LPI certification 102 (release 2) exam prep, Part 1

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. Before you start

About this tutorial

Welcome to "Compiling sources and managing packages," the first of four tutorials designed to prepare you for the Linux Professional Institute's 102 exam. In this tutorial, we'll show you how to compile programs from sources, how to manage shared libraries, and how to use the Red Hat and Debian package management systems.

This tutorial on compiling sources and managing packages is ideal for those who want to learn about or improve their Linux package management skills. This tutorial is particularly appropriate for those who will be setting up applications on Linux servers or desktops. For many readers, much of this material will be new, but more experienced Linux users may find this tutorial to be a great way to "round out" their important Linux system administration skills. If you are new to Linux, we recommend that you start with *Part 1* and work through the series from there.

By the end of this *series* of tutorials (eight in all, covering the LPI 101 and 102 exams), you'll have the knowledge you need to become a Linux Systems Administrator and will be ready to attain an LPIC Level 1 certification from the Linux Professional Institute if you so choose.

For those who have taken the *release 1 version* of this tutorial for reasons other than LPI exam preparation, you probably don't need to take this one. However, if you do plan to take the exams, you should strongly consider reading this revised tutorial.

About the authors

For technical questions about the content of this tutorial, contact the authors:

- Daniel Robbins, at *drobbins@gentoo.org*
- Chris Houser, at *chouser@gentoo.org*
- Aron Griffis, at *agriffis@gentoo.org*

Daniel Robbins lives in Albuquerque, New Mexico, and is the Chief Architect 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 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.

Chris Houser, known to his friends as "Chouser," has been a UNIX proponent since 1994 when joined the administration team for the computer science network at Taylor University in Indiana, where he earned his Bachelor's degree in Computer Science and Mathematics. Since then, he has gone on to work in Web application programming, user interface design, professional video software support, and now Tru64 UNIX device driver programming at

Compaq. He has also contributed to various free software projects, most recently to Gentoo Linux. He lives with his wife and two cats in New Hampshire.

Aron Griffis graduated from Taylor University with a degree in Computer Science and an award that proclaimed him the "Future Founder of a Utopian UNIX Commune". Working towards that goal, Aron is employed by Compaq writing network drivers for Tru64 UNIX, and spending his spare time plunking out tunes on the piano or developing Gentoo Linux. He lives with his wife Amy (also a UNIX engineer) in Nashua, NH.

Section 2. Shared libraries

Introducing shared libraries

On Linux systems there are two fundamentally different types of Linux executable programs. The first are called *statically linked* executables. Static executables contain all the functions that they need to execute -- in other words, they're "complete." Because of this, static executables do not depend on any external library to run.

The second are *dynamically linked* executables. We'll get into those in the next panel.

Static vs. dynamic executables

We can use the `ldd` command to determine if a particular executable program is static:

```
# ldd /sbin/sln
        not a dynamic executable
```

"not a dynamic executable" is `ldd`'s way of saying that `sln` is statically linked. Now, let's take a look at `sln`'s size in comparison to its non-static cousin, `ln`:

```
# ls -l /bin/ln /sbin/sln
-rwxr-xr-x    1 root     root        23000 Jan 14 00:36 /bin/ln
-rwxr-xr-x    1 root     root       381072 Jan 14 00:31 /sbin/sln
```

As you can see, `sln` is over *ten* times the size of `ln`. `ln` is so much smaller than `sln` because it is a *dynamic executable*. Dynamic executables are *incomplete* programs that depend on external shared libraries to provide many of the functions that they need to run.

Dynamically linked dependencies

To view a list of all the shared libraries upon which `ln` depends, use the `ldd` command:

```
# ldd /bin/ln
        libc.so.6 => /lib/libc.so.6 (0x40021000)
        /lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0x40000000)
```

As you can see, `ln` depends on the external shared libraries `libc.so.6` and `ld-linux.so.2`. As a rule, dynamically linked programs are much smaller than their statically-linked equivalents. However, statically-linked programs come in handy for certain low-level maintenance tasks. For example, `sln` is the perfect tool to modify various library symbolic links that exist in `/lib`. But in general, you'll find that nearly all executables on a Linux system are of the dynamically linked variety.

The dynamic loader

So, if dynamic executables don't contain everything they need to run, what part of Linux has

the job of loading them along with any necessary shared libraries so that they can execute correctly? The answer is something called the *dynamic loader*, which is actually the `ld-linux.so.2` library that you see listed as a shared library dependency in `ln`'s `ldd` listing. The dynamic loader takes care of loading the shared libraries that dynamically linked executables need in order to run. Now, let's take a quick look at how the dynamic loader finds the appropriate shared libraries on your system.

ld.so.conf

The dynamic loader finds shared libraries thanks to two files -- `/etc/ld.so.conf` and `/etc/ld.so.cache`. If you `cat` your `/etc/ld.so.conf` file, you'll probably see a listing that looks something like this:

```
$ cat /etc/ld.so.conf
/usr/X11R6/lib
/usr/lib/gcc-lib/i686-pc-linux-gnu/2.95.3
/usr/lib/mozilla
/usr/lib/qt-x11-2.3.1/lib
/usr/local/lib
```

The `ld.so.conf` file contains a listing of all directories (besides `/lib` and `/usr/lib`, which are automatically included) in which the dynamic loader will look for shared libraries.

ld.so.cache

But before the dynamic loader can "see" this information, it must be converted into an `ld.so.cache` file. This is done by running the `ldconfig` command:

```
# ldconfig
```

When ldconfig completes, you now have an up-to-date `/etc/ld.so.cache` file that reflects any changes you've made to `/etc/ld.so.conf`. From this point forward, the dynamic loader will look in any new directories that you specified in `/etc/ld.so.conf` when looking for shared libraries.

ldconfig tips

To view all the shared libraries that `ldconfig` can "see," type:

```
# ldconfig -p | less
```

There's one other handy trick you can use to configure your shared library paths. Sometimes, you'll want to tell the dynamic loader to try to use shared libraries in a specific directory before trying any of your `/etc/ld.so.conf` paths. This can be handy in situations where you are running an older application that doesn't work with the currently-installed versions of your libraries.

LD_LIBRARY_PATH

To instruct the dynamic loader to check a certain directory first, set the `LD_LIBRARY_PATH` variable to the directories that you would like searched. Separate multiple paths using commas; for example:

```
# export LD_LIBRARY_PATH="/usr/lib/old:/opt/lib"
```

After `LD_LIBRARY_PATH` has been exported, any executables started from the current shell will use libraries in `/usr/lib/old` or `/opt/lib` if possible, falling back to the directories specified in `/etc/ld.so.conf` if some shared library dependencies are still unsatisfied.

We've completed our coverage of Linux shared libraries. To learn more about shared libraries, type `man ldconfig` and `man ld.so`.

Section 3. Compiling applications from sources

Introduction

Let's say you find a particular application that you'd like to install on your system. Maybe you need to run a very recent version of this program, but this most recent version isn't yet available in a packaging format such as rpm. Perhaps this particular application is only available in source form, or you need to enable certain features of the program that are not enabled in the rpm by default.

Whatever the reason, whether of necessity or simply just because you *want* to compile the program from its sources, this section will show you how.

Downloading

Your first step will be to locate and download the sources that you want to compile. They'll probably be in a single archive with a trailing .tar.gz, tar.Z, tar.bz2, or .tgz extension. Go ahead and download the archive with your favorite browser or ftp program. If the program happens to have a Web page, this would be a good time to visit it to familiarize yourself with any installation documentation that may be available.

The program you're installing could depend on the existence of any number of other programs that may or may not be currently installed on your system. If you know for sure that your program depends on other programs or libraries that are not currently installed, you'll need to get these packages installed first (either from a binary package like rpm or by compiling them from their sources also.) Then, you'll be in a great position to get your original source file successfully installed.

Unpacking

Unpacking the source archive is relatively easy. If the name of your archive ends with .tar.gz, .tar.Z, or .tgz, you should be able to unpack the archive by typing:

```
$ tar xzvf archivename.tar.gz
```

(x is for extract, z is for gzip decompression, v is for verbose (print the files that are extracted), and `f` means that the filename will appear next on the command line.)

Nearly all "source tarballs" will create one main directory that contains all the program's sources. This way, when you unpack the archive, your current working directory isn't cluttered with lots of files -- instead, all files are neatly organized in a single, new directory and don't get in the way.

Listing archives

Every now and then, you may come across an archive that, when decompressed, creates tons of files in your current working directory. While most tarballs aren't created this way, it's been known to happen. If you want to verify that your particular tarball was put together

correctly and creates a main directory to hold the sources, you can view its contents by typing:

```
$ tar tzvf archivename.tar.gz | more
```

(`t` is for a *text* listing of the archive. No extraction occurs.)

If there is no common directory listed on the left-hand side of the archive listing, you'll want to create a new directory, move the tarball inside it, enter the directory, and only then extract the tarball. Otherwise, you'll be in for a mess!

Unpacking bzip2-compressed archives

It's possible that your archive may be in .tar.bz2 format. Archives with this extension have been compressed with bzip2. Bzip2 generally compresses significantly better than gzip. Its only disadvantage is that compression and decompression are slower, and bzip2 consumes more memory than gzip while running. For modern computers, this isn't much of an issue, which is why you can expect bzip2 to become more and more popular as time goes on.

Because bzip2 has been gaining popularity, modern versions of GNU tar recognize the `j` option to mean "this tarball is compressed with bzip2." When tar encounters the `j` option, it will auto-decompress the tarball (by calling the "bzip2" program) before it tries to open the tarball. For example, here's the command to view the contents of a .tar.bz2 file:

```
$ tar tjvf archive.tar.bz2 | less
```

And here is the command to view the contents of a .tar.gz file:

```
$ tar tzvf archive.tar.gz | less
```

And here is the command to view the contents of a .tar (uncompressed) file:

```
$ tar tvf archive.tar | less
```

bzip2 pipelines

So, your version of tar doesn't recognize those handy bzip2 shortcuts -- what can be done? Fortunately, there's an easy way to extract the contents of bzip2 tarballs that will work on nearly all UNIX systems, even if the system in question happens to have a non-GNU version of tar. To view the contents of a bzip2 file, we can create a shell pipeline:

```
$ cat archive.tar.bz2 | bzip2 -d | tar tvf - | less
```

This next pipeline will actually extract the contents of archive.tar.bz2:

```
$ cat archive.tar.bz2 | bzip2 -d | tar xvf -
```

bzip2 pipelines (continued)

In the previous two examples, we created a standard UNIX pipeline to view and extract files

from our archive file. Since tar was called with the `f -` option, it read tar data from stdin, rather than trying to read data from a file on disk.

