FILE AND PRINT

First Steps with Samba

PART 1 Bridging the gap between MS Windows and Linux for file and print services, with **Dr Chris Brown.**

My wife sounded puzzled. "Someone from *Linux Format* called. They want you to do a piece about the samba". And no wonder. Why should someone like me, with less physical grace than a JCB, be writing a tutorial about a Brazilian dance, and why might a computing magazine print it? So I explained that this was one of those cute Unixy names like *Apache* and GNU and GNOME, and she said this was clearly some new use of the word 'cute' she hadn't come across before. Will she ever see the light?

Samba, to set the record straight, is a suite of services and programs that runs on Linux and that provides file and print services to Windows clients. It began as a one-man project in 1992 (the one man was Andrew Tridgell) and has since grown into one of the world's flagship open source projects, with a large, active, international development team. (See **www.samba.org**) The name *Samba* comes from the original name of the protocol which Microsoft used for file and printer sharing – Server Message Block, or SMB. (Later, in a typical Microsoft marketing upgrade, the name was changed to "Common Internet File System" or CIFS. The new name is misleading. For a number of reasons, SMB (or CIFS) is really only suitable for deployment across a local area network, not across the Internet. And despite the use of 'common' in the name, it remains primarily a Microsoft-centric protocol.)

Interoperability between Linux and Windows is an essential part of any Linux migration plan. The commercial importance of *Samba* is that it allows file and print services which would otherwise be provided by Windows NT or Windows 2000 machines to be provided on Linux at zero software cost and, probably, with a smaller hardware footprint than Windows. Replacing a few NT file servers with Linux and *Samba* is one of the easiest ways for a company to dip its toes into the Linux waters. Because the Windows client machines can't tell the difference between a share exported by a *Samba* server and a Windows file share, very little reconfiguration is required, and the change is essentially transparent to the end users.

In this two part tutorial, we'll first have a look at how file and printer sharing works in the Windows world and then move on to see how *Samba* fits in. We'll look at some simple *Samba* configurations before progressing to the slightly thornier problem of user identity and authentication.

A Short History of NetBIOS & SMB

It feels a bit odd writing about Microsoft's technologies for a Linux readership, like doing an article on Richard Branson for BA's inflight magazine. But a little background is necessary to understand what *Samba* is all about.

The history of the SMB protocol goes back a long way. In the beginning, (soon after the heavier elements had started condensing out after the Big Bang,) IBM invented the PC, DOS, and a piece of firmware that handled the low-level I/O within the machine called the BIOS (Basic Input/Output System). When PCs started being put onto networks, an extension to the BIOS, called NetBIOS, was developed to handle machine to machine communication. With NetBIOS, each machine had a name, up to 15 characters long, which was used directly to address network packets to their recipients. NetBIOS was a lightweight protocol requiring very little memory, and worked fine for the small LANs for which it was designed. However, it had two major drawbacks – it had no mechanism for routing, and it relied heavily on broadcasting. These features made NetBIOS unsuitable for use either on WANs or on point-to-point links. To overcome these limitations, a way was developed for NetBIOS traffic to be encapsulated into TCP/IP datagrams. This scheme, known as NetBIOS over TCP/IP (NBT) is defined in RFCs 1001 and 1002. The scheme retained the concept of NetBIOS names, but uses IP addresses for routing and delivery. *Samba* uses NBT.

NBT brought with it a requirement for NetBIOS name resolution – that is, it needed a mechanism to convert the NetBIOS names to the underlying IP addresses. Just as there are several ways to do hostname-to-IP lookups, there are several ways of performing NetBIOS name resolution, too. Clients can consult a local file (on Linux it's usually /etc/samba/lmhosts), or they can use a NetBIOS Name Server. Microsoft's implementation of a NetBIOS name server is called WINS (Windows Internet Name Server). As we will see, *Samba* can act as a NetBIOS name server, too. As a last resort, clients will broadcast requests to resolve NetBIOS names.

