unix

DOS was never intended to be a multiuser, networked operating system. Unix, on the other hand, was designed that way from the start. Consequently, there are inconsistencies and gaps in coverage between the two filesystems that Samba must not only be aware of, but also provide solutions for. One of the biggest gaps is how Unix and DOS handle permissions with files.

Let's take a look at how Unix assigns permissions. All Unix files have read, write, and execute bits for three classifications of users: owner, group, and world. These permissions can be seen at the extreme left-hand side when a `ls -al` command is issued in a Unix directory. For example:

```
-rwxr--r--   1 tom     users   2014 Apr 13 14:11 access.conf
```

Windows, on the other hand, has four principal bits that it uses with any file: read-only, system, hidden, and archive. You can view these bits by right-clicking on the

file and choosing the Properties menu item. You should see a dialog similar to Figure 5-6.*

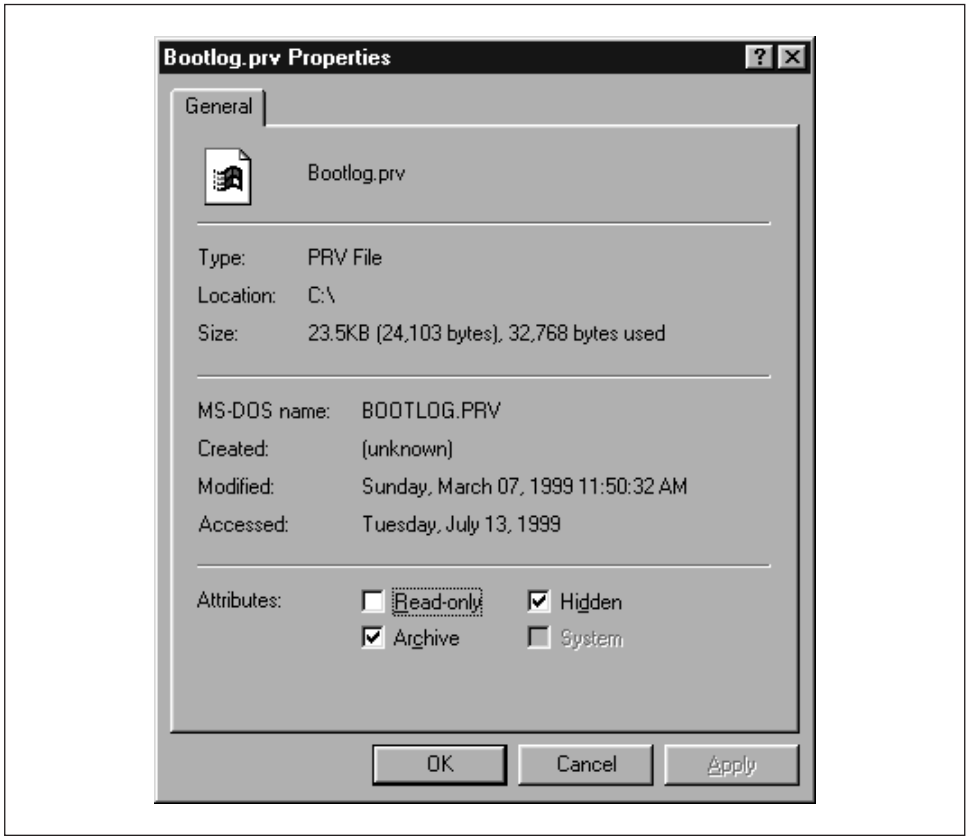

Figure 5-6. DOS and Windows file properties

The definition of each of those bits follows:

Read-only
: The file's contents can be read by a user but cannot be written to.

System
: This file has a specific purpose required by the operating system.

Hidden
: This file has been marked to be invisible to the user, unless the operating systems is explicitly set to show it.

* The system checkbox will probably be greyed for your file. Don't worry about that—you should still be able to see when the box is checked and when it isn't.

Archive
This file has been touched since the last DOS backup was performed on it.

Note that there is no bit to specify that a file is executable. DOS and Windows NT filesystems identify executable files by giving them the extensions .EXE, .COM, .CMD, or .BAT.

Consequently, there is no use for any of the three Unix executable bits that are present on a file in a Samba disk share. DOS files, however, have their own attributes that need to be preserved when they are stored in a Unix environment: the archive, system, and hidden bits. Samba can preserve these bits by reusing the executable permission bits of the file on the Unix side—if it is instructed to do so. Mapping these bits, however, has an unfortunate side-effect: if a Windows user stores a file in a Samba share, and you view it on Unix with the `ls -al` command, some of the executable bits won't mean what you'd expect them to.