If you used the pipeline method to try to extract the contents of your archive and your system complained that bzip2 couldn't be found, it's possible that bzip2 isn't installed on your system. You can download the sources to bzip2 from *http://sources.redhat.com/bzip2*. After installing the bzip2 sources (by following this tutorial), you'll then be able to unpack and install the application you wanted to install in the first place :)

Inspecting sources

Once you've unpacked your sources, you'll want to enter the unpacked directory and check things out. It's always a good idea to locate any installation-related documentation. Typically, this information can be found in a README or INSTALL file located in the main source directory. Additionally, look for README.platform and INSTALL.platform files, where platform is the name of your particular operating system -- in this case "Linux."

Configuration

Many modern sources contain a *configure* script in the main source directory. This script (typically generated by the developers using the GNU autoconf program) is specially designed to set up the sources so that they compile perfectly on your system. When run, the configure script probes your system, determining its capabilities, and creates *Makefiles*, which contain instructions for building and installing the sources on your system.

The configure script is almost always called "configure." If you find a configure script in the main source directory, odds are good that it was put there for your use. If you can't find a configure script, then your sources probably come with a standard Makefile that has been designed to work across a variety of systems -- this means that you can skip the following configuration steps, and resume this tutorial where we start talking about "make."

Using configure

Before running the configure script, it's a good idea to get familiar with it. By typing `./configure --help`, you can view all the various configuration options that are available for your program. Many of the options you see, especially the ones listed at the top of the `--help` printout, are standard options that will be found in nearly every configure script. The options listed near the end are often related to the particular package you're trying to compile. Take a look at them and note any you'd like to enable or disable.

The --prefix option

Most GNU autoconf-based configure scripts have a `--prefix` option that allows you to control where your program is installed. By default, most sources install into the /usr/local prefix. This means that binaries end up in /usr/local/bin, man pages in /usr/local/man, etc. This is normally what you want; /usr/local is commonly used to store programs that you compile yourself.

Using --prefix

If you'd like the sources to install somewhere else, say in /usr, you'll want to pass the `--prefix=/usr` option to configure. Likewise, you could also tell configure to install to your /opt tree, by using the `--prefix=/opt` option.

What about FHS?

Sometimes, a particular program may default to installing some of its files to non-standard locations on disk. In particular, a source archive may have a number of installation paths that do not follow the Linux Filesystem Hierarchy Standard (FHS). Fortunately, the configure script doesn't just permit the changing of the install prefix, but also allows us to change the install location for various system components such as man pages.

This capability comes in very handy, since most source archives aren't yet FHS-compliant. Often, you'll need to add a `--mandir=/usr/share/man` and a `--infodir=/usr/share/info` to the configure command line in order to make your source package configure itself to eventually install its files in the "correct" locations.

Time to configure

Once you've taken a look at the various configure options and determined which ones you'd like to use, it's time to run configure. Please note that you may not *need* to include any command-line options when you run configure -- in the majority of situations, the defaults will work (but may not be *exactly* what you want).

Time to configure (continued)

To run configure, type:

```
$ ./configure <options>
```

This could look like:

```
$ ./configure
```

or

```
$ ./configure --prefix=/usr --mandir=/usr/share/man --infodir=/usr/share/info --with-th
```

The options you need will depend on the particular package you're configuring. When you run configure, it will spend a minute or two detecting whatfeatures or tools are available on your system, printing out the results of its various configuration checks as it goes.

config.cache

Once the configuration process completes, the configure script stores all its configuration

data in a file called `config.cache`. This file lives in the same directory as the configure script itself. If you ever need to run `./configure` again after you've updated your system configuration, make sure you `rm config.cache` first; otherwise, configure will simply use the old settings without rechecking your system.

configure and Makefiles

After the configure script completes, it's time to compile the sources into a running program. A program called *make* is used to perform this step. If your software package contained a configure script, then when you ran it, configure created Makefiles that were specially customized for your system. These files tell the make program how to build the sources and install the resultant binaries, man pages, and support files.

Makefile intro

Makefiles are typically named makefile or Makefile. There will normally be one makefile in each directory that contains source files, in addition to one that sits in the main source directory. The autoconf-generated Makefiles contain instructions (officially called *rules*) that specify how to build certain *targets*, like the program you want to install. make figures out the order in which all the rules should run.

Invoking make

Invoking make is easy; just type "make" in the current directory. The make program will then find and interpret a file called makefile or Makefile in the current directory. If you type "make" all by itself, it will build the *default target*. Developers normally set up their makefiles so that the default target compiles all the sources:

```
$ make
```

Some makefiles won't have a default target, and you'll need to specify one in order to get the compilation started:

```
$ make all
```

After typing one of these commands, your computer will spend several minutes compiling your program into object code. Presuming it completes with no errors, you'll be ready to install the compiled program onto your system.

Installation

After the program is compiled, there's one more important step: installation. Although the program is compiled, it's not yet ready for use. All its components need to be copied from the source directory to the correct "live" locations on your filesystem. For example, all binaries need to be copied to /usr/local/bin, and all man pages need to be installed into /usr/local/man, etc.

Before you can install the software, you'll need to become the root user. This is typically done by either logging in as root on a separate terminal or typing `su`, at which point you'll be prompted for root's password. After typing it in, you'll have root privileges until you exit from your current shell session by typing "exit" or hitting control-D. If you're already root, you're ready to go!

make install

Installing sources is easy. In the main source directory, simply type:

```
# make install
```

Typing "make install" will tell make to satisfy the "install" target; this target is traditionally used to copy all the freshly created source files to the correct locations on disk so that your program can be used. If you didn't specify a `--prefix` option, it's very likely that quite a few files and directories will be copied to your /usr/local tree. Depending on the size of the program, the install target may take anywhere from several seconds to a few minutes to complete.

In addition to simply copying files, make install will also make sure the installed files have the correct ownership and permissions. After `make install` completes successfully, the program is installed and ready (or *almost* ready) for use!

Once it's installed

Now that your program is installed, what's next? Running it, of course! If you're not familiar with how to use the program you just installed, you'll want to read the program's man page by typing:

```
$ man programname
```

It's possible that a program may require additional configuration steps. For example, if you installed a Web server, you'll need to configure it to start automatically when your system boots. You may also need to customize a configuration file in /etc before your application will run.

Ta da!

Now that you've fully installed a software package from its sources, you can run it! To start the program, type:

```
$ programname
```

Congratulations!

Possible problems

It's very possible that configure or make, or possibly even make install, aborted with some kind of error code. The next several panels will help you correct common problems.

Missing libraries

Every now and then, you may experience a problem where configure bombs out because you don't have a certain library installed. In order for you to continue the build process, you'll need to temporarily put your current program configuration on hold and track down the sources or binary package for the library that your program needs. Once the correct library is installed, configure or make should be happy and complete successfully.

Other problems

Sometimes, you'll run into some kind of error that you simply don't know how to fix. As your experience with UNIX/Linux grows, you'll be able to diagnose more and more seemingly cryptic error conditions that you encounter during the configure and make process.

Sometimes, errors occur because an installed library is too old (or possibly even too new!). Other times, the problem you're having is actually the fault of the developers, who may not have anticipated their program running on a system such as yours -- or maybe they just made a typo :)

Other problems (continued)

For problems such as these, use your best judgment to determine where to go for help. If this is your first attempt at compiling a program from source, this may be a good time to select another, easier program to compile. Once you get the simpler program compiled, you may have the necessary experience to fix your originally encountered problem. As you continue to learn more about how UNIX works, you'll get closer to the point where you can actually "tweak" Makefiles and sources to get even seemingly flaky code to compile cleanly!

Section 4. Package management concepts

Package management advantages

Beyond building applications from sources, there's another method for installing software on your Linux system. All Linux distributions employ some form of package management for installing, upgrading, and uninstalling software packages. Package management offers clear advantages over installing directly from source:

- Ease of installation and uninstallation
- Ease of upgrading already-installed packages
- Protection of configuration files
- Simple tracking of installed files

Package management disadvantages

Before jumping into instructions for using the most popular package management tools, I'll acknowledge that there are some Linux users who dislike package management. They might propose some of the following downsides:

- Binaries built for a specific system perform better
- Resolving package dependencies is a headache
- Package database corruption can render a system unmaintainable
- Packages are hard to create

There is some truth to these statements, but the general consensus among Linux users is that the advantages outweigh the disadvantages. Additionally, each stumbling block listed above has a corresponding rebuttal: Multiple packages can be built to optimize for different systems; package managers can be augmented to resolve dependencies automatically; databases can be rebuilt based on other files; and the initial effort expended in creating a package is mitigated by the ease of upgrading or removing that package later.

Section 5. rpm, the (R)ed Hat (P)ackage (M)anager

Getting started with rpm

The introduction of Red Hat's rpm in 1995 was a huge step forward for Linux distributions. Not only did it make possible the management of packages on Red Hat Linux, but due to its GPL license, rpm has become the defacto standard for open source packaging.

The rpm program has a command-line interface by default, although there are GUIs and Web-based tools to provide a friendlier interface. In this section we'll introduce the most common command-line operations, using the Xsnow program for the examples. If you would like to follow along, you can download the Xsnow rpm below, which should work on most rpm-based distributions.

- *xsnow-1.41-1.i386.rpm*

Note: If you find the various uses of the term "rpm" confusing in this section, keep in mind that "rpm" usually refers to the program, whereas "an rpm" or "the rpm" usually refers to an rpm package.