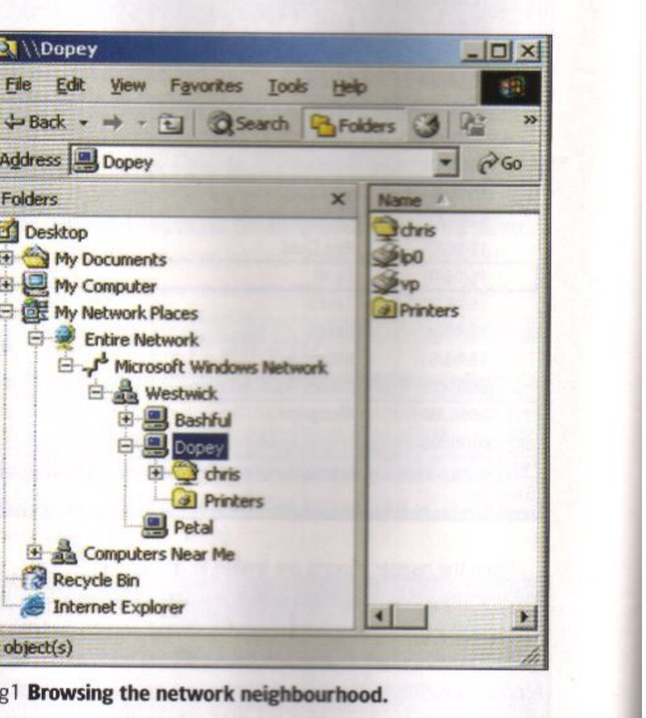

Fig1 **Browsing the network neighbourhood.**

Resource Sharing

There are a couple of concepts central to the Windows resource-sharing model: workgroups and shares. A workgroup is simply a group of machines, identified by a workgroup name, which share resources amongst themselves. A share is a named, advertised resource on a specific machine, usually a filesystem or a printer. For example, my Windows 2000 machine BASHFUL is advertising a file share called projects which corresponds to the directory C:\J2SDK_Forte\projects.

The 'projects' share on the machine BASHFUL is identified by the name '//BASHFUL/projects'. This is an example of what is rather grandly called the Universal Naming Convention, or UNC. (I find the claim of universality hard to take, and have a strong suspicion that its use does not even extend throughout our own galaxy. But I digress.) Note that BASHFUL is the NetBIOS name of the machine, which is not necessarily the same as its TCP/IP hostname – *ie* the name DNS knows it as.

The collection of resources made up by all these shares is known in the Microsoft world as the Network Neighbourhood. You'll almost certainly have browsed a network neighbourhood using *Windows Explorer*; if not, **Fig1** shows what it looks like. Here, we're looking at the machines in the workgroup Westwick, and have expanded the view to show the shares on one specific machine, Dopey.

How does *Windows Explorer* know what machines are in the workgroup? Is there a configuration file somewhere that makes all this explicit? No, what happens is that all machines register with one specific machine on the network called the master browser. The master browser is not pre-determined, it is chosen by holding an 'election' across the network. Potentially, any machine that speaks SMB (including a *Samba* server) can take on this role. The list of machines held in the master browser is called the browse list. (Note that browsing, in this context, is not related to web browsing). In **Fig1**, we see that machines Bashful, Dopey and Petal are in the browse list for the workgroup Westwick. And although you can't tell from the figure, Dopey and Petal are both *Samba* servers running on Linux. The actual list of shares on Dopey, which is expanded in the figure, is not obtained from the browse list; it is obtained by contacting the SMB server on Dopey directly.

For those of you who prefer a command line interface, even on Windows, the command

```
net view \\dopey
```

will show the shares on dopey, and the command:

```
net use m: \\dopey\chris
```

will attach the share chris as drive letter m:

If you're familiar with native Linux file sharing (using NFS), there are a couple of differences between NFS and Microsoft's file-sharing mechanisms to keep in mind. First, SMB has an extra level of naming for exported shares. For example, a server can export the directory C:\images\vintage under the share name 'oldcars'. Clients access the files via the share name; they never see the name of the underlying directory. Using NFS on the other hand, a server might export the directory /images/vintage and the client would use the self-same directory name in an NFS mount command to attach the filesystem; for example:

```
# mount dopey.example.com:/images/vintage /mnt/oldcars
```

The second difference is that NFS doesn't support the notion of workgroups or browse lists. It's true that you can ask a specific NFS server about its exports, for example:

```
# showmount -e dopey.example.com
```

will list the exported file systems on **dopey.example.com**, but you have to just know, somehow, that **dopey** is out there in the first place. There is no command that says "show me all the NFS servers on this network". [See the following note below.] [**Note:** OK, I guess that's not strictly true. You could run a command such as **rpcinfo -b mountd 3** which would tell you how many machines on the local subnet have a mountd daemon listening. But that's not really the same as having an actively maintained browse list.]

