LPI certification 102 exam prep, Part 4

Presented by developerWorks, your source for great tutorials

ibm.com/developerWorks

Table of Contents

If you're viewing this document online, you can click any of the topics below to link directly to that section.

Section 1. About this tutorial

What does this tutorial cover?

Welcome to "Secure shell and file sharing", the fourth of four tutorials designed to prepare you for the Linux Professional Institute's 102 exam. In this tutorial, we'll introduce you to the secure shell (ssh) and related tools and show you how to use and configure Network File System (NFS) version 3 servers and clients. By studying this series of tutorials (eight in all; this is part eight), you'll have the knowledge you need to become a Linux Systems Administrator and will be ready to attain an LPIC Level 1 certification (exams 101 and 102) from the Linux Professional Institute if you so choose.

The LPI logo is a trademark of the *Linux Professional Institute*.

Should I take this tutorial?

This tutorial is ideal for those who want to learn about or improve their basic Linux networking and file sharing skills. It's especially appropriate for those who will be setting up applications on Linux servers or desktops. For many, much of this material will be new, but more experienced Linux users may find this tutorial to be a great way of rounding out their important Linux system administration skills.

If you are new to Linux, we recommend that you first complete the previous tutorials in the LPI certification 101 and 102 exam prep series before continuing:

- *101 series, Part 1: Linux fundamentals*
- *101 series, Part 2: Basic administration*
- *101 series, Part 3: Intermediate administration*
- *101 series, Part 4: Advanced administration*
- *102 series, Part 1: Compiling sources and managing packages*
- *102 series, Part 2: Compiling and configuring the kernel*
- *102 series, Part 3: Networking*

About the author

For technical questions about the content of this tutorial, contact the author, Daniel Robbins, at *drobbins@gentoo.org*.

Residing in Albuquerque, New Mexico, Daniel Robbins is the President/CEO of Gentoo Technologies, Inc., the creator of *Gentoo Linux*, an advanced Linux for the PC, and the **Portage** system, a next-generation ports system for Linux. He has also served as a contributing author for the Macmillan books *Caldera OpenLinux Unleashed*, *SuSE Linux Unleashed*, and *Samba Unleashed*. Daniel has been involved with computers in some fashion since the second grade, when he was first exposed to the Logo programming language as well as to a potentially dangerous dose of Pac Man. This probably explains why he has since served as a Lead Graphic Artist at **SONY Electronic Publishing/Psygnosis**. Daniel enjoys spending time with his wife, Mary, and their daughter, Hadassah.

Section 2. Secure shell

Interactive logins

Back in the old days, if you wanted to establish an interactive login session over the network, you used `telnet` or `rsh`. However, as networking became more popular, these tools became less and less appropriate, because they're horrendously insecure.

The data going between the telnet client and server isn't encrypted, and can thus be read by anyone snooping the network. Not only that, but *authentication* (the sending of your password to the server) is performed in plain text, making it a trivial matter for someone capturing your network data to get instant access to your password. In fact, using a network sniffer, it's possible for someone to reconstruct your entire telnet session, seeing everything on the screen that you saw!

Obviously, these tools that were designed with the assumption that the network was secure and unsniffable are inappropriate for today's distributed and public networks.

Secure shell

A better solution was needed, and that solution came in the form of a tool called secure shell, or `ssh`. The most popular modern incarnation of this tool is available in the openssh package, available for virtually every Linux distribution, not to mention many other systems.

What sets `ssh` apart from its insecure cousins is that it *encrypts* all communications between the client and the server using strong encryption. By doing this, it becomes difficult (impossible, even) to monitor the communications between the client and server. In this way, `ssh` provides its service as advertised -- it is a *secure* shell. In fact, `ssh` has excellent "all-round" security -- even authentication takes advantage of encryption and various key exchange strategies to ensure that the user's password cannot be easily grabbed by anyone monitoring data being transmitted over the network.

In this age of the popularization of the Internet, `ssh` is a valuable tool for enhancing network security when using Linux systems. Most security-savvy network admins discourage the use of -- or even don't allow the use of -- telnet and rsh on their systems at all because `ssh` is such a capable and secure replacement.