Installing an rpm

To get started, let's install our Xsnow rpm using `rpm -i`:

```
# rpm -i xsnow-1.41-1.i386.rpm
```

If this command produced no output, then it worked! You should be able to run Xsnow to enjoy a blizzard on your X desktop. Personally, we prefer to have some visual feedback when we install an rpm, so we like to include the `-h` (hash marks to indicate progress) and `-v` (verbose) options:

```
# rpm -ivh xsnow-1.41-1.i386.rpm
xsnow                       ##################################################
```

Re-installing an rpm

If you were following along directly, you might have seen the following message from rpm in the previous example:

```
# rpm -ivh xsnow-1.41-1.i386.rpm
package xsnow-1.41-1 is already installed
```

There may be occasions when you wish to re-install an rpm, for instance if you were to accidentally delete the binary /usr/X11R6/bin/xsnow. In that case, you should first remove the rpm with `rpm -e`, then re-install it. Note that the information message from rpm in the following example does not hinder the removal of the package from the system:

```
# rpm -e xsnow
removal of /usr/X11R6/bin/xsnow failed: No such file or directory

# rpm -ivh xsnow-1.41-1.i386.rpm
xsnow                       ##################################################
```

Forcefully installing an rpm

Sometimes removing an rpm isn't practical, particularly if there are other programs on the system that depend on it. For example, you might have installed an "x-amusements" rpm which lists Xsnow as a dependency, so using `rpm -e` to remove Xsnow is disallowed:

```
# rpm -e xsnow
error: removing these packages would break dependencies:
        /usr/X11R6/bin/xsnow is needed by x-amusements-1.0-1
```

In that case, you could re-install Xsnow using the `--force` option:

```
# rpm -ivh --force xsnow-1.41-1.i386.rpm
xsnow                       ##################################################
```

Installing or removing with --nodeps

An alternative to using `--force` in the previous panel would be to remove the rpm using the `--nodeps` option. This disables rpm's internal dependency checking, and is *not recommended* in most circumstances. Nonetheless, it is occasionally useful:

```
# rpm -e --nodeps xsnow

# rpm -ivh xsnow-1.41-1.i386.rpm
xsnow                       ##################################################
```

You can also use `--nodeps` when installing an rpm. To re-iterate what was said above, using `--nodeps` is not recommended, however it is sometimes necessary:

```
# rpm -ivh --nodeps xsnow-1.41-1.i386.rpm
xsnow                       ##################################################
```

Upgrading packages

There is now an rpm of Xsnow version 1.42 on the Xsnow author's Website. You may want to upgrade your existing Xsnow installation for your particular version of Linux. If you were to use `rpm -ivh --force`, it would appear to work, but rpm's internal database would list *both* versions as being installed. Instead, you should use `rpm -U` to upgrade your installation:

```
# rpm -Uvh xsnow-1.42-1.i386.rpm
xsnow                       ##################################################
```

Here's a little trick: we rarely use `rpm -i` at all, because `rpm -U` will simply install an rpm if it doesn't exist yet on the system. This is especially useful if you specify multiple packages on the command-line, where some are currently installed and some are not:

```
# rpm -Uvh xsnow-1.42-1.i386.rpm xfishtank-2.1tp-1.i386.rpm
xsnow                       ##################################################
xfishtank                   ##################################################
```

Querying with rpm -q

You might have noticed in the examples that installing an rpm requires the full filename, but removing an rpm requires only the name. This is because rpm maintains an internal database of the currently installed packages, and you can reference installed packages by name. For example, let's ask rpm what version of Xsnow is installed:

```
# rpm -q xsnow
xsnow-1.41-1
```

In fact, rpm knows even more about the installed package than just the name and version. We can ask for a lot more information about the Xsnow rpm using `rpm -qi`:

```
# rpm -qi xsnow
Name        : xsnow                         Relocations: (not relocateable)
Version     : 1.41                               Vendor: Dan E. Anderson http://www.dan.d
Release     : 1                             Build Date: Thu 10 May 2001 01:12:26 AM EDT
Install date: Sat 02 Feb 2002 01:00:43 PM EST      Build Host: danx.drydog.com
Group       : Amusements/Graphics           Source RPM: xsnow-1.41-1.src.rpm
Size        : 91877                            License: Copyright 1984, 1988, 1990, 1993
Packager    : Dan E. Anderson http://dan.drydog.com/
URL         : http://www.euronet.nl/~rja/Xsnow/
Summary     : An X Window System based dose of Christmas cheer.
Description :
The Xsnow toy provides a continual gentle snowfall, trees, and Santa
Claus flying his sleigh around the screen on the root window.
Xsnow is only for the X Window System, though; consoles just get coal.
```

Listing files with rpm -ql

The database maintained by rpm contains quite a lot of information. We've already seen that it keeps track of what versions of packages are installed, and their associated information. It can also list the files owned by a given installed package using `rpm -ql`:

```
# rpm -ql xsnow
/etc/X11/applnk/Games/xsnow.desktop
/usr/X11R6/bin/xsnow
/usr/X11R6/man/man1/xsnow.1x.gz
```

Combined with the `-c` option or the `-d` option, you can restrict the output to configuration or documentation files, respectively. This type of query is more useful for larger rpms with long file lists, but we can still demonstrate using the Xsnow rpm:

```
# rpm -qlc xsnow
/etc/X11/applnk/Games/xsnow.desktop

# rpm -qld xsnow
/usr/X11R6/man/man1/xsnow.1x.gz
```

Querying packages with rpm -qp

If you had the information available with `rpm -qi` *before* installing the package, you might have been able to better decide whether or not to install it. Actually, using `rpm -qp` allows you to query an rpm file instead of querying the database. All of the queries we've seen so far can be applied to rpm files as well as installed packages. Here are all the examples again, this time employing the `-p` option:

```
# rpm -qp xsnow-1.41-1.i386.rpm
xsnow-1.41-1

# rpm -qpi xsnow-1.41-1.i386.rpm
[same output as rpm -qi in the previous panel]

# rpm -qpl xsnow-1.41-1.i386.rpm
/etc/X11/applnk/Games/xsnow.desktop
/usr/X11R6/bin/xsnow
/usr/X11R6/man/man1/xsnow.1x.gz

# rpm -qplc xsnow-1.41-1.i386.rpm
/etc/X11/applnk/Games/xsnow.desktop

# rpm -qpld xsnow-1.41-1.i386.rpm
/usr/X11R6/man/man1/xsnow.1x.gz
```

Querying all installed packages

You can query all the packages installed on your system by including the `-a` option. If you pipe the output through `sort` and into a pager, then it's a nice way to get a glimpse of what's installed on your system. For example:

```
# rpm -qa | sort | less
[output omitted]
```

Here's how many rpms we have installed on one of our systems:

```
# rpm -qa | wc -l
    287
```

And here's how many files are in all those rpms:

```
# rpm -qal | wc -l
  45706
```

Here's a quick tip: Using `rpm -qa` can ease the administration of multiple systems. If you

redirect the sorted output to a file on one machine, then do the same on the other machine, you can use the `diff` program to see the differences.

Finding the owner for a file

Sometimes it's useful to find out what rpm owns a given file. In theory, you could figure out what rpm owns /usr/X11R6/bin/xsnow (pretend you don't remember) using a shell construction like the following:

```
# rpm -qa | while read p; do rpm -ql $p | grep -q '^/usr/X11R6/bin/xsnow$' && echo $p; d
xsnow-1.41-1
```

Since this takes a long time to type, and even longer to run (1m50s on one of our Pentiums), the rpm developers thoughtfully included the capability in rpm. You can query for the owner of a given file using `rpm -qf`:

```
# rpm -qf /usr/X11R6/bin/xsnow
xsnow-1.41-1
```

Even on the Pentium, that only takes 0.3s to run. And even fast typists will enjoy the simplicity of `rpm -qf` compared to the complex shell construction :)

Showing dependencies

Unless you employ options such as `--nodeps`, rpm normally won't allow you to install or remove packages that break dependencies. For example, you can't install Xsnow without first having the X libraries on your system. Once you have Xsnow installed, you can't remove the X libraries without removing Xsnow first (and probably half of your installed packages).

This is a strength of rpm, even if it's frustrating sometimes. It means that when you install an rpm, it should just *work*. You shouldn't need to do much extra work, since rpm has already verified that the dependencies exist on the system.

Sometimes when you're working on resolving dependencies, it can be useful to query a package with the `-R` option to learn about everything it expects to be on the system. For example, the Xsnow package depends on the C library, the math library, the X libraries, and specific versions of rpm:

```
# rpm -qpR xsnow-1.41-1.i386.rpm
rpmlib(PayloadFilesHavePrefix) <= 4.0-1
ld-linux.so.2
libX11.so.6
libXext.so.6
libXpm.so.4
libc.so.6
libm.so.6
libc.so.6(GLIBC_2.0)
libc.so.6(GLIBC_2.1.3)
rpmlib(CompressedFileNames) <= 3.0.4-1
```

You can also query the installed database for the same information by omitting the `-p`:

```
# rpm -qR xsnow
```

Verifying the integrity of a package

When you download an rpm from the Web or an ftp site, for the sake of security you may want to verify its integrity before installing. All rpms are "signed" with an MD5 sum. Additionally, some authors employ a PGP or GPG signature to further secure their packages. To check the signature of a package, you can use the `--checksig` option:

```
# rpm --checksig xsnow-1.41-1.i386.rpm
xsnow-1.41-1.i386.rpm: md5 GPG NOT OK
```

Wait a minute! According to that output, the GPG signature is *NOT OK*. Let's add some verbosity to see what's wrong:

```
# rpm --checksig -v xsnow-1.41-1.i386.rpm
xsnow-1.41-1.i386.rpm:
MD5 sum OK: 8ebe63b1dbe86ccd9eaf736a7aa56fd8
gpg: Signature made Thu 10 May 2001 01:16:27 AM EDT using DSA key ID B1F6E46C
gpg: Can't check signature: public key not found
```

So, the problem is that we couldn't retrieve the author's public key. After we retrieve the public key from the *package author's Website* (shown in the output from `rpm -qi`), the signature checks out:

```
# gpg --import dan.asc
gpg: key B1F6E46C: public key imported
gpg: /root/.gnupg/trustdb.gpg: trustdb created
gpg: Total number processed: 1
gpg:               imported: 1
```

```
# rpm --checksig xsnow-1.41-1.i386.rpm
xsnow-1.41-1.i386.rpm: md5 gpg OK
```

Verifying an installed package

Similarly to checking the integrity of an rpm, you can also check the integrity of your installed files using `rpm -V`. This step makes sure that the files haven't been modified since they were installed from the rpm:

```
# rpm -V xsnow
```

Normally this command displays no output to indicate a clean bill of health. Let's spice things up and try again:

```
# rm /usr/X11R6/man/man1/xsnow.1x.gz
```

```
# cp /bin/sh /usr/X11R6/bin/xsnow

# rpm -V xsnow
S.5....T   /usr/X11R6/bin/xsnow
missing    /usr/X11R6/man/man1/xsnow.1x.gz
```

This output shows us that the Xsnow binary fails MD5 sum, file size, and mtime tests. And the man page is missing altogether! Let's repair this broken installation:

```
# rpm -e xsnow
removal of /usr/X11R6/man/man1/xsnow.1x.gz failed: No such file or directory

# rpm -ivh xsnow-1.41-1.i386.rpm
xsnow                       ##################################################
```

Configuring rpm

Rpm rarely needs configuring. It simply works out of the box. In older versions of rpm, you could change things in `/etc/rpmrc` to affect run-time operation. In recent versions, that file has been moved to `/usr/lib/rpm/rpmrc`, and is not meant to be edited by system administrators. Mostly it just lists flags and compatibility information for various platforms (e.g. i386 is compatible with all other x86 architectures).