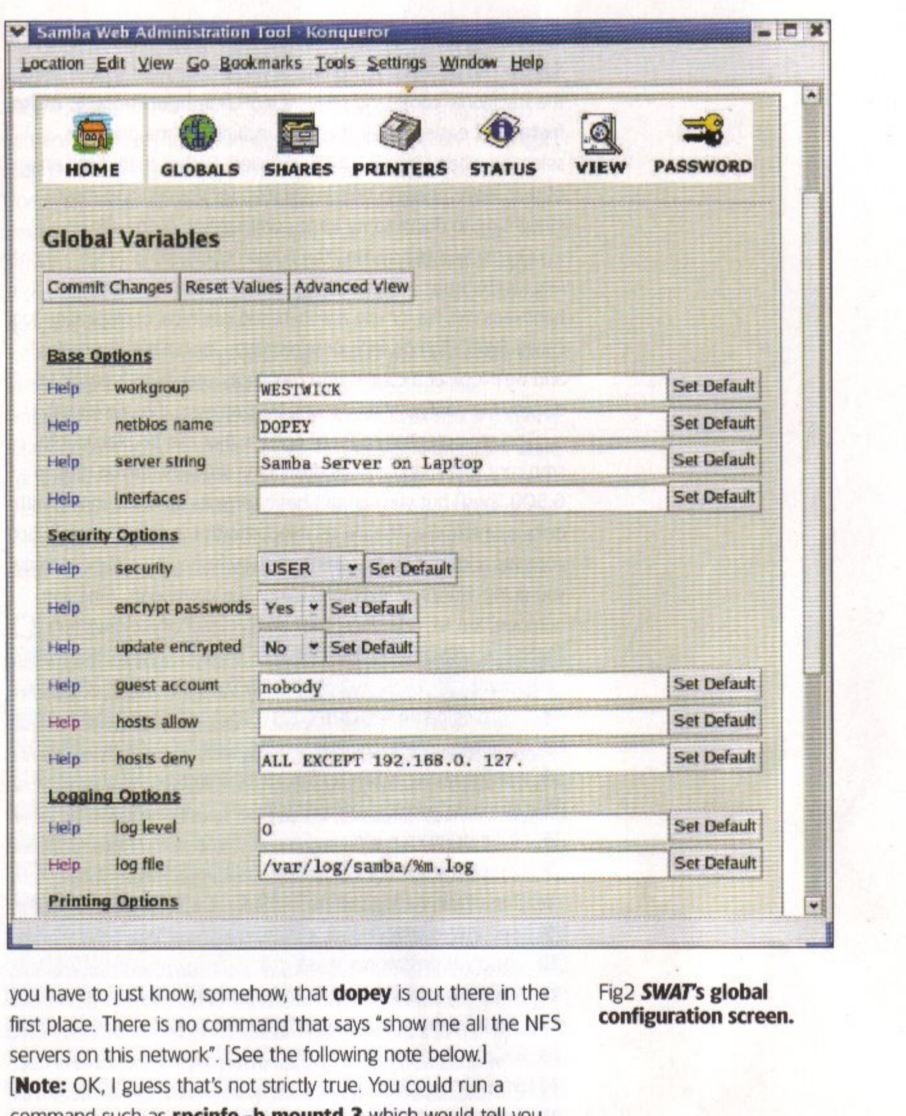

Fig2 **SWAT's global configuration screen.**

Where does Samba fit in?