Using ssh

Generally, most distributions' openssh packages can be used without any manual configuration. After installing openssh, you'll have a couple of binaries. One is, of course, `ssh` -- the secure shell client that can be used to connect to any system running `sshd`, the secure shell server. To use `ssh`, you typically start a session by typing something like:

```
$ ssh drobbins@otherbox
```

Above, I instruct `ssh` to login as the "drobbins" user account on remotebox. As with telnet, you'll be prompted for a password; after entering it, you'll be presented with a new login session on the remote system.

Starting sshd

If you want to allow `ssh` connections to your machine, you'll need to start the `sshd` server. To start the `sshd` server, you would typically use the rc-script that came with the openssh package, typing something like:

```
# /etc/init.d/sshd start
```

or

```
# /etc/rc.d/init.d/sshd start
```

If necessary, you can adjust configuration options for `sshd` by modifying the /etc/ssh/sshd_config file. For more information on the various options available, type `man sshd`.

Secure copy

The openssh package also comes with a handy tool called `scp`, which stands for "secure copy". You can use this command to securely copy files to and from various systems on the network. For example, if I wanted to copy ~/foo.txt to my home directory on remotebox, I could type:

```
$ scp ~/foo.txt drobbins@remotebox:
```

After being prompted for my password on remotebox, the copy will be performed. Or, if I wanted to copy a file called bar.txt in remotebox's /tmp directory to my current working directory on my local system, I could type:

```
$ scp drobbins@remotebox:/tmp/bar.txt .
```

Secure shell authentication options

Openssh also has a number of other authentication methods. Used properly, they can allow you to authenticate with remote systems without having to type in a password or passphrase for every connection. To learn more about how to do this, read the *developerWorks* openssh key management articles (listed in Resources, the last section of this tutorial).

Section 3. NFS

Introducing NFS

The Network File System (NFS) is a technology that allows the transparent sharing of files between UNIX and Linux systems connected via a Local Area Network, or LAN. NFS has been around for a long time; it's well known and used extensively used in the Linux and UNIX worlds. In particular, NFS is often used to share home directories among many machines on the network, providing a consistent environment for a user when he or she logs in to a machine (*any* machine) on the LAN. Thanks to NFS, it's possible to mount remote filesystem trees and have them fully integrated into a system's local filesystem. NFS' transparency and maturity is what makes it such a useful and popular choice for network file sharing under Linux.

NFS basics

To share files using NFS, you first need to set up an NFS server. This NFS server can then "export" filesystems. When a filesystem is exported, it means that it is made available to be accessed by other systems on the LAN. Then, any authorized system that is also set up as an NFS client can mount this exported filesystem using the standard "mount" command. After the mount completes, the remote filesystem is "grafted in" in the same way that a locally-mounted filesystem (like /mnt/cdrom) would be after it is mounted. The fact that all of the file data is being read from the NFS server rather than from a disk is simply not an issue to any standard Linux application. Everything simply works.

Attributes of NFS

Shared NFS filesystems have a number of interesting attributes. The first "nifty attribute" is a result of NFS' stateless design. Because client access to the NFS server is stateless in nature, it's possible for the NFS server to reboot without causing client applications to crash or fail. All access to remote NFS files will simply "pause" until the server comes back online. Also, because of NFS' stateless design, NFS servers can handle large numbers of clients without any additional overhead besides that of transferring the actual file data over the network. In other words, NFS performance is dependant on the amount of NFS data being transferred over the network, rather than the number of machines that happen to be requesting said data.

NFS version 3 under Linux

When setting up NFS, it's strongly recommended that you use NFS version 3 rather than version 2. Version 2 has some significant problems with file locking and generally has a bad reputation for breaking certain applications. On the other hand, NFS version 3 is very nice and robust and does its job well. Now that Linux 2.2.18+ supports NFS 3 clients and servers, there's no reason at all to consider using NFS 2 anymore.