If you wish to configure rpm, you can do so by editing `/etc/rpm/macros`. Since this is rarely necessary, we'll let you read about it in the rpm bundled documentation. You can find the right documentation file with the following command:

```
# rpm -qld rpm | grep macros
```

Additional rpm resources

That's all we've got in this tutorial regarding the Red Hat Package Manager. You should have enough information to administer a system, and there are a lot more resources available. Be sure to check out some of these links:

- *Xsnow Website*
- *rpm Home Page*
- *Maximum RPM - an entire book*
- *The RPM HOWTO at the Linux Documentation Project*
- *Red Hat's chapter on package management with rpm*
- *developerWorks article on creating rpms*
- *rpmfind.net -- a huge collection of rpms*

Section 6. Debian package management

Introducing apt-get

The Debian package management system is made up of several different tools. The command-line tool apt-get is the easiest way to install new packages. For example, to install the program Xsnow, do this as the root user:

```
# apt-get install xsnow
Reading Package Lists... Done
Building Dependency Tree... Done
The following NEW packages will be installed:
  xsnow
0 packages upgraded, 1 newly installed, 0 to remove and 10 not upgraded.
Need to get 17.8kB of archives. After unpacking 42.0kB will be used.
Get:1 http://ftp-mirror.internap.com stable/non-free xsnow 1.40-6 [17.8kB]
Fetched 17.8kB in 0s (18.4kB/s)
Selecting previously deselected package xsnow.
(Reading database ... 5702 files and directories currently installed.)
Unpacking xsnow (from .../archives/xsnow_1.40-6_i386.deb) ...
Setting up xsnow (1.40-6) ...
```

Skimming through this output, you can see that Xsnow was to be installed, then it was fetched it from the Web, unpacked, and finally set up.

Simulated install

If apt-get notices that the package you are trying to install depends on other packages, it will automatically fetch and install those as well. In the last example, only Xsnow was installed, because all of it's dependencies were already satisfied.

Sometimes, however, the list of packages apt-get needs to fetch can be quite large, and it is often useful to see what is going to installed before you let it start. The `-s` option does exactly this. For example, on one of our systems if we try to install the graphical e-mail program `balsa`:

```
# apt-get -s install balsa
Reading Package Lists... Done
Building Dependency Tree... Done
The following extra packages will be installed:
  esound esound-common gdk-imlib1 gnome-bin gnome-libs-data imlib-base libart2
  libaudiofile0 libesd0 libglib1.2 libgnome32 libgnomesupport0 libgnomeui32
  libgnorba27 libgnorbagtk0 libgtk1.2 libjpeg62 liborbit0 libpng2 libproplist0
  libtiff3g libungif3g zlib1g
The following NEW packages will be installed:
  balsa esound esound-common gdk-imlib1 gnome-bin gnome-libs-data imlib-base
  libart2 libaudiofile0 libesd0 libglib1.2 libgnome32 libgnomesupport0
  libgnomeui32 libgnorba27 libgnorbagtk0 libgtk1.2 libjpeg62 liborbit0 libpng2
  libproplist0 libtiff3g libungif3g zlib1g
0 packages upgraded, 24 newly installed, 0 to remove and 10 not upgraded.
```

It then goes on to list the order in which the packages will be installed and configured (or set up).

Package resource list: apt-setup

Since apt-get is automatically fetching packages for you, it must know something about where to find packages that haven't yet been installed. This knowledge is kept in `/etc/apt/sources.list`. Although you can edit this file by hand (see the `sources.list` man page), you may find it easier to use an interactive tool:

```
# apt-setup
```

This tool walks you through the process of finding places to get Debian packages, such as CDROMs, Web sites, and ftp sites. When you're done, it writes out changes to your `/etc/apt/sources.list` file, so that apt-get can find packages when you ask for them.

From apt-get to dselect

The apt-get tool has many command-line options that you can read about in the apt-get man page. The defaults usually work fine, but if you find yourself using the same option frequently, you may want to add a setting to your `/etc/apt/apt.conf` file. This syntax for this configuration file is described in the `apt.conf` man page.

apt-get also has many other commands besides the `install` command we've used so far. One of these is `apt-get dselect-upgrade`, which obeys the Status set for each package on your Debian system.

Starting dselect

The Status for each package is stored in the file `/var/lib/dpkg/status`, but it is best updated using another interactive tool:

```
# dselect
Debian GNU/Linux `dselect' package handling frontend.

 * 0. [A]ccess      Choose the access method to use.
   1. [U]pdate      Update list of available packages, if possible.
   2. [S]elect      Request which packages you want on your system.
   3. [I]nstall     Install and upgrade wanted packages.
   4. [C]onfig      Configure any packages that are unconfigured.
   5. [R]emove      Remove unwanted software.
   6. [Q]uit        Quit dselect.

Move around with ^P and ^N, cursor keys, initial letters, or digits;
Press <enter> to confirm selection.   ^L redraws screen.

Version 1.6.15 (i386).  Copyright (C) 1994-1996 Ian Jackson.   This is
free software; see the GNU General Public License version 2 or later for
copying conditions.  There is NO warranty.  See dselect --license for details.
```

Using dselect Select mode

You can view and change each package's Status by choosing the Select option. It will then display a screenful of help. When you're done reading this, press space. Now you will see a list of packages that looks something like this:

```
EIOM Pri Section  Package      Inst.ver    Avail.ver   Description
     All packages
       Newly available packages
         New Important packages
           New Important packages in section admin
  n* Imp admin    at           <none>      3.1.8-10    Delayed job execution and
  n* Imp admin    cron         <none>      3.0pl1-57.3 management of regular bac
  n* Imp admin    logrotate    <none>      3.2-11      Log rotation utility
           New Important packages in section doc
  n* Imp doc      info         <none>      4.0-4       Standalone GNU Info docum
  n* Imp doc      manpages     <none>      1.29-2      Man pages about using a L
           New Important packages in section editors
  n* Imp editors  ed           <none>      0.2-18.1    The classic unix line edi
  n* Imp editors  nvi          <none>      1.79-16a.1  4.4BSD re-implementation
           New Important packages in section interpreters
  n* Imp interpre perl-5.005   <none>      5.005.03-7. Larry Wall's Practical Ex
           New Important packages in section libs
  n* Imp libs     libident     <none>      0.22-2      simple RFC1413 client lib
  n* Imp libs     libopenldap- <none>      1.2.12-1    OpenLDAP runtime files fo
  n* Imp libs     libopenldap1 <none>      1.2.12-1    OpenLDAP libraries.
  n* Imp libs     libpcre2     <none>      2.08-1      Philip Hazel's Perl Compa
```

The package Status

The Status for each package can be seen under the somewhat cryptic heading `EIOM`. The column we care about is under the `M` character, where each package is marked with one of the following:

To change the Mark, just press the key for the code you want (equal, dash, or underline), but if you want to change the Mark to `*` (asterisk), you have to press + (plus).

When you are done, use an upper-case `Q` to save your changes and exit the Select screen. If you need help at any time in `dselect`, type ? (question mark). Type a space to get back out of a help screen.

Install and Configure (dpkg-reconfigure)

Debian doesn't install or remove packages based on their Status settings until you run something like `apt-get dselect-upgrade`. This command actually does several steps for you at once -- Install, Remove, and Configure. The Install and Remove steps shouldn't need to stop to ask you any questions. The Configure step, however, may ask any number of questions in order to set up the package just the way you want it.

There are other ways to run these steps. For example, you can choose each step individually from the main `dselect` menu.

Some packages use a system called `debconf` for their Configure step. Those that do can ask their setup questions in a variety of ways, such as in a text terminal, through a graphical

interface, or through a Web page. To configure one of these packages, use the `dpkg-reconfigure` command. You can even use it to make sure *all* `debconf` packages have been completely configured:

```
# dpkg-reconfigure --all
debconf: package "3c5x9utils" is not installed or does not use debconf
debconf: package "3dchess" is not installed or does not use debconf
debconf: package "9menu" is not installed or does not use debconf
debconf: package "9wm" is not installed or does not use debconf
debconf: package "a2ps" is not installed or does not use debconf
debconf: package "a2ps-perl-ja" is not installed or does not use debconf
debconf: package "aalib-bin" is not installed or does not use debconf
```

This will produce a very long list of packages that do not use debconf, but it will also find some that do and present easy-to-use forms for you to answer the questions that each package asks.

Getting the status of an installed package

The Debian package management tools we've reviewed so far are best for handling multi-step operations with long lists of packages. But they don't cover some of the nuts-and-bolts operations of package management. For this kind of work, you want to use `dpkg`.

For example, to get the complete status and description of a package, use the `-s` option:

```
# dpkg -s xsnow
Package: xsnow
Status: install ok installed
Priority: optional
Section: non-free/x11
Installed-Size: 41
Maintainer: Martin Schulze <joey@debian.org>
Version: 1.40-6
Depends: libc6, xlib6g (>= 3.3-5)
Description: Brings Christmas to your desktop
 Xsnow is the X-windows application that will let it snow on the
 root window, in between and on windows.  Santa and his reindeer
 will complete your festive-season feeling.
```

The link between a file and its .deb

Since a `.deb` package contains files, you would think there would be a way to list the files within the package. Well, you would be right; just use the `-L` option:

```
# dpkg -L xsnow
/.
/usr
/usr/doc
/usr/doc/xsnow
/usr/doc/xsnow/copyright
/usr/doc/xsnow/readme.gz
/usr/doc/xsnow/changelog.Debian.gz
```

```
/usr/X11R6
/usr/X11R6/bin
/usr/X11R6/bin/xsnow
/usr/X11R6/man
/usr/X11R6/man/man6
/usr/X11R6/man/man6/xsnow.6.gz
```

To go the other way around, and find which package contains a specific file, use the `-S` option:

```
# dpkg -S /usr/doc/xsnow/copyright
xsnow: /usr/doc/xsnow/copyright
```

The name of the package is listed just to the left of the colon.