Three Samba options decide whether the bits are mapped: `map archive`, `map system`, and `map hidden`. These options map the archive, system, and hidden attributes to the owner, group, and world execute bits of the file, respectively. You can add these options to the `[data]` share, setting each of their values as follows:

```
[data]
    path = /home/samba/data
    browseable = yes
    guest ok = yes
    writeable = yes
    map archive = yes
    map system = yes
    map hidden = yes
```

After that, try creating a file in the share under Unix—such as `hello.java`—and change the permissions of the file to 755. With these Samba options set, you should be able to check the permissions on the Windows side and see that each of the three values has been checked in the Properties dialog box. What about the read-only attribute? By default, Samba 2.0 sets this whenever a file does not have the Unix owner write permission bit set. In other words, you can set this bit by changing the permissions of the file to 555.

We should warn you that the default value of the `map archive` option is `yes`, while the other two options have a default value of `no`. This is because many programs do not work properly if the archive bit is not stored correctly for DOS and Windows files. The system and hidden attributes, however, are not critical for a program's operation and are left to the discretion of the administrator.

Figure 5-7 summarizes the Unix permission bits and illustrates how Samba maps those bits to DOS attributes. Note that the group read/write and world read/write bits do not directly translate to a DOS attribute, but they still retain their original Unix definitions on the Samba server.

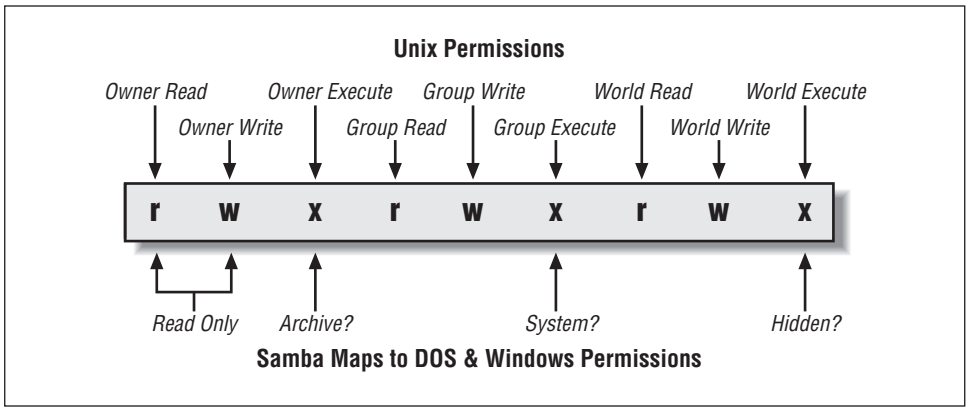

Figure 5-7. How Samba and Unix view the permissions of a file

Creation masks

Samba has several options to help with file creation masks. File creation masks (or *umasks*) help to define the permissions a file or directory will receive at the time it is created. In Unix, this means that you can control what permissions a file or directory does not have when it is created. For files accessed from Windows, this means you can disable the read-only, archive, system, and hidden attributes of a file as well.

For example, the `create mask` option will force the permissions of a file created by a Windows client to be at most 744:

```
[data]
    path = /home/samba/data
    browseable = yes
    guest ok = yes
    writeable = yes
    create mask = 744
```

while the `directory mask` option shown here will force the permissions of a newly created directory to be at most 755:

```
[data]
    path = /home/samba/data
    browseable = yes
    guest ok = yes
    writeable = yes
    directory mask = 755
```

Alternatively, you can also force various bits with the `force create mode` and `force directory mode` options. These options will perform a logical OR against the file and directory creation masks, ensuring that those bits that are specified will always be set. You would typically set these options globally in order to ensure that group and world read/write permissions have been set appropriately for new files or directories in each share.

In the same spirit, if you wish to explicitly set the Unix user and group attributes of a file that is created on the Windows side, you can use the `force user` and `force group` options. For example:

```
[data]
    path = /home/samba/data
    browseable = yes
    guest ok = yes
    writeable = yes

    create mask = 744
    directory mask = 755
    force user = joe
    force group = accounting
```

These options actually assign a static Unix user and group to each connection that is made to a share. However, this occurs *after* the client authenticates; it does not allow free access to a share. These options are frequently used for their side effects of assigning a specific user and group to each new file or directory that is created in a share. Use these options with discretion.

Finally, one of the capabilities of Unix that DOS lacks is the ability to delete a read-only file from a writable directory. In Unix, if a directory is writable, a read-only file in that directory can still be removed. This could permit you to delete files in any of your directories, even if the file was left by someone else.