OK, enough of the Windows networking lecture. Where does *Samba* fit in? Well, *Samba* can act in the role of an SMB file and print server, as a NetBios name server, and as a master browser. The 'server' side of *Samba* actually consists of several daemons, utilities and configuration files, the most important of which are shown in the box immediately below.

Key server-side components
Samba's utilities and configuration

FILE	DESCRIPTION
smbd	The main daemon providing file and print services
nmbd	Daemon providing NetBIOS name resolution (equivalent to Microsoft's WINS) and network neighbourhood browsing
smb.conf	Samba's main configuration file, usually in the directory /etc/samba
testparm	A program which performs a basic syntax check of the smb.conf file
smbpasswd	A tool for managing samba accounts and passwords

Configuring Samba

Samba can be installed from binary RPMs, or from source using the traditional command sequence of **./configure; make; make install**. Or, easiest of all, it can be included in the package selection when Linux is initially installed. Rather than dwell on any of that, we're going to take installation for granted and move straight on to doing some basic configuration of the server.

On my Red Hat system, the *Samba* config file is /etc/samba/smb.conf. Your mileage may differ. You can either hand-edit this file, or use a web-based graphical configuration tool called *SWAT* (*Samba Web Administration Tool*). *SWAT* is prettier, and we'll look at a couple of screenshots later, but it's easier to discuss the configuration options by examining the file directly.

The are potentially a lot of options that can be placed into smb.conf (the manual page describing the file's format runs to 6,500 lines!) but you can get basic services up and running with quite a simple file. The file as shipped with the *Samba* distribution contains a lot of helpful comments which may be sufficient to get you going. Here's an example, minus the comments. (The line numbers are for reference; they are not part of the file.)

```
1 [global]
2   netbios name = DOPEY
3   workgroup = WESTWICK
4   server string = Samba Server on Laptop
5   printcap name = /etc/printcap
6   load printers = yes
7   printing = lprng
8   log file = /var/log/samba/%m.log
9   security = user
10  encrypt passwords = yes
11  smb passwd file = /etc/samba/smbpasswd
12  hosts deny = ALL EXCEPT 192.168.0. 127.
13
14 [cdrom]
15  path = /mnt/cdrom
16  read only = yes
17  guest ok = yes
18
19 [homes]
20  comment = Home Directories
21  browseable = no
22  writable = yes
23  valid users = %S
24  create mode = 0664
25
26 [printers]
27  comment = All Printers
28  path = /var/spool/samba
29  browseable = yes
30  public = yes
31  writable = no
32  printable = yes
```

The section headed [global] (lines **1-12**) define settings that apply to the server as a whole. Lines **2-3** specify the NetBIOS name for this server, and the workgroup it will be part of. Lines **5-7** tell the server to make available as shares all the printers listed in the machines /etc/printcap file. (This is the standard configuration file for the traditional *lprng* print system on Linux.)

Line **8** determines where samba will do its logging. The %m in the file name is a *Samba* variable. *Samba* maintains a number of these and replaces them by the appropriate value. The variable %m is replaced by the NetBIOS name of the client machine. So this line is a Cunning Trick to cause *Samba* to log each client in a separate file. For example, entries for client bashful will be logged to /var/log/samba/bashful.log. The 'log level' directive (not shown in the example) controls the amount of detail written to this file.

Lines **9-12** are concerned with access control. The setting **security = user** means that users have to establish their identity when they connect to the server by supplying a user name and a password; access to shares is then controlled on the basis of that user identity. There is another, fundamentally different, mode called **security = share** in which clients gain access to shares not by establishing their own identity but by supplying a password associated with the share. This security mode was used by early versions of Windows such as Windows 95 but is not recommended for current use.

Line **10** says that passwords sent from windows clients will be encrypted, and line **11** says which file *Samba* should look up the encrypted passwords in. We'll talk more about this in the second part of the tutorial when we discuss authentication in detail. Finally, line **12** says that we'll only accept connections from clients on our local network 192.168.0.0 or from the local machine via its loopback address.