Finding packages to install

Usually, apt-get will already know about any Debian package you might need. If it doesn't, you may be able to find the package among these *lists of Debian packages*, or elsewhere on the Web.

If you do find and download a `.deb` file, you can install it using the `-i` option:

```
# dpkg -d /tmp/dl/xsnow_1.40-6_i386.deb
```

If you can't find the package you're looking for as a `.deb` file, but you find a `.rpm` or some other type of package, you may be able to use *alien*. The alien program can convert packages from various formats into `.debs`.

Additional Debian package management resources

There is a lot more to the Debian package management system than we covered here. There is also a lot more to Debian than its package management system. The following sites will help round out your knowledge in these areas:

- *Debian home page*
- *Debian installation guide*
- *Lists of Debian packages*
- *Alien home page*
- *Guide to creating your own Debian packages*

Section 7. Summary and resources

Summary

Congratulations, you've reached the end of this tutorial! We hope it has helped solidify your knowledge of compiling programs from sources, managing shared libraries, and using the Red Hat and Debian package management systems. Please join us in our next tutorial, entitled "Configuring and compiling the kernel," in which we show you how to modify and compile the Linux kernel from source code. By continuing in this tutorial series, you'll soon be ready to attain your LPIC Level 1 Certification from the Linux Professional Institute.

On the next page, you'll find a number of resources that you will find helpful in learning more about the subjects presented in this tutorial.

Resources

At *linuxdoc.org*, you'll find linuxdoc's collection of guides, HOWTOs, FAQs and man-pages to be invaluable. Be sure to check out *Linux Gazette* and *LinuxFocus* as well.

The Linux System 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*, Daniel shows you how to use `bash` programming constructs to write your own `bash` scripts. This bash series (particularly Parts 1 and 2) will be excellent additional preparation for the LPIC Level 1 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*

We highly recommend the *Technical FAQ for Linux users* by Mark Chapman, 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. We also recommend *Linux glossary for Linux users*, also from Mark.

If you're not familiar with the vi editor, we strongly recommend that you check out Daniel's *Vi intro -- the cheat sheet method tutorial* . This 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`.

Feedback

Please send any tutorial feedback you may have to the authors:

- Daniel Robbins, at *drobbins@gentoo.org*
- Chris Houser, at *chouser@gentoo.org*
- Aron Griffis, at *agriffis@gentoo.org*

LPI certification 101 exam prep (release 2), Part 3

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. Before you start

About this tutorial

Welcome to "Intermediate administration," the third of four tutorials designed to prepare you for the Linux Professional Institute's 101 (release 2) exam. This tutorial (Part 3) is ideal for those who want to improve their knowledge of fundamental Linux administration skills. We'll cover a variety of topics, including system and Internet documentation, the Linux permissions model, user account management, and login environment tuning.

If you are new to Linux, we recommend that you start with *Part 1* and *Part 2*. For some, 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 foundational Linux system administration skills.

By the end of this *series* of tutorials (eight in all covering the LPI 101 and 102 exams), you will have the knowledge you need to become a Linux Systems Administrator and will be ready to attain an LPIC Level 1 certification from the Linux Professional Institute if you so choose.

For those who have taken the *release 1 version* of this tutorial for reasons other than LPI exam preparation, you probably don't need to take this one. However, if you do plan to take the exams, you should strongly consider reading this revised tutorial.

About the authors

For technical questions about the content of this tutorial, contact the authors:

- Daniel Robbins, at *drobbins@gentoo.org*
- Chris Houser, at *chouser@gentoo.org*
- Aron Griffis, at *agriffis@gentoo.org*

Residing in Albuquerque, New Mexico, **Daniel Robbins** is the Chief Architect of *Gentoo Linux* an advanced ports-based Linux metadistribution. Besides writing articles, tutorials, and tips for the *developerWorks* Linux zone and Intel Developer Services, he has also served as a contributing author for several books, including *Samba Unleashed* and *SuSE Linux Unleashed*. Daniel enjoys spending time with his wife, Mary, and his daughter, Hadassah. You can contact Daniel at *drobbins@gentoo.org*.

Chris Houser, known to his friends as "Chouser," has been a UNIX proponent since 1994 when he joined the administration team for the computer science network at Taylor University in Indiana, where he earned his Bachelor's degree in Computer Science and Mathematics. Since then, he has gone on to work in Web application programming, user interface design, professional video software support, and now Tru64 UNIX device driver programming at *Compaq*. He has also contributed to various free software projects, most recently to *Gentoo Linux*). He lives with his wife and two cats in New Hampshire. You can contact Chris at *chouser@gentoo.org*.

Aron Griffis graduated from Taylor University with a degree in Computer Science and an award that proclaimed, "Future Founder of a Utopian UNIX Commune." Working towards that goal, Aron is employed by *Compaq* writing network drivers for Tru64 UNIX, and spending his

spare time plunking out tunes on the piano or developing *Gentoo Linux*. He lives with his wife Amy (also a UNIX engineer) in Nashua, New Hampshire.

Section 2. System and network documentation

Types of Linux system documentation

There are essentially three sources of documentation on a Linux system: manual pages, info pages, and application-bundled documentation in /usr/share/doc. In this section, we'll explore each of these sources before looking "outside the box" for more information.

Manual pages

Manual pages, or "man pages", are the classic form of UNIX and Linux reference documentation. Ideally, you can look up the man page for any command, configuration file, or library routine. In practice, Linux is free software, and some pages haven't been written or are showing their age. Nonetheless, man pages are the first place to look when you need help.

To access a man page, simply type `man` followed by your topic of inquiry. A pager will be started, so you will need to press **q** when you're done reading. For example, to look up information about the `ls` command, you would type:

```
$ man ls
```

Manual pages, continued

Knowing the layout of a man page can be helpful to jump quickly to the information you need. In general, you will find the following sections in a man page:

`NAME`	Name and one-line description of the command
`SYNOPSIS`	How to use the command
`DESCRIPTION`	In-depth discussion on the functionality of the command
`EXAMPLES`	Suggestions for how to use the command
`SEE ALSO`	Related topics (usually man pages)

man page sections

The files that comprise manual pages are stored in /usr/share/man (or in /usr/man on some older systems). Inside that directory, you will find that the manual pages are organized into the following sections:

`man1`	User programs
`man2`	System calls
`man3`	Library functions
`man4`	Special files

man5	File formats
man6	Games
man7	Miscellaneous

Multiple man pages

Some topics exist in more than one section. To demonstrate this, let's use the `whatis` command, which shows all the available man pages for a topic:

```
$ whatis printf
printf               (1)  - format and print data
printf               (3)  - formatted output conversion
```

In this case, `man printf` would default to the page in section 1 ("User Programs"). If we were writing a C program, we might be more interested in the page from section 3 ("Library functions"). You can call up a man page from a certain section by specifying it on the command line, so to ask for *printf(3)*, we would type:

```
$ man 3 printf
```

Finding the right man page

Sometimes it's hard to find the right man page for a given topic. In that case, you might try using `man -k` to search the **NAME** section of the man pages. Be warned that it's a substring search, so running something like `man -k ls` will give you a lot of output! Here's an example using a more specific query:

```
$ man -k whatis
apropos              (1)  - search the whatis database for strings
makewhatis           (8)  - Create the whatis database
whatis               (1)  - search the whatis database for complete words
```

All about apropos

The example on the previous panel brings up a few more points. First, the `apropos` command is exactly equivalent to `man -k`. (In fact, I'll let you in on a little secret. When you run `man -k`, it actually runs `apropos` behind the scenes.) The second point is the `makewhatis` command, which scans all the man pages on your Linux system and builds the database for `whatis` and `apropos`. Usually this is run periodically by root to keep the database updated:

```
# makewhatis
```

For more information on "man" and friends, you should start with its man page:

```
$ man man
```

The MANPATH

By default, the `man` program will look for man pages in /usr/share/man, /usr/local/man, /usr/X11R6/man, and possibly /opt/man. Sometimes, you may find that you need to add an additional item to this search path. If so, simply edit /etc/man.conf in a text editor and add a line that looks like this:

```
MANPATH /opt/man
```

From that point forward, any man pages in the /opt/man/man* directories will be found. Remember that you'll need to rerun `makewhatis` to add these new man pages to the whatis database.

GNU info

One shortcoming of man pages is that they don't support hypertext, so you can't jump easily from one to another. The GNU folks recognized this shortcoming, so they invented another documentation format: "info" pages. Many of the GNU programs come with extensive documentation in the form of info pages. You can start reading info pages with the `info` command:

```
$ info
```

Calling `info` in this way will bring up an index of the available pages on the system. You can move around with the arrow keys, follow links (indicated with a star) using the Enter key, and quit by pressing **q**. The keys are based on Emacs, so you should be able to navigate easily if you're familiar with that editor. For an intro to the Emacs editor, see the *developerWorks* tutorial, *Living in Emacs*.

GNU info, continued

You can also specify an info page on the command line:

```
$ info diff
```

For more information on using the `info` reader, try reading its info page. You should be able to navigate primitively using the few keys I've already mentioned:

```
$ info info
```

/usr/share/doc

There is a final source for help within your Linux system. Many programs are shipped with additional documentation in other formats: text, PDF, PostScript, HTML, to name a few. Take

a look in /usr/share/doc (or /usr/doc on older systems). You'll find a long list of directories, each of which came with a certain application on your system. Searching through this documentation can often reveal some gems that aren't available as man pages or info pages, such as tutorials or additional technical documentation. A quick check reveals there's a lot of reading material available:

```
$ cd /usr/share/doc
$ find . -type f | wc -l
   7582
```

Whew! Your homework this evening is to read just half (3791) of those documents. Expect a quiz tomorrow. ;-)

The Linux Documentation Project

In addition to system documentation, there are a number of excellent Linux resources on the Internet. The *Linux Documentation Project* (http://www.tldp.org) is a group of volunteers who are working on putting together the complete set of free Linux documentation. This project exists to consolidate various pieces of Linux documentation into a location that is easy to search and use.

An LDP overview

The LDP is made up of the following areas:

- Guides - longer, more in-depth books, such as *The Linux Programmer's Guide* (http://www.tldp.org/LDP/lpg/)
- HOWTOs - subject-specific help, such as the *DSL HOWTO* (http://www.tldp.org/HOWTO/DSL-HOWTO/)
- FAQs - Frequently Asked Questions with answers, such as the *Brief Linux FAQ* (http://www.tldp.org/FAQ/faqs/BLFAQ)
- man pages - help on individual commands (these are the same manual pages you get on your Linux system when you use the `man` command)

If you aren't sure which section to peruse, you can take advantage of the search box, which allows you to find things by topic.

