LPI exam 201 prep: File and service sharing

Intermediate Level Administration (LPIC-2) topic 209

Skill Level: Intermediate

Brad Huntting (huntting@glarp.com)
Mathematician
University of Colorado

David Mertz, Ph.D. (mertz@gnosis.cx)
Developer
Gnosis Software

02 Sep 2005

In this tutorial, Brad Huntting and David Mertz continue preparing you to take the Linux Professional Institute® Intermediate Level Administration (LPIC-2) Exam 201. In this fifth of eight tutorials, you learn how to use a Linux™ system as a networked file server using any of several protocols supported by Linux.

Section 1. Before you start

Learn what these tutorials can teach you and how you can get the most from them.

About this series

The Linux Professional Institute (LPI) certifies Linux system administrators at junior and intermediate levels. To attain each level of certification, you must pass two LPI exams.

Each exam covers several topics, and each topic has a weight. The weights indicate the relative importance of each topic. Very roughly, expect more questions on the exam for topics with higher weight. The topics and their weights for LPI exam 201 are:

Topic 201

Linux kernel (weight 5).

Topic 202
System startup (weight 5).

Topic 203
Filesystems (weight 10).

Topic 204
Hardware (weight 8).

Topic 209
File and service sharing (weight 8). The focus of this tutorial.

Topic 211
System maintenance (weight 4).

Topic 213
System customization and automation (weight 3).

Topic 214
Troubleshooting (weight 6).

The Linux Professional Institute does not endorse any third-party exam preparation material or techniques in particular. For details, please contact info@lpi.org.

About this tutorial

Welcome to "File and service sharing," the fifth of eight tutorials designed to prepare you for LPI exam 201. In this tutorial, you learn how to use a Linux system as a networked file server using any of several protocols supported by Linux.

The tutorial is organized according to the LPI objectives for this topic, as follows:

2.209.1 Configuring a Samba server (weight 5)
You will be able to set up a Samba server for various clients. This objective includes setting up a login script for Samba clients and setting up an nmbd WINS server. Also included is changing the workgroup in which a server participates, defining a shared directory in smb.conf, defining a shared printer in smb.conf, using nmblookup to test WINS server functionality, and using the smbmount command to mount an SMB share on a Linux client.

2.209.2 Configuring an NFS server (weight 3)
You will be able to create an exports file and specify filesystems to be exported. This objective includes editing exports file entries to restrict access to certain hosts, subnets, or netgroups. Also included is specifying mount options in the exports file, configuring user ID mapping, mounting an NFS filesystem on a client, and using mount options to specify soft or hard and background retries, signal handling, locking, and block size. You should also be able to

configure tcpwrappers to further secure NFS.

The current LPI exam objectives for topic 209 exam cover NFS and Samba. But if you are a system administrator designing a server configuration, you should also consider whether FTP, SCP/SSH, HTTP, or other protocols might, in fact, meet your requirements.

One of the most significant uses for Linux, particularly in a server context, is to provide shared files to client systems. In fact, in a general way, serving files is probably most of what all networking is used for. This tutorial -- and in fact, this series of tutorials -- will not address peer-to-peer file-sharing servers such as BitTorrent. Rather, this tutorial looks only at older client-server arrangements: A central server that provides disk stores for multiple clients. Even when clients upload files, those are always stored and served by the server, rather than in a decentralized fashion.

Protocols widely used for file serving include HTTP (the WWW), TFTP (Trivial File Transfer Protocol), FTP (File Transfer Protocol), SCP (Secure Copy Protocol, a specialized use of SSH), RCP (Remote Copy Protocol, generally deprecated), NFS (Network File System), and Samba (server message block). HTTP and SSH will be discussed in upcoming tutorials for LPI exam 202, as will security issues around FTP. TFTP and RCP are special purpose or deprecated and will not be addressed in these tutorials.

This tutorial looks at NFS and Samba in some detail and briefly describes FTP. NFS and Samba are network file-sharing protocols that allow mostly transparent access to remote filesystems. FTP might require a custom FTP client program, although many desktop environments or tools (on Linux or otherwise) hide the details of this negotiation and effectively present the same user interface as an NFS- or Samba-mounted drive.

Prerequisites

To get the most from this tutorial, you should already have a basic knowledge of Linux and a working Linux system on which you can practice the commands covered in this tutorial.

Section 2. Configuring an NFS server

Using NFS on a client

If the server is properly configured and the client has appropriate permissions, mounting a remote filesystem with NFS requires only the `mount` command:

```
mount -t nfs my.nfs.server.com:/path/on/server /path/on/client
```

or a suitable entry in /etc/fstab:

```
my.nfs.server.com:/path/on/server /path/on/client nfs rw,soft
0 0
```

The `soft` option tells the kernel to send an I/O error (EIO) to user processes in the event of network difficulties. The default `hard` option will cause processes to hang while the NFS server is unreachable.