Having completed the global configuration of the server, at lines **14-17** we get to define an actual share. The share name is **cdrom** and the Linux path name corresponding to this share is /mnt/cdrom, on which, presumably, a CD is mounted. Line **16** stops users from trying to write to this share and line **17** allows users to connect to this share without a password. On the Linux machine, these unauthenticated users will assume the identity of the "guest user" which by default is usually the Linux account "nobody". Since nobody doesn't own any files, (sorry about the grammar, but I said what I meant), such users will only have access to world-readable files in the share.

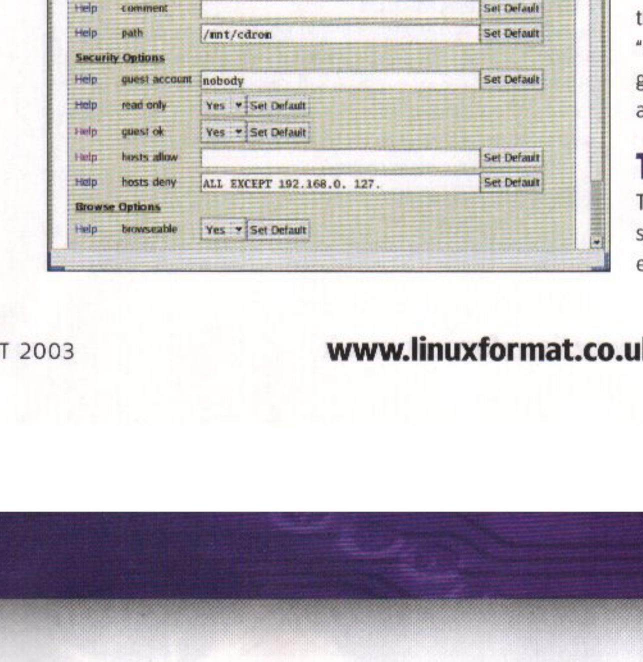

Fig3 **SWAT's share configuration screen.**

The magic [homes] share

The **[homes]** share, defined in lines 19-24 of the file, is treated specially by *Samba*, and provides an easy way to automatically export all users' home directories. It works like this. When a windows user connects to the *Samba* server and authenticates, *Samba* will dynamically create a share named after that user and associate it with that user's home directory on the Linux machine. So if user Mike logs in, he'll see a share called *mike*. If Jane logs in, she'll see a share called *jane*, and so on. This is obviously very convenient when large user populations are being served. Line **21** of the config file stops the **[homes]** share itself from showing up in the browse list; it does not stop the user's individual home share from being visible.

Line **23** uses another of **Samba**'s variables. **%S** will be replaced by the name of the current share, which in this case is the user's name. So this line ensures that the only user who can access this share is the one whose home directory it corresponds to. It would be possible, for example, to add the administrator into the list of users who was allowed access to this share.

The **create mode** directive at line **24** is used to limit the access permissions that newly created files in this share will have. This is probably not the place to start explaining how the octal number 0644 is interpreted (you probably know already if you've used the Linux **chmod** command to set file permissions), but the result is that any files created in this share have (at most) the permissions rw-r--r-- (The whole business of mapping the windows access control model for files onto the UNIX model is very tricky and is something I'd rather not get into right now.)

The last section of the smb.conf file (lines **26-32**) establish printer shares on this server. There is no need to export each printer individually, although you can if you want. As configured, *Samba* will automatically create shares for all printers defined by the underlying print system.

There is a useful little tool which comes as part of the *Samba* suite with the rather generic and unimaginative name of *testparm*. This tool will check the syntax of your smb.conf file and will report all the options in effect, including all the defaults which are not explicitly set in the config file. It's useful to check that you haven't mis-spelled a directive, or tried to specify an illegal value for an option.