The LDP additionally provides a list of Links and Resources such as *Linux Gazette* (see links in Resources on page 28) and *LinuxFocus*, as well links to mailing lists and news archives.

Mailing lists

Mailing lists provide probably the most important point of collaboration for Linux developers. Often projects are developed by contributors who live far apart, possibly even on opposite sides of the globe. Mailing lists provide a method for each developer on a project to contact all the others, and to hold group discussions via e-mail. One of the most famous

development mailing lists is the "*Linux Kernel Mailing List*" (http://www.tux.org/lkml/).

More about mailing lists

In addition to development, mailing lists can provide a method for asking questions and receiving answers from knowledgeable developers, or even other users. For example, individual distributions often provide mailing lists for newcomers. You can check your distribution's Web site for information on the mailing lists it provides.

If you took the time to read the LKML FAQ at the link on the previous panel, you might have noticed that mailing list subscribers often don't take kindly to questions being asked repeatedly. It's always wise to search the archives for a given mailing list before writing your question. Chances are, it will save you time, too!

Newsgroups

Internet "newsgroups" are similar to mailing lists, but are based on a protocol called NNTP ("Network News Transfer Protocol") instead of e-mail. To participate, you need to use an NNTP client such as **slrn** or **pan**. The primary advantage is that you only take part in the discussion when you want, instead of having it continually arrive in your inbox. :-)

The newsgroups of primary interest start with **comp.os.linux**. You can browse the *list on the LDP site* (http://www.tldp.org/linux/#ng).

As with mailing lists, newsgroup discussion is often archived. A popular newsgroup archiving site is *Deja News*.

Vendor and third-party Web sites

Web sites for the various Linux distributions often provide updated documentation, installation instructions, hardware compatibility/incompatibility statements, and other support such as a knowledge base search tool. For example:

- *Redhat Linux* (http://www.redhat.com)
- *Debian Linux* (http://www.debian.org)
- *Gentoo Linux* (http://www.gentoo.org)
- *SuSE Linux* (http://www.suse.com)
- *Caldera* (http://www.caldera.com)
- *Turbolinux* (http://www.turbolinux.com)

Linux consultancies

Some Linux consultancies, such as Linuxcare and Mission Critical Linux, provide some free documentation as well as pay-for support contracts. There are many Linux consultancies; below are a couple of the larger examples:

- *LinuxCare* (http://www.linuxcare.com)
- *Mission Critical Linux* (http://www.missioncriticallinux.com)

Hardware and software vendors

Many hardware and software vendors have added Linux support to their products in recent years. At their sites, you can find information about which hardware supports Linux, software development tools, released sources, downloads of Linux drivers for specific hardware, and other special Linux projects. For example:

- *IBM and Linux* (http://www.ibm.com/linux)
- *Compaq and Linux* (http://www.compaq.com/products/software/linux)
- *SGI and Linux* (http://www.sgi.com/developers/technology/linux)
- *HP and Linux* (http://www.hp.com/products1/linux)
- *Sun and Linux* (http://www.sun.com/linux)
- *Oracle and Linux* (http://technet.oracle.com/tech/linux/content.html)

Developer resources

In addition, many hardware and software vendors have developed wonderful resources for Linux developers and administrators. At the risk of sounding self-promoting, one of the most valuable Linux resources run by a hardware/software vendor is the *IBM developerWorks Linux zone* (http://www.ibm.com/developerworks/linux).

Section 3. The Linux permissions model

One user, one group

In this section, we'll take a look at the Linux permissions and ownership model. We've already seen that every file is owned by one user and one group. This is the very core of the permissions model in Linux. You can view the user and group of a file in a `ls -l` listing:

```
$ ls -l /bin/bash
-rwxr-xr-x    1 root     wheel      430540 Dec 23 18:27 /bin/bash
```

In this particular example, the `/bin/bash` executable is owned by `root` and is in the `wheel` group. The Linux permissions model works by allowing three independent levels of permission to be set for each filesystem object -- those for the file's owner, the file's group, and all other users.

Understanding "ls -l"

Let's take a look at our `ls -l` output and inspect the first column of the listing:

```
$ ls -l /bin/bash
-rwxr-xr-x    1 root     wheel      430540 Dec 23 18:27 /bin/bash
```

This first field `-rwxr-xr-x` contains a *symbolic* representation of this particular files' permissions. The first character (`-`) in this field specifies the *type* of this file, which in this case is a regular file. Other possible first characters:

```
'd' directory
'l' symbolic link
'c' character special device
'b' block special device
'p' fifo
's' socket
```

Three triplets

```
$ ls -l /bin/bash
-rwxr-xr-x    1 root     wheel      430540 Dec 23 18:27 /bin/bash
```

The rest of the field consists of *three* character triplets. The first triplet represents permissions for the owner of the file, the second represents permissions for the file's group, and the third represents permissions for all other users:

```
"rwx"
"r-x"
"r-x"
```

Above, the `r` means that reading (looking at the data in the file) is allowed, the `w` means that writing (modifying the file, as well as deletion) is allowed, and the `x` means that "execute"

(running the program) is allowed. Putting together all this information, we can see that everyone is able to read the contents of and execute this file, but only the owner (root) is allowed to modify this file in any way. So, while normal users can copy this file, only root is allowed to update it or delete it.

Who am I?

Before we take a look at how to change the user and group ownership of a file, let's first take a look at how to learn your current user id and group membership. Unless you've used the `su` command recently, your current user id is the one you used to log in to the system. If you use `su` frequently, however, you may not remember your current effective user id. To view it, type `whoami`:

```
# whoami
root
# su drobbins
$ whoami
drobbins
```

What groups am I in?

To see what groups you belong to, use the `group` command:

```
$ groups
drobbins wheel audio
```

In the above example, I'm a member of the `drobbins`, `wheel`, and audio groups. If you want to see what groups other user(s) are in, specify their usernames as arguments:

```
$ groups root daemon
root : root bin daemon sys adm disk wheel floppy dialout tape video
daemon : daemon bin adm
```

Changing user and group ownership

To change the owner or group of a file or other filesystem object, use `chown` or `chgrp`, respectively. Each of these commands takes a name followed by one or more filenames.

```
# chown root /etc/passwd
# chgrp wheel /etc/passwd
```

You can also set the owner and group simultaneously with an alternate form of the `chown` command:

```
# chown root.wheel /etc/passwd
```

You may not use `chown` unless you are the superuser, but `chgrp` can be used by anyone to change the group ownership of a file to a group to which they belong.

Recursive ownership changes

Both `chown` and `chgrp` have a `-R` option that can be used to tell them to recursively apply ownership and group changes to an entire directory tree. For example:

```
# chown -R drobbins /home/drobbins
```

Introducing chmod

`chown` and `chgrp` can be used to change the owner and group of a filesystem object, but another program -- called `chmod` -- is used to change the `rwx` permissions that we can see in an `ls -l` listing. `chmod` takes two or more arguments: a "mode", describing how the permissions should be changed, followed by a file or list of files that should be affected:

```
$ chmod +x scriptfile.sh
```

In the above example, our "mode" is `+x`. As you might guess, a `+x` mode tells `chmod` to make this particular file executable for both the user and group and for anyone else.

If we wanted to *remove* all execute permissions of a file, we'd do this:

```
$ chmod -x scriptfile.sh
```

User/group/other granularity

So far, our `chmod` examples have affected permissions for all three triplets -- the user, the group, and all others. Often, it's handy to modify only one or two triplets at a time. To do this, simply specify the symbolic character for the particular triplets you'd like to modify before the `+` or `-` sign. Use `u` for the "user" triplet, `g` for the "group" triplet, and `o` for the "other/everyone" triplet:

```
$ chmod go-w scriptfile.sh
```

We just removed write permissions for the group and all other users, but left "owner" permissions untouched.

Resetting permissions

In addition to flipping permission bits on and off, we can also reset them altogether. By using the `=` operator, we can tell `chmod` that we want the specified permissions and no others:

```
$ chmod =rx scriptfile.sh
```

Above, we just set all "read" and "execute" bits, and unset all "write" bits. If you just want to reset a particular triplet, you can specify the symbolic name for the triplet before the = as

follows:

```
$ chmod u=rx scriptfile.sh
```

Numeric modes

Up until now, we've used what are called *symbolic* modes to specify permission changes to `chmod`. However, there's another common way of specifying permissions: using a 4-digit octal number. Using this syntax, called *numeric permissions syntax*, each digit represents a permissions triplet. For example, in `1777`, the `777` sets the "owner", "group", and "other" flags that we've been discussing in this section. The `1` is used to set the special permissions bits, which we'll cover later (see "The elusive first digit on page 17 " at the end of this section). This chart shows how the second through fourth digits (`777`) are interpreted:

Mode	Digit
`rwx`	`7`
`rw-`	`6`
`r-x`	`5`
`r--`	`4`
`-wx`	`3`
`-w-`	`2`
`--x`	`1`
`---`	`0`

Numeric permission syntax

Numeric permission syntax is especially useful when you need to specify *all* permissions for a file, such as in the following example:

```
$ chmod 0755 scriptfile.sh
$ ls -l scriptfile.sh
-rwxr-xr-x    1 drobbins drobbins        0 Jan  9 17:44 scriptfile.sh
```

In this example, we used a mode of `0755`, which expands to a complete permissions setting of `-rwxr-xr-x`.

The umask

When a process creates a new file, it specifies the permissions that it would like the new file to have. Often, the mode requested is `0666` (readable and writable by everyone), which is more permissive that we would like. Fortunately, Linux consults something called a "umask" whenever a new file is created. The system uses the umask value to reduce the originally specified permissions to something more reasonable and secure. You can view your current

umask setting by typing `umask` at the command line:

```
$ umask
0022
```

On Linux systems, the umask normally defaults to `0022`, which allows others to read your new files (if they can get to them) but not modify them.