In addition, the helper programs `rpc.lockd`, `rpc.statd`, and `rpc.quotad` may be run on client and/or server.

Configuring an NFS server (part one)

An NFS server requires three distinct programs, as well as three optional programs.

When an NFS client mounts an NFS filesystem, it contacts the following server daemons, most of which must run standalone (as opposed to being started from `inetd`):

- `portmap`: Sometimes named `portmapper` or `rpc.bind`.
- `rpc.mountd`: Sometimes `mounted`.
- `rpc.nfsd`: Sometimes `nfsd`.

In addition, there are three optional helper programs: `rpc.lockd`, `rpc.statd`, and `rpc.quotad` which, respectively, provide global locking, accelerate the `lstat` family of syscalls (used by `ls -l`, etc.), and provide support for quotas.

Configuring an NFS server (part two)

All three NFS-related servers use TCPwrappers (`tcpd`) for access controland therefore may require entries in /etc/host.allow.

Neither `nfsd` nor `portmap` normally require any configuration beyond /etc/hosts.allow.

The configuration file for `mountd` is (indirectly) /etc/exports. It says which filesystems can be mounted by which clients. Under the Linux implementation of NFS, /etc/exports is not directly parsed by `mountd`. Instead, the `exportfs -a` command parses /etc/exports and writes the result to /var/lib/nfs/xtab where `mountd` can read it. There are other flags to `exportfs` which allow these two files to be desynchronized. That is, you may temporarily add or remove exported directories without modifying the semi-permanent records in /etc/exports.

Administrators of other Unix-like servers should note that the syntax of the Linux

/etc/exports file differs significantly from that of SunOS or BSD.

Configuring /etc/hosts.allow and /etc/hosts.deny

The configuration file /etc/hosts.allow describes hosts that are allowed to connect to a Linux system. This configuration is not specific to NFS, but a system needs to be permitted to connect in the first place to use and NFS server. Similarly, /etc/hosts.deny is a list of hosts prohibited from connecting.

Slightly unintuitively, first allowed hosts are searched, then denied hosts, but anything left unmatched is granted access. This does not mean that the login mechanisms of individual servers are not still operative, but a cautious administrator might deny anything not explicitly permitted (a little paranoia is good) by using:

```
# /etc/hosts.deny
ALL:ALL EXCEPT localhost:DENY
```

With an /etc/hosts.deny set to deny everything (except connections from LOCALHOST), only those connections explicitly permitted will be allowed. For example:

```
#/etc/hosts.allow
# Allow localhost and intra-net domain to use all servers
ALL : 127.0.0.1, 192.168.
# Let everyone ssh here except 216.73.92.* and .microsoft.com
sshd: ALL EXCEPT 216.73.92. .microsoft.com : ALLOW
# Let users in the *.example.net domain ftp in
ftpd: .example.net
```

Configuring /etc/exports

Here's a sample /etc/export file:

```
# sample /etc/exports file / master(rw)
trusty(rw,no_root_squash) /projects proj*.local.domain(rw)
/usr *.local.domain(ro) @trusted(rw) /home/joe
pc001(rw,all_squash,anonuid=150,anongid=100) /pub
(ro,insecure,all_squash)
```

Normally, `root` (uid 0) on the client is treated as `nobody` (uid 65534) on the server; this is called *root squashing* since it protects files owned by root (and not group/other writable) from being altered by NFS clients. The `no_root_squash` tag disables this behavior and allows the root user on `trusty` full access to the `/` partition. This can be useful for installing and configuring software.

The `/usr` partition will be read only for all hosts except those in the "trusted" netgroup.

When `/home/joe` is mounted by `pc001`, all remote users (regardless of uid/gid) will be treated as if they have uid=150, gid=100. This is useful if the remote NFS client is a single-user workstation or does not support different users (like with DOS).

Normally, Linux (and other Unix-like operating systems) reserves the use of TCP and UDP ports 1-1023 (so called *secure ports*) for use by processes running as root. To ensure that the root user has initiated a remote NFS mount, the NFS server normally requires remote clients to use secure ports when mounting NFS filesystems. This convention, however, is not honored by some operating systems (notably Windows). In such cases, the *insecure* option allows the NFS client to use any TCP/UDP port. This is usually required when serving Windows clients.

NFS utilities

`nfsstat` displays a time series of NFS-related statistics (client and/or server) regarding the local machine similar to `iostat` and `vmstat`.

