Sharing files with Windows users via Samba

In this section, I'll walk you through setting up your very own **Samba** file server. I'll also go over a sample configuration to get you started so that you can add your own shares. First, we'll need to make sure that the `samba` package is installed on our server:

```
sudo apt install samba
```

When you install the `samba` package, you'll have a new daemon installed on your server, `smbd`. The `smbd` daemon will be automatically started and enabled for you. You'll also be provided with a default configuration file for Samba, located at `/etc/samba/smb.conf`. For now, I recommend stopping `samba` since we have yet to configure it:

```
sudo systemctl stop smbd
```

Since we're going to configure Samba from scratch, we should start with an empty configuration file. Let's back up the original file, rather than overwrite it. The default file includes some useful notes and samples, so we should keep it around for future reference:

```
sudo mv /etc/samba/smb.conf /etc/samba/smb.conf.orig
```

Now, we can begin a fresh configuration. Although it's not required, I like to split my Samba configuration up between two files, `/etc/samba/smb.conf` and `/etc/samba/smbshared.conf`. You don't have to do this, but I think it makes the configuration cleaner and easier to read. First, here is a sample `/etc/samba/smb.conf` file:

```
[global]
server string = File Server
workgroup = WORKGROUP
security = user
map to guest = Bad User
name resolve order = host bcast wins
include = /etc/samba/smbshared.conf
```

As you can see, this is a really short file. Basically, we're including only the lines we absolutely need to in order to set up a file server with Samba. Next, I'll explain each line and what it does.

```
[global]
```

With the `[global]` stanza, we're declaring the global section of our configuration, which will consist of settings that will impact Samba as a whole. There will also be additional stanzas for individual shares, which we'll get to later.

```
server string = File Server
```

The `server string` is somewhat of a description field for the `File Server`. If you've browsed networks from Windows computers before, you may have seen this field. Whatever you type here will display underneath the server's name in Windows Explorer. This isn't required, but it's nice to have.

```
workgroup = WORKGROUP
```

Here, we're setting the `workgroup`, which is the exact same thing as a `workgroup` on Windows PCs. In short, the workgroup is a namespace that describes a group of machines. When browsing network shares on Windows systems, you'll see a list of workgroups, and then one or more computers within that workgroup. In short, this is a way to logically group your nodes. You can set this to whatever you like. If you already have a workgroup in your organization, you should set it here to match the workgroup names of your other machines. The default workgroup name is simply `WORKGROUP` on Windows PCs, if you haven't customized the workgroup name at all.

```
security = user
```

This setting sets up Samba to utilize usernames and passwords for authentication to the server. Here, we're setting the `security` mode to `user`, which means we're using local users to authenticate, rather than other options such as `ads` (Active Directory) or `domain` (domain controller), which are both outside the scope of this book.

```
map to guest = Bad User
```

This option configures Samba to treat unauthenticated users as guest users. Basically, unauthenticated users will still be able to access shares, but they will have guest permissions instead of full permissions. If that's not something you want, then you can omit this line from your file. Note that if you do omit this, you'll need to make sure that both your server and client PCs have the same user account names on either side. Ideally, we want to use directory-based authentication, but that's beyond the scope of this book.

```
name resolve order = host bcast wins
```

The `name resolve order` setting configures how Samba resolves hostnames. In this case, we're using the broadcast name first, followed by `wins`. Since `wins` has been pretty much abandoned (and replaced by DNS), we include it here solely for compatibility with legacy networks.

```
include = /etc/samba/smbshared.conf
```

Remember how I mentioned that I usually split my Samba configurations into two different files? On this line, I'm calling that second `/etc/samba/smbshared.conf` file. The contents of the `smbshared.conf` file will be inserted right here, as if we only had one file. We haven't created the `smbshared.conf` file yet. Let's take care of that next. Here's a sample `smbshared.conf` file:

```
[Documents]
path = /share/documents
force user = myuser
force group = users
public = yes
writable = no

[Public]
path = /share/public
force user = myuser
force group = users
create mask = 0664
force create mode = 0664
directory mask = 0777
force directory mode = 0777
public = yes
```

```
writable = yes
```

As you can see, I'm separating share declarations in their own file. We can see several interesting things within `smbshared.conf`. First, we have two stanzas, `[Documents]` and `[Public]`. Each stanza is a share name, which will allow Windows users to access the share under `//servername/share-name`. In this case, this file will give us two shares: `//servername/Documents` and `//servername/Public`. The `Public` share is writable for everyone, though the `Documents` share is restricted to read-only.
The `Documents` share has the following options:

```
path = /share/documents
```