The umask, continued

To make new files more secure by default, you can change the umask setting:

```
$ umask 0077
```

This umask will make sure that the group and others will have absolutely no permissions for any newly created files. So, how does the umask work? Unlike "regular" permissions on files, the umask specifies which permissions should be turned *off*. Let's consult our mode-to-digit mapping table so that we can understand what a umask of `0077` means:

Mode	Digit
`rwx`	7
`rw-`	6
`r-x`	5
`r--`	4
`-wx`	3
`-w-`	2
`--x`	1
`---`	0

Using our table, the last three digits of `0077` expand to `---rwxrwx`. Now, remember that the umask tells the system which permissions to *disable*. Putting two and two together, we can see that all "group" and "other" permissions will be turned off, while "user" permissions will remain untouched.

Introducing suid and sgid

When you initially log in, a new shell process is started. You already know that, but you may not know that this new shell process (typically `bash`) runs using your user id. As such, the `bash` program can access all files and directories that you own. In fact, we as users are totally dependent on other programs to perform operations on our behalf. Because the programs you start inherit *your* user id, they cannot access any filesystem objects for which you haven't been granted access.

Introducing suid and sgid, continued

For example, the passwd file cannot be changed by normal users directly, because the "write" flag is off for every user except root:

```
$ ls -l /etc/passwd
-rw-r--r--    1 root     wheel        1355 Nov  1 21:16 /etc/passwd
```

However, normal users *do* need to be able to modify /etc/passwd (at least indirectly) whenever they need to change their password. But, if the user is unable to modify this file, how exactly does this work?

suid

Thankfully, the Linux permissions model has two special bits called `suid` and `sgid`. When an executable program has the `suid` bit set, it will run on behalf of the *owner* of the executable, rather than on behalf of the person who started the program.

Now, back to the /etc/passwd problem. If we take a look at the `passwd` executable, we can see that it's owned by root:

```
$ ls -l /usr/bin/passwd
-rwsr-xr-x    1 root     wheel       17588 Sep 24 00:53 /usr/bin/passwd
```

You'll also note that in place of an `x` in the user's permission triplet, there's an `s`. This indicates that, for this particular program, the `suid` and executable bits are set. Because of this, when `passwd` runs, it will execute on behalf of the root user (with full superuser access) rather than that of the user who ran it. And because `passwd` runs with `root` access, it's able to modify the /etc/passwd file with no problem.

suid/sgid caveats

We've seen how `suid` works, and `sgid` works in a similar way. It allows programs to inherit the group ownership of the program rather than that of the current user.

Here's some miscellaneous yet important information about `suid` and `sgid`. First, `suid` and `sgid` bits occupy the same space as the `x` bits in a `ls -l` listing. If the `x` bit is also set, the respective bits will show up as `s` (lowercase). However, if the `x` bit is not set, it will show up as a `S` (uppercase).

Another important note: `suid` and `sgid` come in handy in many circumstances, but improper use of these bits can allow the security of a system to be breached. It's best to have as few `suid` programs as possible. The `passwd` command is one of the few that must be `suid`.

Changing suid and sgid

Setting and removing the `suid` and `sgid` bits is fairly straightforward. Here, we set the `suid` bit:

```
# chmod u+s /usr/bin/myapp
```

And here, we *remove* the `sgid` bit from a directory. We'll see how the `sgid` bit affects directories in just a few panels:

```
# chmod g-s /home/drobbins
```

Permissions and directories

So far, we've been looking at permissions from the perspective of regular files. When it comes to directories, things are a bit different. Directories use the same permissions flags, but they are interpreted to mean slightly different things.

For a directory, if the "read" flag is set, you may *list* the contents of the directory; "write" means you may *create* files in the directory; and "execute" means you may *enter* the directory and access any sub-directories inside. Without the "execute" flag, the filesystem objects inside a directory aren't accessible. Without a "read" flag, the filesystem objects inside a directory aren't viewable, but objects inside the directory can still be accessed as long as someone knows the full path to the object on disk.

Directories and sgid

And, if a directory has the "sgid" flag enabled, any filesystem objects created inside it will inherit the group of the directory. This particular feature comes in handy when you need to create a directory tree to be used by a group of people that all belong to the same group. Simply do this:

```
# mkdir /home/groupspace
# chgrp mygroup /home/groupspace
# chmod g+s /home/groupspace
```

Now, any users in the group mygroup can create files or directories inside /home/groupspace, and they will be automatically assigned a group ownership of mygroup as well. Depending on the users' umask setting, new filesystem objects may or may not be readable, writable, or executable by other members of the mygroup group.

Directories and deletion

By default, Linux directories behave in a way that may not be ideal in all situations. Normally, anyone can rename or delete a file inside a directory, as long as they have *write* access to that directory. For directories used by individual users, this behavior is usually just fine.

However, for directories that are used by many users, especially /tmp and /var/tmp, this behavior can be bad news. Since *anyone* can write to these directories, *anyone* can delete or rename anyone else's files -- even if they don't own them! Obviously, it's hard to use /tmp for anything meaningful when any other user can type `rm -rf /tmp/*` at any time and destroy everyone's files.

Thankfully, Linux has something called the *sticky bit*. When /tmp has the sticky bit set (with a `chmod +t`), the only people who are able to delete or rename files in /tmp are the directory's owner (typically root), the file's owner, or root. Virtually all Linux distributions enable /tmp's sticky bit by default, but you may find that the sticky bit comes in handy in other situations.

The elusive first digit

And to conclude this section, we finally take a look at the elusive first digit of a numeric mode. As you can see, this first digit is used for setting the sticky, suid, and sgid bits:

suid	sgid	sticky	mode digit
on	**on**	**on**	**7**
on	**on**	**off**	**6**
on	**off**	**on**	**5**
on	**off**	**off**	**4**
off	**on**	**on**	**3**
off	**on**	**off**	**2**
off	**off**	**on**	**1**
off	**off**	**off**	**0**

Here's an example of how to use a 4-digit numeric mode to set permissions for a directory that will be used by a workgroup:

```
# chmod 1775 /home/groupfiles
```

As homework, figure out the meaning of the 1755 numeric permissions setting. :)

Section 4. Linux account management

Introducing /etc/passwd

In this section, we'll look at the Linux account management mechanism, starting with the /etc/passwd file, which defines all the users that exist on a Linux system. You can view your own /etc/passwd file by typing `less /etc/passwd`.

Each line in /etc/passwd defines a user account. Here's an example line from my /etc/passwd file:

```
drobbins:x:1000:1000:Daniel Robbins:/home/drobbins:/bin/bash
```

As you can see, there is quite a bit of information on this line. In fact, each /etc/passwd line consists of multiple fields, each separated by a `:`.

The first field defines the username (`drobbins`)), and the second field contains an `x`. On ancient Linux systems, this field contained an encrypted password to be used for authentication, but virtually all Linux systems now store this password information in another file.

The third field (`1000`) defines the numeric user id associated with this particular user, and the fourth field (`1000`) associates this user with a particular group; in a few panels, we'll see where group `1000` is defined.

The fifth field contains a textual description of this account -- in this case, the user's name. The sixth field defines this user's home directory, and the seventh field specifies the user's default shell -- the one that will be automatically started when this user logs in.

/etc/passwd tips and tricks

You've probably noticed that there are many more user accounts defined in /etc/passwd than actually log in to your system. This is because various Linux components use user accounts to enhance security. Typically, these system accounts have a user id ("uid") of under 100, and many of them will have something like /bin/false listed as a default shell. Since the /bin/false program does nothing but exit with an error code, this effectively prevents these accounts from being used as login accounts -- they are for internal use only.

/etc/shadow

So, user accounts themselves are defined in /etc/passwd. Linux systems contain a companion file to /etc/passwd that's called /etc/shadow. This file, unlike /etc/passwd, is readable only by root and contains encrypted password information. Let's look at a sample line from /etc/shadow:

```
drobbins:$1$12345678901234567890123456789O1:11664:0:-1:-1:-1:-1:0
```

Each line defines password information for a particular account, and again, each field is

separated by a `:`. The first field defines the particular user account with which this shadow entry is associated. The second field contains an encrypted password. The remaining fields are described in the following table:

field 3	# of days since 1/1/1970 that the password was modified
field 4	# of days before password will be allowed to be changed (0 for "change anytime")
field 5	# of days before system will force user to change to a new password (-1 for "never")
field 6	# of days before password expires that user will be warned about expiration (-1 for "no warning")
field 7	# of days after password expiration that this account is automatically disabled by the system (-1 for "never disable")
field 8	# of days that this account has been disabled (-1 for "this account is enabled")
field 9	Reserved for future use

/etc/group

Next, we take a look at the /etc/group file, which defines all the groups on a Linux system. Here's a sample line:

```
drobbins:x:1000:
```

The /etc/group field format is as follows. The first field defines the name of the group; the second field is a vestigial password field that now simply holds an `x`, and the third field defines the numeric group id of this particular group. The fourth field (empty in the above example) defines any users that are members of this group.

You'll recall that our sample /etc/passwd line referenced a group id of 1000. This has the effect of placing the drobbins user in the drobbins group, even though the drobbins username isn't listed in the fourth field of /etc/group.

Group notes

A note about associating users with groups: on some systems, you'll find that every new login account is associated with an identically named (and usually identically numbered) group. On other systems, all login accounts will belong to a single users group. The approach that you use on the system(s) you administrate is up to you. Creating matching groups for each user has the advantage of allowing users to more easily control access to their own files by placing trusted friends in their personal group.

Adding a user and group by hand

Now, I'll show you how to create your own user and group account. The best way to learn how to do this is to add a new user to the system *manually*. To begin, first make sure that your `EDITOR` environment variable is set to your favorite text editor:

```
# echo $EDITOR
vim
```

If it isn't, you can set `EDITOR` by typing something like:

```
# export EDITOR=/usr/bin/emacs
```

Now, type:

```
# vipw
```

You should now find yourself in your favorite text editor with the /etc/passwd file loaded up on the screen. When modifying system passwd and group files, it's very important to use the `vipw` and `vigr` commands. They take extra precautions to ensure that your critical passwd and group files are locked properly so they don't become corrupted.

Editing /etc/passwd

Now that you have the /etc/passwd file up, go ahead and add the following line:

```
testuser:x:3000:3000:LPI tutorial test user:/home/testuser:/bin/false
```

We've just added a "testuser" user with a UID of 3000. We've added him to a group with a GID of 3000, which we haven't created just yet. Alternatively, we could have assigned this user to the GID of the users group if we wanted. This new user has a comment that reads `LPI tutorial test user`; the user's home directory is set to /home/testuser, and the user's shell is set to /bin/false for security purposes. If we were creating an non-test account, we would set the shell to /bin/bash. OK, go ahead and save your changes and exit.