DOS filesystems are not designed for multiple users, and so its designers decided that read-only means "protected against accidental change, including deletion," rather than "protected against some other user on a single-user machine." So the designers of DOS prohibited removal of a read-only file. Even today, Windows file systems exhibit the same behavior.

Normally, this is harmless. Windows programs don't try to remove read-only files because they know it's a bad idea. However, a number of source-code control programs—which were first written for Unix—run on Windows and require the ability to delete read-only files. Samba permits this behavior with the `delete readonly` option. In order to enable this functionality, set the option to `yes`:

```
[data]
    path = /home/samba/data
    browseable = yes
    guest ok = yes
    writeable = yes

    create mask = 744
    directory mask = 755
    force user = joe
    force group = accounting
    delete readonly = yes
```

File and Directory Permission Options

The options for file and directory permissions are summarized in Table 5-5; each option is then described in detail.

Table 5-5. File and Directory Permission Options

Option	Parameters	Function	Default	Scope
`map archive`	boolean	Preserve DOS archive attribute in user execute bit (0100).	`yes`	Share
`map system`	boolean	Preserve DOS system attribute in group execute bit (0010).	`no`	Share
`map hidden`	boolean	Preserve DOS hidden attribute in world execute bit (0001).	`no`	Share
`create mask (create mode)`	numeric	Sets the maximum permissions for files created by Samba.	`0744`	Share
`directory mask (directory mode)`	numeric	Sets the maximum permissions for directories created by Samba.	`0755`	Share
`force create mode`	numeric	Forces the specified permissions (bitwise or) for directories created by Samba.	`0000`	Share
`force directory mode`	numeric	Forces the specified permissions (bitwise or) for directories created by Samba.	`0000`	Share
`force group (group)`	string (group name)	Sets the effective group for a user accessing this share.	None	Share
`force user`	string (username)	Sets the effective username for a user accessing this share.	None	Share
`delete readonly`	boolean	Allows a user to delete a read-only file from a writable directory.	`no`	Share

create mask

The argument for this option is an octal number indicating which permission flags may be set at file creation by a client in a share. The default is 0755, which means the Unix owner can at most read, write, and optionally execute his or her own files, while members of the user's group and others can only read or execute them. If you need to change it for non-executable files, we recommend 0644, or `rw-r--r--`. Keep in mind that the execute bits may be used by the server to map certain DOS file attributes, as described earlier. If you're altering the create mask, those bits have to be part of the create mask as well.

directory mask

The argument for this option is an octal number indicating which permission flags may be set at directory creation by a client in a share. The default is 0755, which allows everyone on the Unix side to at most read and traverse the directories, but allows only you to modify them. We recommend the mask 0750, removing access by world users.

force create mode

This option sets the permission bits that Samba will force to be set when a file permission change is made. It's often used to force group permissions, mentioned previously. It can also be used to preset any of the DOS attributes we mentioned: archive (0100), system (0010), or hidden (0001). This option always takes effect after the `map archive`, `map system`, `map hidden`, and `create mask` options.

Many Windows applications rename their data files to *datafile.bak* and create new ones, thus changing their ownership and permissions so that members of the same Unix group can't edit them. Setting `force create mask = 0660` will keep the new file editable by members of the group.

force directory mode

This option sets the permission bits which Samba will force when a directory permission change is made or a directory is created. It's often used to force group permissions, as mentioned previously. This option defaults to 0000, and can be used just like the `force create mode` to add group or other permissions if needed. This option always takes effect after the `map archive`, `map system`, `map hidden`, and `directory mask` options.

force group

This option, sometimes called `group`, assigns a static group ID that will be used on all connections to a service after the client has successfully authenticated. This assigns a specific group to each new file or directory created from an SMB client.

force user

The `force user` option assigns a static user ID that will be used on all connections to a service after the client has successfully authenticated. This assigns a specific user to each new file or directory created from an SMB client.

delete readonly

This option allows a user to delete a directory containing a read-only file. By default, DOS and Windows will not allow such an operation. You probably will want to leave this option turned off unless a program needs this capability; many Windows users would be appalled to find that they'd accidentally deleted a file which they had set read-only. In fact, even the Unix `rm` command will ask users if they really want to override the protection and delete read-only files. It's a good idea to have Samba be at least as cautious.

map archive

The DOS archive bit is used to flag a file that has been changed since it was last archived (e.g., backed up with the DOS archive program.) Setting the Samba option `map archive = yes` causes the DOS archive flag to be mapped to the Unix execute-by-owner (0100) bit. It's best to leave this option on if your Windows users are doing their own backups, or are using programs that require the archive bit. Unix lacks the notion of an archive bit entirely. Backup programs typically keep a file that lists what files were backed up on what date, so comparing file modification dates serves the same purpose.