The `showmount` command queries `mountd` and shows which clients are currently mounting filesystems. As NFS is a stateless protocol and the `mountd` daemon is queried infrequently, the output of `showmount` can become inaccurate. Unfortunately, there is not really any way to force `showmount` to become accurate. However, where it is inaccurate, `showmount` almost always errs in showing stale mounts rather than omitting active mounts (relatively harmlessly).

In this context, "stateless" means that the `nfsd` daemons that serve up the actual file data have no memory of which files are open, nor even which clients have which partitions mounted. Each request (readblock, writeblock, etc.) contains all the information needed to complete it (partition id provided by `mountd`, inode number, block number, read/write/etc., data). The HTTP protocol is similar in this respect. An upside of statelessness if the server reboots, the clients will notice only a brief period of interrupted access.

Section 3. Configuring a samba server

Samba server configuration

The Samba server `smbd` provides file and print services (largely for Windows clients). While it can be started from `inetd`, it is typically run as a stand alone daemon `smbd -D`. `nmbd` is the NetBios nameserver (or WINS server). It too can be run from `inetd`, but is more typically run as a stand alone daemon `nmbd -D`. Samba can function as a server in a Windows WORKGROUP, as well as Primary Domain Controller.

The configuration file for both `smbd` and `nmbd` is /etc/samba/smb.conf. Copious configuration parameters are described in the smb.conf manpage. The lmhosts file is used to map NetBios names to IP addresses. Its format is similar to (but not identical to) the /etc/hosts file.

There are several excellent HOWTOs on the subject of Samba configuration as well as several books. This section touches on the basic ideas with pointers to more complete documentation.

Setting up a home-directory file share

The following smb.conf snippet allows users to access their (local) home directories from remote Samba clients:

```
[homes]
   comment = Home Directories
   browseable = no
```

This is usually included in the default smb.conf file.

Setting up a print share with CUPS

Of the numerous Unix printing systems, CUPS is the least antiquated and probably the currently most popular. Depending on your distribution, CUPS may be enabled in the default smb.conf. Here is a simple example of a CUPS print share:

```
[global]
  load printers = yes
  printing = cups
  printcap name = cups

[printers]
  comment = All Printers
  path = /var/spool/samba
  browseable = no
  public = yes
  guest ok = yes
  writable = no
  printable = yes
  printer admin = root

[print$]
  comment = Printer Drivers
  path = /etc/samba/drivers
  browseable = yes
  guest ok = no
  read only = yes
  write list = root
```

CUPS can provide ppd (Postscript printer description) files and Windows drivers for clients which, when setup properly, allows remote users to take advantage of the full range of a printer's features (color versus black-and-white, resolution, paper-tray select, double- vs. single-sided printing, etc.). Traditional Unix printing systems are quite cumbersome by comparison. Consult the cupsaddsmb manpage for more information.

Authentication

Samba (unlike NFS) requires individual users to authenticate with the server. As with any network-authentication service, care should be taken to insure that passwords

are never passed over the network unencrypted. See the section on encrypting passwords in the smb.conf manpage for details.

There are a variety of mechanisms Samba can use to authenticate remote users (clients). By their nature most of these are incompatible with the standard Unix password hash. The notable exception is when passwords are passed over the wire in the clear, unencrypted, which is almost always a bad idea.

Assuming you encrypt passwords on the wire, `smbpasswd` will usually be used to set up users with an initial Samba password. The "Unix password sync" option allows `smbpasswd` to change Unix passwords whenever users change their Samba password.

Alternatively, the `pam_smb` module, when configured, can authenticate Linux users using the Samba database directly. As if that's not enough choices, LDAP can be used to authenticate Samba and/or Linux users.

Debugging Samba

When configuring a Samba server, the `testparm` (also called `smbtestparm`) command can be quite useful. It will parse the smb.conf file and report any problems.

The `nmblookup` command does for Samba what `nslookup` does for DNS; it queries the NetBios directory. See the `nmblookup` manpage for more details.

Samba client configuration

The `smbclient` command provides FTP-like access to a Samba file share. Transparent access to SMB file shares is trickier; see the `smbmount` manpage or the sharit' package for more info.

Section 4. Configuring File Transfer Protocol servers

About FTP

FTP is an old and widely used network protocol. FTP is normally run over two separate ports, 20 and 21. Port 21 is used as a control stream (transmitting login information and commands) while port 20 is used as the data stream over which actual file content is transmitted.

Generally, FTP is not considered a very secure protocol in the sense that in its

default mode of operation, control information -- login passwords -- are transmitted in the clear. For that matter, data streams are also unencrypted, but FTP shares that feature with NFS and Samba (for secure data channels, SSH/SCP is a better choice). It *is* possible to layer FTP's control port over SSH, hence protecting control information.