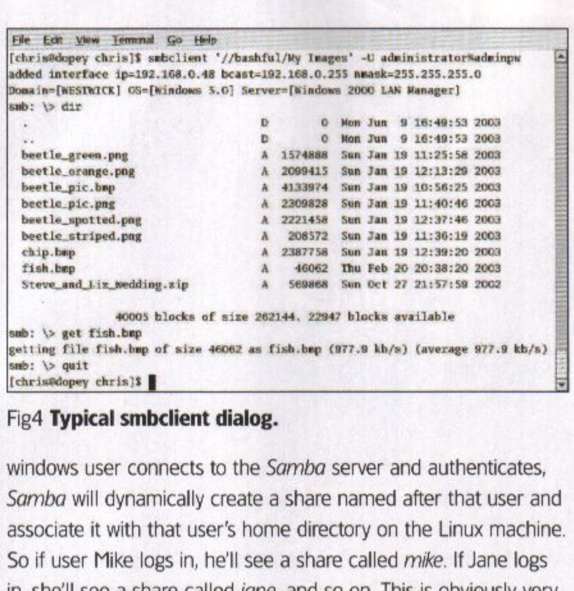

Fig4 **Typical smbclient dialog.**

Configuring with SWAT

If you'd prefer not to have to learn the syntax of the smb.conf file, you can configure it using a graphical, browser based tool called *SWAT*. Having been (to quote Eric Raymond) "marinaded in UNIX" for many years, I'm not a great fan of graphical configuration tools, but as they go, *SWAT* is pretty good. It's a part of the *Samba* suite but may be separately packaged (on my Red Hat system it's in the RPM package *samba-swat*). It's started via *xinetd* and will need to be enabled, either by editing the file /etc/xinetd.d/swat or by running the command **chkconfig swat on**.

Once running, *SWAT* listens for HTTP requests on port 901 and you can connect to it with a web browser by entering a URL such as http://localhost:901. *SWAT* offers several configuration screens allowing you to configure the globals section (see **Fig2**), individual shares (see **Fig3**), or the printers section of the file. There are other screens which allow you to view the text of the config file you've generated, and the status of the running servers. On its home page (not to be confused with the page for the **[home]** share) there are convenient links to all the *Samba* documentation.

As you'll see from **Fig2** and **Fig3**, *SWAT* just provides a fill-in-the-blanks alternative to hand-editing directives into smb.conf. It does, however, ensure that directives which can only take one of a specific set of values are not set to an illegal value. For example, the boolean option **guest ok** can only be set to 'yes' or 'no'. Also, the 'Help' links down the left-hand side of the screen take you directly to the entry in the smb.conf manual page which describes that directive. This is a nice convenience.

Client-side Samba

Although *Samba* is mainly known as a server to provide file, print, name resolution and browsing services from Linux to Windows clients, there is a client side to *Samba*, too. The client-side tools can be used to access shares on a Windows machine, or simply to run tests against a *Samba* server without having access to a Windows machine. One of the key client-side tools is a command line utility called *smbclient*.

For example, in order to list the shares on the machine bashful, use the command:

```
# smbclient -U administrator -L bashful
```

Note that **bashful**, in this example, is the NetBIOS name of the server, which as we've seen is not necessarily related to the DNS name of the machine (assuming it has one). In the example, administrator is the user you're logging in as; you'll be prompted for the password. Alternatively you can enter the password on the command line like this:

```
# smbclient -U administrator%adminpw -L bashful
```

although I'm uncomfortable with this approach because I prefer echoing to be turned off when I type a password.

You can also use *smbclient* to connect to a specific share. If you do this, you'll get an interactive prompt at which *smbclient* offers an ftp-like command repertoire. **Fig4** shows an example of connecting to the share 'My Images' on the server bashful and retrieving the file fish.bmp. The dialog should look familiar to anyone who has used ftp from the command line.

The Linux kernel also supports SMB as a filesystem type, so if you want to do more than just copy a few files across from an SMB share it's probably easier to mount the share onto a local directory. The command would look something like this:

```
# mount -t smbfs -o username=administrator
//bashful/projects /mnt/bashful
```

In this example, **administrator** is the name I'm logging in as on the SMB server, **//bashful/projects** is the UNC share name, and **/mnt/bashful** is the local mount point. Again, you'll be prompted for the password. Now you can simply **cd** to the directory /mnt/bashful and access the files there as if they were local. Let's just repeat that to be perfectly clear: here, we're using Linux as a client to access shares exported from an SMB server, presumably a Windows machine, although there's no technical reason that you couldn't share files between Linux boxes this way, too. LXF

NEXT MONTH

We'll wrap up this two-part tutorial by taking a detailed look at the tricky business of authentication in *Samba*.