Setting this option to `yes` causes an occasional surprise on Unix when a user notices that a data file is marked as executable, but rarely causes harm. If a user tries to run it, he or she will normally get a string of error messages as the shell tries to execute the first few lines as commands. The reverse is also possible; an executable Unix program looks like it hasn't been backed up recently on Windows. But again, this is rare, and is usually harmless.

map system

The DOS system attribute is used to indicate files that are required by the operating system, and should not be deleted, renamed, or moved without special effort. Set this option only if you need to store Windows system files on the Unix file server. Executable Unix programs will appear to be non-removable special Windows files when viewed from Windows clients. This may prove mildly inconvenient if you want to move or remove one. For most sites, however, this is fairly harmless.

map hidden

DOS uses the hidden attribute to indicate that a file should not ordinarily be visible in directory listings. Unix doesn't have such a facility; it's up to individual programs (notably the shell) to decide what to display and what not to display. Normally, you won't have any DOS files that need to be hidden, so the best thing to do is to leave this option turned off.

Setting this option to `yes` causes the server to map the hidden flag onto the executable-by-others bit (0001). This feature can produce a rather startling effect. Any Unix program that is executable by world seems to vanish when you look for it from a Windows client. If this option is not set, however, and a Windows user attempts to mark a file hidden on a Samba share, it will not work—Samba has no place to store the hidden attribute!

Name Mangling and Case

Back in the days of DOS and Windows 3.1, every filename was limited to eight upper-case characters, followed by a dot, and three more uppercase characters. This was known as the *8.3 format*, and was a huge nuisance. Windows 95/98, Windows NT, and Unix have since relaxed this problem by allowing many more case-sensitive characters to make up a filename. Table 5-6 shows the current naming state of several popular operating systems.

Table 5-6. Operating System Filename Limitations

Operating System	File Naming Rules
DOS 6.22 or below	Eight characters followed by a dot followed by a three-letter extension (8.3 format); case insensitive
Windows 3.1 for Workgroups	Eight characters followed by a dot followed by a three-letter extension (8.3 format); case insensitive
Windows 95/98	127 characters; case sensitive
Windows NT	127 characters; case sensitive
Unix	255 characters; case sensitive

Samba still has to remain backwards compatible with network clients who store files only in the 8.3 format, such as Windows for Workgroups. If a user creates a file on a share called *antidisestablishmentarianism.txt*, a Windows for Workgroups client couldn't tell it apart from another file in the same directory called *antidisease.txt*. Like Windows 95/98 and Windows NT, Samba has to employ a special methodology of translating a long filename to an 8.3 filename in such a way that similar filenames will not cause collisions. This is called *name mangling*, and Samba deals with this in a manner that is similar, but not identical to, Windows 95 and its successors.

The Samba Mangling Operation

Here is how Samba mangles a long filename into an 8.3 filename:

- If the original filename does not begin with a dot, up to the first five alphanumeric characters that occur before the last dot (if there is one) are converted

to uppercase. These characters are used as the first five characters of the 8.3 mangled filename.

- If the original filename begins with a dot, the dot is removed and up to the first five alphanumeric characters that occur before the last dot (if there is one) are converted to uppercase. These characters are used as the first five characters of the 8.3 mangled filename.
- These characters are immediately followed a special mangling character: by default, a tilde (~), although Samba allows you to change this character.
- The base of the long filename before the last period is hashed into a two-character code; parts of the name after the last dot may be used if necessary. This two character code is appended to the 8.3 filename after the mangling character.
- The first three characters after the last dot (if there is one) of the original filename are converted to uppercase and appended onto the mangled name as the extension. If the original filename began with a dot, three underscores (_ _ _) are used as the extension instead.

Here are some examples:

```
virtuosity.dat                      VIRTU~F1.DAT
.htaccess                           HTACC~U0._ _ _
hello.java                          HELLO~1F.JAV
team.config.txt                     TEAMC~04.TXT
antidisestablishmentarianism.txt    ANTID~E3.TXT
antidiseast.txt                     ANTID~9K.TXT
```

Using these rules will allow Windows for Workgroups to differentiate the two files on behalf of the poor individual who is forced to see the network through the eyes of that operating system. Note that the same long filename should always hash to the same mangled name with Samba; this doesn't always happen with Windows. The downside of this approach is that there can still be collisions; however, the chances are greatly reduced.