Securing NFS

It's important to mention that NFS version 2 and 3 have some very clear security limitations. They were designed to be used in a specific environment -- a secure, trusted LAN. In particular, NFS 2 and 3 were designed to be used on a LAN where "root" access to the machine is only allowed by administrators. Due to the design of NFS 2 and NFS 3, if a malicious user has "root" access to a machine on your LAN, he or she will be capable of bypassing NFS security and very likely be able to access or even modify files on the NFS server that he or she wouldn't normally be able to otherwise. For this reason, NFS should not be deployed casually. If you're going to use NFS on your LAN, great -- but set up a firewall first. Make sure that people outside your LAN won't be able to access your NFS server. Then, make sure that your internal LAN is relatively secure, and that you are fully aware of all the hosts participating in your LAN. Once your LAN's security has been thoroughly reviewed and (if necessary) improved, you're ready to safely use NFS (see Part 7 of this tutorial series for more on this).

Section 4. Setting up NFS

Setting up NFS under Linux

The first step in using NFS 3 is to set up an NFS 3 server. Choose the system that will be serving files to the rest of your LAN. On this machine, we'll need to enable NFS server support in the kernel. You should use a 2.2.18+ kernel (2.4+ recommended) to take advantage of NFS 3, which is much more stable than NFS 2. If you're compiling your own custom kernel, enter your /usr/src/linux directory and run `make menuconfig`. Then, select the "File systems section," then the "Network File Systems" section, and ensure that the following options are enabled:

```
<*> NFS file system support
[*]   Provide NFSv3 client support
<*> NFS server support
[*]   Provide NFSv3 server support
```

Getting ready for /etc/exports

Next, compile and install your new kernel and reboot. Your system will now have NFS 3 server and client support built-in.

Now that our NFS server has support for NFS in the kernel, it's time to set up an /etc/exports file. The /etc/exports file will describe the local filesystems that will be made available for export, as well as which hosts will be able to access these filesystems, and whether they will be exported as read/write or read-only. It will also allow us to specify other options that control NFS behavior.

But before we look at the format of the /etc/exports file, a big fat implementation warning is in order! The NFS implementation in the Linux kernel only allows the export of one local directory per filesystem. This means that if both /usr and /home are on the same ext3 filesystem (using /dev/hda6, for example), then you can't have both /usr and /home export lines in /etc/exports. If you try to add these lines, you'll see errors like this when your /etc/exports file gets reread (which will happen if you type `exportfs -ra` after your NFS server is up and running):

```
sidekick:/home: Invalid argument
```

Working around export restrictions

Here's how to work around this problem. If /home and /usr are on the same underlying local filesystem, you can't export them both. So just export /. NFS clients will then be able to mount /home and /usr via NFS just fine, but your NFS server's /etc/exports file will now be "legal," containing only one export line per underlying local filesystem. Now that you understand this implementation of Linux NFS, we're ready to take a look at the format of /etc/exports.

The /etc/exports file

Probably the best way to understand the format of /etc/exports is to look at a quick example. Here's a simple /etc/exports file that I use on my NFS server:

```
# /etc/exports: NFS file systems being exported.  See exports(5).
/ 192.168.1.9(rw,no_root_squash)
/mnt/backup 192.168.1.9(rw,no_root_squash)
```

As you can see, the first line in my /etc/exports file is a comment. On the second line, I select my root ("/") filesystem for export. Note that while this exports everything under "/", it will not export any other local filesystem. For example, if my NFS server has a CD-ROM mounted at /mnt/cdrom, the contents of the CDROM will not be available unless they are exported explicitly in /etc/exports. Now, notice the third line in my /etc/exports file. On this line, I export /mnt/backup; as you might guess, /mnt/backup is on a separate filesystem from /, and it contains a backup of my system. Each line also has a "192.168.1.9(rw,no_root_squash)" on it. This information tells nfsd to only make these exports available to the NFS client with the IP address of 192.168.1.9. It also tells nfsd to make these filesystems writeable as well as readable by NFS client systems, and instructs the NFS server to allow the remote NFS client to allow a superuser account to have true "root" access to the filesystems.