This is the path to the share, which must exist on the server's filesystem. In this case, when a user reads files from `//servername/Documents`on a Windows system, they will be reading data from `/share/documents` on the Ubuntu server that's housing the share.

```
force user = myuser
force group = users
```

These two lines are basically bypassing user ownership. When a user reads this share, they are treated as `myuser` instead of their actual user account. Normally, you would want to set up LDAP or Active Directory to manage your user accounts and handle their mapping to Ubuntu Server, but a full discussion of directory-based user access isn't covered in this book, so I provided the `force` options as an easy starting point. The user account you set here must exist on the server.

```
public = yes
writable = no
```

With these two lines, we're configuring what users are able to do once they connect to this share. In this case, `public = yes` means that the share is publicly available, though `writable = no` prevents anyone from making changes to the contents of this share. This is useful if you want to share files with others, but you want to restrict access and stop anyone from being able to modify the content.

The `Public` share has some additional settings that weren't found in the `Documents` share:

```
create mask = 0664
force create mode = 0664
directory mask = 0777
force directory mode = 0777
```

With these options, I'm setting up how the permissions of files and directories will be handled when new content is added to the share. Directories will be given `777` permissions and files will be given permissions of `664`. Yes, these permissions are very open; note that the share is named `Public`, which implies full access anyway, and its intended purpose is to house data that isn't confidential or restricted:

```
public = yes
writable = yes
```

Just as I did with the previous share, I'm setting up the share to be publicly available, but this time I'm also configuring it to allow users to make changes.

To take advantage of this configuration, we need to start the Samba daemon. Before we do though, we want to double-check that the directories we entered into our `smbshared.conf` file exist, so if you're using my example, you'll need to create `/share/documents` and `/share/public`:

```
sudo mkdir /share
sudo mkdir /share/documents
sudo mkdir /share/public
```

Also, the user account that was referenced in the force user and the group referenced in the force group must both exist and have ownership over the shared directories:

```
sudo chown -R jay:users /share
```

At this point, it's a good idea to use the `testparm` command, which will test the syntax of our Samba configuration files for us. It won't necessarily catch every error we could have made, but it is a good command to run to quickly check the sanity. This command will first check the syntax, and then it will print the entire configuration to the terminal for you to have a chance to review it. If you see no errors here, then you can proceed to start the service:

```
sudo systemctl start smbd
```

Then, check the status to ensure that the daemon is running:

```
sudo systemctl status smbd
```

This will produce output similar to the following:

That really should be all there is to it; you should now have `Documents` and `Public` shares on your file server that Windows users should be able to access. In fact, your Linux machines should be able to access these shares as well. On Windows, **Windows Explorer** has the ability to browse file shares on your network. If in doubt, try pressing the *Windows* key and the *r* key at the same time to open the **Run** dialog box, and then type the **Universal Naming Convention** (**UNC**) path to the share (`\\servername\Documents` or `\\servername\Public`). You should be able to see any files stored in either of those directories. In the case of the `Public` share, you should be able to create new files there as well.

On Linux systems, if you have a desktop environment installed, most of them feature a file manager that supports browsing network shares. Since there are a handful of different desktop environments available, the method varies from one distribution or configuration to another. Typically, most Linux file managers will have a network link within the file manager, which will allow you to easily browse your local shares. In the case of Ubuntu itself, you may need to install `smbclient` and `cifs-utils` via `apt`, so if you get an error you can try installing those packages. In the following screenshot, an Ubuntu 22.04 desktop client is browsing shares on a local Samba server:

Figure 12.2: Browsing a Samba share from a Linux client

If your file manager doesn't show you the available shares on your server, you can also access a Samba share by adding an entry for it in the `/etc/fstab` file, such as the following:

```
//myserver/share/documents /mnt/documents cifs username=myuser,noauto 0 0
```

Then, assuming the local directory exists (`/mnt/documents` in the example `fstab` line), you should be able to mount the share with the following command:

```
sudo mount /mnt/documents
```

In the `fstab` entry, I included the `noauto` option so that your system won't mount the Samba share at boot time (you'll need to do so manually with the `mount` command). If you do want the Samba share automatically mounted at boot time, change `noauto` to `auto`. However, you may receive errors during boot if for some reason the server hosting your Samba shares isn't accessible, which is why I prefer the `noauto` option.

If you'd prefer to mount the Samba share without adding an `fstab` entry, the following example command should do the trick; just change the share name and mount point to match your local configuration:

```
sudo mount -t cifs //myserver/Documents -o username=myuser /mnt/documents
```

From here, feel free to experiment. You can add additional shares as appropriate, and customize your Samba implementation as you see fit. In the next section, we'll explore NFS.