You generally want to use the mangling configuration options with only the oldest clients. We recommend doing this without disrupting other clients by adding an `include` directive to the *smb.conf* file:

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

This resolves to *smb.conf.WfWg* when a Window for Workgroups client attaches. Now you can create a file */usr/local/samba/lib/smb.conf.WfWg* which might contain these options:

```
[global]
    case sensitive = no
    default case = upper
    preserve case = no
```

```
short preserve case = no
mangle case = yes
mangled names= yes
```

If you are not using Windows for Workgroups 3.1, then you probably do not need to change any of these options from their defaults.

Representing and resolving filenames with Samba

Another item that we should point out is that there is a difference between how an operating system *represents* a file and how it *resolves* it. For example, if you've used Windows 95/98/NT, you have likely run across a file called *README.TXT*. The file can be represented by the operating system entirely in uppercase letters. However, if you open an MS-DOS prompt and enter the command `edit readme.txt`, the all-caps file is loaded into the editing program, even though you typed the name in lowercase letters!

This is because the Windows 95/98/NT family of operating systems resolves files in a case-insensitive manner, even though the files are represented it in a case-sensitive manner. Unix-based operating systems, on the other hand, always resolve files in a case-sensitive manner; if you try to edit *README.TXT* with the command `vi readme.txt`, you will likely be editing the empty buffer of a new file.

Here is how Samba handles case: if the `preserve case` is set to `yes`, Samba will always use the case provided by the operating system for representing (not resolving) filenames. If it is set to `no`, it will use the case specified by the `default case` option. The same is true for `short preserve case`. If this option is set to `yes`, Samba will use the default case of the operating system for representing 8.3 filenames; otherwise it will use the case specified by the `default case` option. Finally, Samba will always resolve filenames in its shares based on the value of the `case sensitive` option.

Mangling Options

Samba allows you to give it more refined instructions on how it should perform name mangling, including those controlling the case sensitivity, the character inserted to form a mangled name, and the ability to manually map filenames from one format to another. These options are shown in Table 5-7.

Table 5-7. Name Mangling Options

Option	Parameters	Function	Default	Scope
`case sensitive (casesignames)`	boolean	If `yes`, Samba will treat filenames as case-sensitive (Windows doesn't).	`no`	Share
`default case`	(`upper` or `lower`)	Case to assume as default (only used when preserve case is `no`).	Lower	Share

Table 5-7. Name Mangling Options (continued)

Option	Parameters	Function	Default	Scope
`preserve case`	boolean	If `yes`, keep the case the client supplied (i.e., do not convert to `default case`).	`yes`	Share
`short preserve case`	boolean	If `yes`, preserve case of 8.3-format names that the client provides.	`yes`	Share
`mangle case`	boolean	Mangle a name if it is mixed case.	`no`	Share
`mangled names`	boolean	Mangles long names into 8.3 DOS format.	`yes`	Share
`mangling char`	string (single character)	Gives mangling character.	`~`	Share
`mangled stack`	numerical	Number of mangled names to keep on the local mangling stack.	`50`	Global
`mangled map`	string (list of patterns)	Allows mapping of filenames from one format into another.	None	Share

case sensitive

This share-level option, which has the obtuse synonym `casesignames`, specifies whether Samba should preserve case when resolving filenames in a specific share. The default value for this option is `no`, which is how Windows handles file resolution. If clients are using an operating system that takes advantage of case-sensitive filenames, you can set this configuration option to `yes` as shown here:

```
[accounting]
    case sensitive = yes
```

Otherwise, we recommend that you leave this option set to its default.

default case

The `default case` option is used with `preserve case`. This specifies the default case (upper or lower) that Samba will use when it creates a file on one of its shares on behalf of a client. The default case is `lower`, which means that newly created files will use the mixed-case names given to them by the client. If you need to, you can override this global option by specifying the following:

```
[global]
    default case = upper
```

If you specify this value, the names of newly created files will be translated into uppercase, and cannot be overridden in a program. We recommend that you use the default value unless you are dealing with a Windows for Workgroups or other 8.3 client, in which case it should be `upper`.

preserve case

This option specifies whether a file created by Samba on behalf of the client is created with the case provided by the client operating system, or the case specified by the `default case` configuration option above. The default value is `yes`, which uses the case provided by the client operating system. If it is set to `no`, the value of the `default case` option is used.