Another /etc/exports file

Here's an /etc/exports that will export the same filesystems as the one in the previous panel, except that it will make my exports available to all machines on my LAN -- 192.168.1.1 through 192.168.1.254:

```
# /etc/exports: NFS file systems being exported.  See exports(5).
/ 192.168.1.1/24(rw,no_root_squash)
/mnt/backup 192.168.1.1/24(rw,no_root_squash)
```

In this sample /etc/exports file, I use a host mask of /24 to mask out the last eight bits in the IP address I specify. It's also very important that there is no space between the IP address specification and the "(", or NFS will interpret your information incorrectly. And, as you might guess, there are other options that one can specify besides "rw" and "no_root_squash"; type "man exports" for a complete list.

Starting the NFS 3 server

Once /etc/exports is configured, you're ready to start your NFS server. Most distributions will have an "nfs" initialization script that can be used to start NFS -- type `/etc/init.d/nfs start` or `/etc/rc.d/init.d/nfs start` to use it. Once NFS is started, typing `rpcinfo` should display output that looks something like this:

```
# rpcinfo -p
   program vers proto   port
    100000    2   tcp    111  portmapper
    100000    2   udp    111  portmapper
    100024    1   udp  32802  status
    100024    1   tcp  46049  status
```

```
    100011    1   udp    998  rquotad
    100011    2   udp    998  rquotad
    100003    2   udp   2049  nfs
    100003    3   udp   2049  nfs
    100003    2   tcp   2049  nfs
    100003    3   tcp   2049  nfs
    100021    1   udp  32804  nlockmgr
    100021    3   udp  32804  nlockmgr
    100021    4   udp  32804  nlockmgr
    100021    1   tcp  48026  nlockmgr
    100021    3   tcp  48026  nlockmgr
    100021    4   tcp  48026  nlockmgr
    100005    1   udp  32805  mountd
    100005    1   tcp  39293  mountd
    100005    2   udp  32805  mountd
    100005    2   tcp  39293  mountd
    100005    3   udp  32805  mountd
    100005    3   tcp  39293  mountd
```

Changing export options

If you ever change your /etc/exports file while your NFS daemons are running, simply type `exportfs -ra` to apply your changes. Now that your NFS server is up and running, you're ready to configure NFS clients so that they can mount your exported filesystems:

Configuring NFS clients

Kernel configuration for NFS 3 clients is similar to that of the NFS server, except that you only need to ensure that the following options are enabled:

```
<*> NFS file system support
[*]   Provide NFSv3 client support
```

Starting NFS client services

To start the appropriate NFS client daemons, you can typically use a system initialization script called "nfslock" or "nfsmount". Typically, this script will start rpc.statd, which is all the NFS 3 client needs -- rpc.statd allows file locking to work properly. Once all your client services are set up, running `rpcinfo` on the local machine will display output that looks like this:

```
# rpcinfo
   program vers proto   port
    100000    2   tcp    111  portmapper
    100000    2   udp    111  portmapper
    100024    1   udp  32768  status
    100024    1   tcp  32768  status
```

You can also perform this check from a remote system by typing `rpcinfo -p myhost`, as follows:

```
# rpcinfo -p sidekick
   program vers proto   port
    100000    2   tcp    111  portmapper
    100000    2   udp    111  portmapper
    100024    1   udp  32768  status
    100024    1   tcp  32768  status
```

Mounting exported NFS filesystems

Once both client and server are set up correctly (and assuming that the NFS server is configured to allow connections from the client), you can go ahead and mount an exported NFS filesystem on the client. In this example, `inventor` is the NFS server and `sidekick` (IP address 192.168.1.9) is the NFS client. Inventor's /etc/exports file contains a line that looks like this, allowing connections from any machine on the 192.168.1 network:

```
/ 192.168.1.1/24(rw,no_root_squash)
```

Now, logged into `sidekick` as root, we can type:

```
# mount inventor:/ /mnt/nfs
```

Inventor's root filesystem will now be mounted on sidekick at /mnt/nfs; you should now be able to type `cd /mnt/nfs` and look around inside and see inventor's files. Again, note that if inventor's /home tree is on another filesystem, then /mnt/nfs/home will not contain anything -- another `mount` (as well as another entry in inventor's /etc/exports file) will be required to access that data.