Traditional FTP clients provide their own shell environment over which to transmit control commands and configure connections. Sometimes GUI frontends are used to provide friendlier interfaces to FTP transfers. However these days, many non-dedicated tools incorporate FTP -- everything from file managers to text editors are often happy to work with files served by an FTP server.

Anonymous FTP

For what FTP is most often used for, security is not usually an issue. Probably most often, FTP servers are used for "anonymous FTP" -- that is, data that is available to the world at large and therefore doesn't require much security. By convention, a username of *anonymous* is configured to allow access and an identifying password (often an email address) is requested but not verified. Sometimes a username/password is required, but such a combination is provided without any deep user authentication (for instance, with people who want to volunteer for a project).

Most Web browsers and many file managers and tools support FTP servers transparently. Often these tools will use an FTP URL to request a file (or also to upload a file *to* a server). For example, the command-line tool `wget` will retreive files from FTP servers using the following:

```
$ wget ftp://example.net/pub/somefile
$ wget ftp://user:passwd@example.net/pub/somefile
```

File managers will often mount an FTP server in a manner that is essentially identical to a local filesystem or NFS or Samba drive (this does not, however, use the `mount` and /etc/fstab system; such pseudo-partitions are usually named by their URL).

Choices of FTP servers

Given the age and ubiquity of FTP, a bewildering number of implementations are available and installed with various Linux distributions. Configuring the FTP server you decide to use will require a visit to the documentation accompanying the particular server.

Some popular Linux FTP servers include

- wu-ftpd.
- vsftpd.

- ProFTPd.
- BSD ftpd.
- TUX FTP.

There are many less used ones as well. In most every case, the configuration of a server will live in a file like /etc/FOOftpd.conf (for an appropriate value of "FOO"). I am fond of vsftpd which is both fast and avoids known security glitches (the "vs" stands for "very secure").

A sample FTPd configuration file

Given the wealth of servers, configuration syntaxes will differ. But a few concepts taken from /etc/vsftpd.conf illustrate the types of options other servers provide. For vsftpd, each option takes the form `option=value` with the usual hash marks for comment lines. Most other FTPd configuration files are similar.

- `anonymous_enable`: Controls whether anonymous logins are permitted.
- `anon_world_readable_only`: When enabled, anonymous users will only be allowed to download world-readable files.
- `chroot_local_user`: If enabled, local users will be placed in a `chroot()` jail in their home directory after login.
- `pasv_enable`: Should the server use the "passive FTP" style in which clients initiate ports (helps with firewalls at clients).
- `ssl_enable`: If enabled, vsftpd will support SSL secure connections.
- `tcp_wrappers`: If enabled incoming connections will be fed through access control (like /etc/hosts.allow and /etc/hosts.deny).

Launching an FTP server

In the simplest case, you may start an FTP server the same way you might launch any daemon:

```
% sudo vsftpd
```

At this point the server will listen for incoming connections according the rules configured in its configuration file. You may also launch an FTP server from an "network super-server" such as inetd or xinetd. The LPI 202 tutorials will discuss these super-servers.

Launching a daemon individually, even if in appropriate startup scripts -- either for a particular runlevel or in /etc/rcS.d/ -- gives you finer control over the behavior of an FTP server.

Resources

Learn

- At the LPIC Program, find task lists, sample questions, and detailed objectives for the three levels of the Linux Professional Institute's Linux system administration certification.
- "Introduction to Samba" (developerWorks, June 2000) is a two-part series that shows how to set up and configure a Samba server.
- *Samba Installation, Configuration, and Sizing Guide* (IBM Redbook, May 2003) gives you the basics of installing and configuring Samba.
- "Using Network File System" (developerWorks, February 2005) is a primer on using NFS.
- Find more resources for Linux developers in the developerWorks Linux zone.

Get products and technologies

- Order the SEK for Linux, a two-DVD set containing the latest IBM trial software for Linux from DB2®, Lotus®, Rational®, Tivoli®, and WebSphere®.
- Build your next development project on Linux with IBM trial software, available for download directly from developerWorks.

Discuss

- Participate in the discussion forum for this content.
- Get involved in the developerWorks community by participating in developerWorks blogs.

About the authors

Brad Huntting
Brad has been doing UNIX® systems administration and network engineering for about 14 years at several companies. He is currently working on a Ph.D. in Applied Mathematics at the University of Colorado in Boulder, and pays the bills by doing UNIX support for the Computer Science department.

David Mertz, Ph.D.
David Mertz is Turing complete, but probably would not pass the Turing Test. For more on his life, see his personal Web page. He's been writing the developerWorks columns *Charming Python* and *XML Matters* since 2000. Check out his book *Text Processing in Python* .