Note that this option does not handle 8.3 file requests sent from the client—see the `short preserve case` option below. You may want to set this option to `yes` if applications that create files on the Samba server are sensitive to the case used when creating the file. If you want to force Samba, for example, to mimic the behavior of a Windows NT filesystem, you can leave this option to its default, `yes`.

short preserve case

This option specifies whether an 8.3 filename created by Samba on behalf of the client is created with the default case of the client operating system, or the case specified by the `default case` configuration option. The default value is `yes`, which uses the case provided by the client operating system. You can let Samba choose the case through the `default case` option by setting it as follows:

```
[global]
    short preserve case = no
```

If you want to force Samba to mimic the behavior of a Windows NT filesystem, you can leave this option set to its default, `yes`.

mangled names

This share-level option specifies whether Samba will mangle filenames for 8.3 clients in that share. If the option is set to `no`, Samba will not mangle the names and (depending on the client), they will either be invisible or appear truncated to those using 8.3 operating systems. The default value is `yes`. You can override it per share as follows:

```
[data]
    mangled names = no
```

mangle case

This option tells Samba whether it should mangle filenames that are not composed entirely of the case specified using the `default case` configuration option. The default for this option is `no`. If you set it to `yes`, you should be sure that all clients will be able to handle the mangled filenames that result. You can override it per share as follows:

```
[data]
    mangle case = yes
```

We recommend that you leave this option alone unless you have a well-justified need to change it.

mangling char

This share-level option specifies the mangling character used when Samba mangles filenames into the 8.3 format. The default character used is a tilde (~). You can reset it to whatever character you wish, for instance:

```
[data]
    mangling char = #
```

mangled stack

Samba maintains a local stack of recently mangled 8.3 filenames; this stack can be used to reverse map mangled filenames back to their original state. This is often needed by applications that create and save a file, close it, and need to modify it later. The default number of long filename/mangled filename pairs stored on this stack is 50. However, if you want to cut down on the amount of processor time used to mangle filenames, you can increase the size of the stack to whatever you wish, at the expense of memory and slightly slower file access.

```
[global]
    mangled stack = 100
```

mangled map

If the default behavior of name mangling is not sufficient, you can give Samba further instructions on how to behave using the `mangled map` option. This option allows you to specify mapping patterns that can be used before or even in place of name mangling performed by Samba. For example:

```
[data]
    mangled map =(*.database *.db) (*.class *.cls)
```

Here, Samba is instructed to search each file it encounters for characters that match the first pattern specified in the parenthesis and convert them to the modified second pattern in the parenthesis for display on an 8.3 client. This is useful in

the event that name mangling converts the filename incorrectly or to a format that the client cannot understand readily. Patterns are separated by whitespaces.

Locks and Oplocks

Concurrent writes to a single file are not desirable in any operating system. To prevent this, most operating systems use *locks* to guarantee that only one process can write to a file at a time. Operating systems traditionally lock entire files, although newer ones allow a range of bytes within a file to be locked. If another process attempts to write to a file (or section of one) that is already locked, it will receive an error from the operating system and will wait until the lock is released.

Samba supports the standard DOS and NT filesystem (deny-mode) locking requests, which allow only one process to write to an entire file on a server at a give time, as well as byte-range locking. In addition, Samba supports a new locking mechanism known in the Windows NT world as *opportunistic locking—oplock* for short.

Opportunistic Locking

Opportunistic locking allows a client to notify the Samba server that it will not only be the exclusive writer of a file, but will also cache its changes to that file on its own machine (and not on the Samba server) in order to speed up file access for that client. When Samba knows that a file has been opportunistically locked by a client, it marks its version as having an opportunistic lock and waits for the client to complete work on the file, at which point it expects the client to send the final changes back to the Samba server for synchronization.

If a second client requests access to that file before the first client has finished working on it, Samba can send an *oplock break* request to the first client. This tells the client to stop caching its changes and return the current state of the file to the server so that the interrupting client can use it as it sees fit. An opportunistic lock, however, is not a replacement for a standard deny-mode lock. It is not unheard of for the interrupting process to be granted an oplock break only to discover that the original process also has a deny-mode lock on a file as well. Figure 5-8 illustrates this opportunistic locking process.

In terms of locks, we highly recommend using the defaults provided by Samba: standard DOS/Windows deny-mode locks for compatibility and oplocks for the extra performance that local caching allows. If your operating system can take advantage of oplocks, it should provide significant performance improvements. Unless you have a specific reason for changing any of these options, it's best to leave them as they are.