Mounting directories *inside* exports

Note that inventor's `/ 192.168.1.1/24(rw,no_root_squash)` line will also allow us to mount directories *inside* /. For example, if inventor's /usr is on the same physical filesystem as /, and you are only interested in mounting inventor's /usr on sidekick, you could type:

```
# mount inventor:/usr /mnt/usr
```

Inventor's /usr tree will now be NFS mounted to the pre-existing /mnt/usr directory. It's important to again note that inventor's /etc/exports file didn't need to explicitly export /usr; it was included "for free" in our "/" export line.

Section 5. Resources and feedback

Resources

This wraps up not only this tutorial, but also our LPI 102 exam series. We hope you've enjoyed the ride! While the tutorial is over, learning never ends. You'll find useful instruction in the following resources, particularly if you plan to take the LPI 102 exam:

The best thing you can do to improve your NFS skills is to try setting up your own NFS 3 server and client(s) -- the experience will be invaluable. The second-best thing you can do is to read *the Linux NFS HOWTO*, which is quite a good HOWTO.

Learn more about what ssh is capable of in the *developerWorks* series on ssh: *Part 1*, *Part 2*, and *Part 3*. Also be sure to visit the home of openssh at *http://www.openssh.com*, which is an excellent place to continue your study of this important tool.

Samba is another important networked file-sharing technology. For more information about Samba, read the *developerWorks* Samba articles: the *first "Key concepts" article*, *Samba installation* article, and *Samba configuration* article.

Once you're up to speed on Samba, spend some time studying the *Linux DNS HOWTO*. The LPI 102 exam is also going to expect that you have some familiarity with Sendmail. Red Hat has a good *Red Hat Sendmail HOWTO* that will help to get you up to speed.

Be sure to also check out the following general resources:

http://www.linuxdoc.org has an excellent collection of guides, HOWTOs, FAQs, and man-pages. While there, be sure to check out *Linux Gazette* and *LinuxFocus*.

The Linux Network Administrators guide, available from *Linuxdoc.org's "Guides" section*, is a good complement to this series of tutorials -- give it a read! You may also find Eric S. Raymond's *Unix and Internet Fundamentals HOWTO* to be helpful.

In the *Bash by example* article series on *developerWorks*, learn how to use `bash` programming constructs to write your own `bash` scripts. This series (particularly parts 1 and 2) are excellent additional preparation for the LPI exam:

- *Bash by example, Part 1: Fundamental programming in the Bourne-again shell*
- *Bash by example, Part 2: More bash programming fundamentals*
- *Bash by example, Part 3: Exploring the ebuild system*

The *Technical FAQ for Linux Users* by Mark Chapman is a 50-page in-depth list of frequently-asked Linux questions, along with detailed answers. The FAQ itself is in PDF (Acrobat) format. If you're a beginning or intermediate Linux user, you really owe it to yourself to check this FAQ out. The *Linux glossary for Linux users*, also from Mark, is also excellent.

If you're not too familiar with the `vi` editor, you should check out Daniel's *tutorial on Vi*. This *developerWorks* tutorial will give you a gentle yet fast-paced introduction to this powerful text editor. Consider this must-read material if you don't know how to use `vi`.

For more information on the Linux Professional Institute, visit the *LPI home page*.

Your feedback

We look forward to getting your feedback on this tutorial. Additionally, you are welcome to contact the lead author, Daniel Robbins, directly at *drobbins@gentoo.org*.

Colophon

This tutorial was written entirely in XML, using the developerWorks Toot-O-Matic tutorial generator. The open source Toot-O-Matic tool is an XSLT stylesheet and several XSLT extension functions that convert an XML file into a number of HTML pages, a zip file, JPEG heading graphics, and two PDF files. Our ability to generate multiple text and binary formats from a single source file illustrates the power and flexibility of XML. (It also saves our production team a great deal of time and effort.)

You can get the source code for the Toot-O-Matic at www6.software.ibm.com/dl/devworks/dw-tootomatic-p. The tutorial Building tutorials with the Toot-O-Matic demonstrates how to use the Toot-O-Matic to create your own tutorials. developerWorks also hosts a forum devoted to the Toot-O-Matic; it's available at www-105.ibm.com/developerworks/xml_df.nsf/AllViewTemplate?OpenForm&RestrictToCategory=11. We'd love to know what you think about the tool.