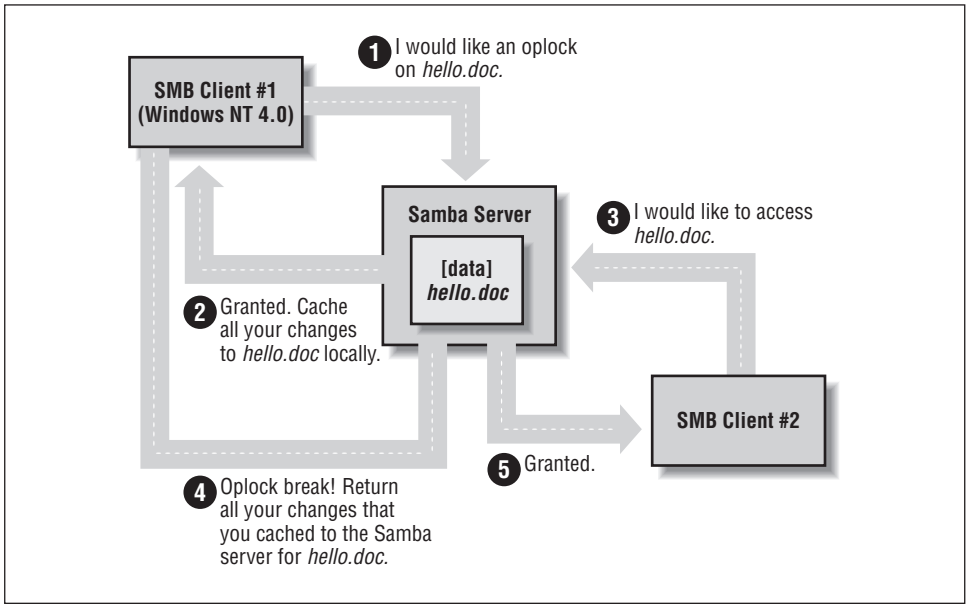

Figure 5-8. Opportunistic locking

Unix and Locking

Windows systems cooperate well to avoid overwriting each other's changes. But if a file stored on a Samba system is accessed by a Unix process, this process won't know a thing about Windows oplocks and could easily ride roughshod over a lock. Some Unix systems have been enhanced to understand the Windows oplocks maintained by Samba. Currently the support exists only in SGI Irix 6.5.2f and later; Linux and FreeBSD should soon follow.

If you have a system that understands oplocks, set `kernel oplocks = yes` in the Samba configuration file. That should eliminate conflicts between Unix processes and Windows users.

If your system does not support kernel oplocks, you could end up with corrupted data when somebody runs a Unix process that reads or writes a file that Windows users also access. However, Samba provides a rough protection mechanism in the absence of kernel oplocks: the `veto oplock files` option. If you can anticipate which Samba files are used by both Windows users and Unix users, set their names in a `veto oplock files` option. This will suppress the use of oplocks on matching filenames, which will supress client caching, and let the Windows and Unix programs use system locking or update times to detect competition for the same file. A sample option is:

```
veto oplock files = /*.dbm/
```

This option allows both Unix processes and Windows users to edit files ending in the suffix *.dbm*. Note that the syntax of this option is similar to `veto files`.

Samba's options for locks and oplocks are given in Table 5-8.

Table 5-8. Locks and Oplocks Configuration Options

Option	Parameters	Function	Default	Scope
`share modes`	boolean	If set to `yes`, turns on support for DOS-style whole-file locks.	`yes`	Share
`locking`	boolean	If `yes`, turns on byte-range locks.	`yes`	Share
`strict locking`	boolean	If `yes`, denies access to an entire file if a byte-range lock exists in it.	`no`	Share
`oplocks`	boolean	If `yes`, turn on local caching of files on the client for this share.	`yes`	Share
`kernel oplocks`	boolean	If `yes`, indicates that the kernel supports oplocks.	`yes`	Global
`fake oplocks`	boolean	If `yes`, tells client the lock was obtained, but doesn't actually lock it.	`no`	Share
`blocking locks`	boolean	Allows lock requestor to wait for the lock to be granted.	`yes`	Share
`veto oplock files`	string (list of filenames)	Does not oplock specified files.	None	Share
`lock directory`	string (fully-qualified pathname)	Sets the location where various Samba files, including locks, are stored.	As specified in Samba makefile	Global

share modes

The most primitive locks available to Samba are deny-mode locks, known as *share modes*, which are employed by programs such as text editors to avoid accidental overwriting of files. For reference, the deny-mode locks are listed in Table 5-9.

Table 5-9. SMB Deny-Mode Locks

Lock	Description
`DENY_NONE`	Do not deny any other file requests.
`DENY_ALL`	Deny all open requests on the current file.
`DENY_READ`	Deny any read-only open requests on the current file.
`DENY_WRITE`	Deny any write-only open requests on the current file.
`DENY_DOS`	If opened for reading, others can read but cannot write to the file. If opened for writing, others cannot open the file at all.
`DENY_FCB`	Obsolete.

The `share modes` parameter, which enforces the use of these locks, is enabled by default. To disable it, use the following command:

```
[accounting]
    share modes = no
```

We highly recommend against disabling the default locking mechanism unless you have a justifiable reason for doing so. Most Windows and DOS applications rely on these locking mechanisms in order to work correctly, and will complain bitterly if this functionality is taken away.

locking

The `locking` option can be used to tell Samba to engage or disengage server-side byte-range locks on behalf of the client. Samba implements byte-range locks on the server side with normal Unix advisory locks and will consequently prevent other properly-behaved Unix processes from overwriting a locked byte range.

This option can be specified per share as follows:

```
[accounting]
    locking = yes
```

If the `locking` option is set to `yes`, the requestor will be delayed until the holder of either type of lock releases it (or crashes). If, however, the option is set to `no`, no byte-range locks will be kept for the files, although requests to lock and unlock files will appear to succeed. The option is set to `yes` by default; however, you can turn this option off if you have read-only media.

strict locking

This option checks every file access for a byte-range lock on the range of bytes being accessed. This is typically not needed if a client adheres to all the locking mechanisms in place. This option is set to `no` by default; however, you can reset it per share as follows:

```
[accounting]
    strict locking = yes
```

If this option is set to `yes`, mandatory locks are enforced on any file with byte-range locks.

blocking locks

Samba also supports *blocking locks*, a minor variant of range locks. Here, if the range of bytes is not available, the client specifies an amount of time that it's willing to wait. The server then caches the lock request, periodically checking to see if the file is available. If it is, it notifies the client; however, if time expires, Samba will tell the client that the request has failed. This strategy prevents the client from continually polling to see if the lock is available.

You can disable this option per share as follows:

```
[accounting]
    blocking locks = no
```

When set to `yes`, blocking locks will be enforced on the file. If this option is set to `no`, Samba behaves as if normal locking mechanisms are in place on the file. The default is `yes`.

oplocks

This option enables or disables support for oplocks on the client. The option is enabled by default. However, you can disable it with the following command:

```
[data]
    oplocks = no
```

If you are in an extremely unstable network environment or have many clients that cannot take advantage of opportunistic locking, it may be better to shut this Samba feature off. Oplocks should be disabled if you are accessing the same files from both Unix applications (such as *vi*) and SMB clients (unless you are lucky enough to have an operating system that supports kernel oplocks as discussed earlier).

fake oplocks

Before opportunistic locking was available on Samba, the Samba daemons pretended to allow oplocks via the `fake oplocks` option. If this option was enabled, all clients were told that the file is available for opportunistic locking, and never warned of simultaneous access. This option is deprecated now that real oplocks are available on Samba.

kernel oplocks

If a Unix application separate from Samba tries to update a file that Samba has oplocked to a Windows client, it will likely succeed (depending on the operating system) and both Samba and the client will never be aware of it. However, if the local Unix operating system supports it, Samba can warn it of oplocked files, which can suspend the Unix process, notify the client via Samba to write its copy back, and only then allow the open to complete. Essentially, this means that the operating system kernel on the Samba system has the ability to handle oplocks as well as Samba.

You can enable this behavior with the `kernel oplocks` option, as follows:

```
[global]
    kernel oplocks = yes
```

Samba can automatically detect kernel oplocks and use them if present. At the time of this writing, this feature is supported only by SGI Irix 6.5.2f and later. However, Linux and FreeBSD support are expected in the near future. A system without kernel oplocks will allow the Unix process to update the file, but the client programs will notice the change only at a later time, if at all.

veto oplock files

You can provide a list of filenames that are never granted opportunistic locks with the `veto oplock files` option. This option can be set either globally or on a per-share basis. For example:

```
veto oplock files = /*.bat/*.htm/
```

The value of this option is a series of patterns. Each pattern entry must begin, end, or be separated from another with a slash (/) character, even if there is only one pattern listed. Asterisks can be used as a wildcard to represent zero or more characters. Questions marks can be used to represent exactly one character.

We recommend that you disable oplocks on any files that are meant to be updated by Unix or are intended to be shared by several processes simultaneously.

lock directory

This option (sometimes called `lock dir`) specifies the location of a directory where Samba will store SMB deny-mode lock files. Samba stores other files in this directory as well, such as browse lists and its shared memory file. If WINS is enabled, the WINS database is written to this directory as well. The default for this option is specified in the Samba makefile; it is typically */usr/local/samba/var/locks*. You can override this location as follows:

```
[global]
    lock directory = /usr/local/samba/locks
```

You typically would not need to override this option, unless you want to move the lock files to a more standardized location, such as */var/spool/locks*.