Editing /etc/shadow

Now, we need to add an entry in /etc/shadow for this particular user. To do this, type `vipw -s`. You'll be greeted with your favorite editor, which now contains the /etc/shadow file. Now, go ahead and *copy* the line of an existing user account (one that has a password and is longer than the standard system account entries):

```
drobbins:$1$12345678901234567890123456789O1:11664:0:-1:-1:-1:-1:0
```

Now, change the username on the copied line to the name of your new user, and ensure that all fields (particularly the password aging ones) are set to your liking:

```
testuser:$1$12345678901234567890123456789O1:11664:0:-1:-1:-1:-1:0
```

Now, save and exit.

Setting a password

You'll be back at the prompt. Now, it's time to set a password for your new user:

```
# passwd testuser
Enter new UNIX password: (enter a password for testuser)
Retype new UNIX password: (enter testuser's new password again)
```

Editing /etc/group

Now that /etc/passwd and /etc/shadow are set up, it's now time to get /etc/group configured properly. To do this, type:

```
# vigr
```

Your /etc/group file will appear in front of you, ready for editing. Now, if you chose to assign a default group of users for your particular test user, you do not need to add any groups to /etc/groups. However, if you chose to create a new group for this user, go ahead and add the following line:

```
testuser:x:3000:
```

Now save and exit.

Creating a home directory

We're nearly done. Type the following commands to create testuser's home directory:

```
# cd /home
# mkdir testuser
# chown testuser.testuser testuser
# chmod o-rwx testuser
```

Our user's home directory is now in place and the account is ready for use. Well, almost ready. If you'd like to use this account, you'll need to use `vipw` to change testuser's default shell to /bin/bash so that the user can log in.

Account admin utils

Now that you know how to add a new account and group by hand, let's review the various time-saving account administration utilities available under Linux. Due to space constraints, we won't cover a lot of detail describing these commands. Remember that you can always get more information about a command by viewing the command's man page. If you are planning to take the LPIC 101 exam, you should spend some time getting familiar with each of these commands.

newgrp

By default, any files that a user creates are assigned to the user's group specified in /etc/passwd. If the user belongs to other groups, he or she can type `newgrp thisgroup` to set current default group membership to the group thisgroup. Then, any new files created will inherit thisgroup membership.

chage

The `chage` command is used to view and change the password aging setting stored in /etc/shadow.

gpasswd

A general-purpose group administration tool.

groupadd/groupdel/groupmod

Used to add/delete/modify groups in /etc/group

More commands

useradd/userdel/usermod

Used to add/delete/modify users in /etc/passwd. These commands also perform various other convenience functions. See the man pages for more information.

pwconv/grpconv

Used to convert passwd and group files to "new-style" shadow passwords. Virtually all Linux systems already use shadow passwords, so you should never need to use these commands.

pwunconv/grpunconv

Used to convert passwd, shadow, and group files to "old-style" non-shadow passwords. You should never need to use these commands.

Section 5. Tuning the user environment

Introducing "fortune"

Your shell has many useful options that you can set to fit your personal preferences. So far, however, we haven't discussed any way to have these settings set up automatically every time you log in, except for re-typing them each time. In this section we will look at tuning your login environment by modifying startup files.

First, let's add a friendly message for when you first log in. To see an example message, run `fortune`:

```
$ fortune
No amount of careful planning will ever replace dumb luck.
```

.bash_profile

Now, let's set up `fortune` so that it gets run every time you log in. Use your favorite text editor to edit a file named .bash_profile in your home directory. If the file doesn't exist already, go ahead and create it. Insert a line at the top:

```
fortune
```

Try logging out and back in. Unless you're running a display manager like xdm, gdm, or kdm, you should be greeted cheerfully when you log in:

```
mycroft.flatmonk.org login: chouser
Password:
Freedom from incrustations of grime is contiguous to rectitude.
$
```

The login shell

When bash started, it walked through the .bash_profile file in your home directory, running each line as though it had been typed at a bash prompt. This is called *sourcing* the file.

Bash acts somewhat differently depending on how it is started. If it is started as a login shell, it will act as it did above -- first sourcing the system-wide /etc/profile, and then your personal ~/.bash_profile.

There are two ways to tell bash to run as a login shell. One way is used when you first log in: bash is started with a process name of -bash. You can see this in your process listing:

```
$ ps u
USER       PID %CPU %MEM   VSZ  RSS TTY      STAT START   TIME COMMAND
chouser    404  0.0  0.0  2508  156 tty2     S     2001   0:00 -bash
```

You will probably see a much longer listing, but you should have at least one **COMMAND**

with a dash before the name of your shell, like -bash in the example above. This dash is used by the shell to determine if it's being run as a login shell.

Understanding --login

The second way to tell bash to run as a login shell is with the `--login` command-line option. This is sometimes used by terminal emulators (like xterm) to make their bash sessions act like initial login sessions.

After you have logged in, more copies of your shell will be run. Unless they are started with `--login` or have a dash in the process name, these sessions will not be login shells. If they give you a prompt, however, they are called *interactive* shells. If bash is started as interactive, but not login, it will ignore /etc/profile and ~/.bash_profile and will instead source ~/.bashrc.

interactive	login	profile	rc
yes	yes	source	ignore
yes	no	ignore	source
no	yes	source	ignore
no	no	ignore	ignore

Testing for interactivity

Sometimes bash sources your ~/.bashrc, even though it isn't really interactive, such as when using commands like `rsh` and `scp`. This is important to keep in mind because printing out text, like we did with the `fortune` command earlier, can really mess up these non-interactive bash sessions. It's a good idea to use the `PS1` variable to detect whether the current shell is truly interactive before printing text from a startup file:

```
if [ -n "$PS1" ]; then
fortune
fi
```

/etc/profile and /etc/skel

As a system administrator, you are in charge of /etc/profile. Since it is sourced by everyone when they first log in, it is important to keep it in working order. It is also a powerful tool in making things work correctly for new users as soon as they log into their new account.

However, there are some settings that you may want new users to have as defaults, but also allow them to change easily. This is where the /etc/skel directory comes in. When you use the `useradd` command to create a new user account, it copies all the files from /etc/skel into the user's new home directory. That means you can put helpful .bash_profile and .bashrc files in /etc/skel to get new users off to a good start.

export

Variables in bash can be marked so that they are set the same in any new shells that it starts; this is called being marked for *export*. You can have bash list all of the variables that are currently marked for export in your shell session:

```
$ export
declare -x EDITOR="vim"
declare -x HOME="/home/chouser"
declare -x MAIL="/var/spool/mail/chouser"
declare -x PAGER="/usr/bin/less"
declare -x PATH="/bin:/usr/bin:/usr/local/bin:/home/chouser/bin"
declare -x PWD="/home/chouser"
declare -x TERM="xterm"
declare -x USER="chouser"
```

Marking variables for export

If a variable is not marked for export, any new shells that it starts will not have that variable set. However, you can mark a variable for export by passing it to the `export` built-in:

```
$ FOO=foo
$ BAR=bar
$ export BAR
$ echo $FOO $BAR
foo bar
$ bash
$ echo $FOO $BAR
bar
```

In this example, the variables FOO and BAR were both set, but only BAR was marked for export. When a new bash was started, it had lost the value for FOO. If you exit this new bash, you can see that the original one still has values for both FOO and BAR:

```
$ exit
$ echo $FOO $BAR
foo bar
```

Export and set -x

Because of this behavior, variables can be set in ~/.bash_profile or /etc/profile and marked for export, and then never need to be set again. There are some options that cannot be exported, however, and so they must be put in put in your ~/.bashrc **and** your profile in order to be set consistently. These options are adjusted with the `set` built-in:

```
$ set -x
```

The `-x` option causes bash to print out each command it is about to run:

```
$ echo $FOO
```

```
+ echo foo
foo
```

This can be very useful for understanding unexpected quoting behavior or similar strangeness. To turn off the `-x` option, do `set +x`. See the bash man page for all of the options to the `set` built-in.

Setting variables with "set"

The `set` built-in can also be used for setting variables, but when used that way, it is optional. The bash command `set FOO=foo` means exactly the same as `FOO=foo`. Un-setting a variable is done with the `unset` built-in:

```
$ FOO=bar
$ echo $FOO
bar
$ unset FOO
$ echo $FOO
```

Unset vs. FOO=

This is **not** the same as setting a variable to nothing, although it is sometimes hard to tell the difference. One way to tell is to use the `set` built-in with no parameters to list all current variables:

```
$ FOO=bar
$ set | grep ^FOO
FOO=bar
$ FOO=
$ set | grep ^FOO
FOO=
$ unset FOO
$ set | grep ^FOO
```

Using `set` with no parameters like this is similar to using the `export` built-in, except that `set` lists all variables instead of just those marked for export.

Exporting to change command behavior

Often, the behavior of commands can be altered by setting environment variables. Just as with new bash sessions, other programs that are started from your bash prompt will only be able to see variables that are marked for export. For example, the command `man` checks the variable PAGER to see what program to use to step through the text one page at a time.

```
$ PAGER=less
$ export PAGER
$ man man
```

With PAGER set to `less`, you will see one page at a time, and pressing the space bar moves on to the next page. If you change PAGER to `cat`, the text will be displayed all at once,

without stopping.

```
$ PAGER=cat
$ man man
```

Using "env"

Unfortunately, if you forget to set PAGER back to `less`, `man` (as well as some other commands) will continue to display all their text without stopping. If you wanted to have PAGER set to `cat` just once, you could use the `env` command:

```
$ PAGER=less
$ env PAGER=cat man man
$ echo $PAGER
less
```

This time, PAGER was exported to `man` with a value of `cat`, but the PAGER variable itself remained unchanged in the bash session.

Section 6. Summary and resources

Summary

Congratulations on finishing Part 3 of this tutorial series! At this point, you should know how to locate information in system and Internet documentation, and you should have a good grasp of the Linux permissions model, user account management, and login environment tuning.

Resources

Be sure to check out the various Linux documentation resources covered in this tutorial -- particularly the *Linux Documentation Project* (http://www.tldp.org). You'll find its collection of guides, HOWTOs, FAQs, and man pages to be invaluable. Be sure to check out *Linux Gazette* (http://www.tldp.org/LDP/LG/current/) and *LinuxFocus* (http://www.tldp.org/LDP/LG/current/) as well.

The *Linux System Administrators guide* (available from the "Guides" section at www.tldp.org) 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* (http://www.tldp.org/HOWTO/Unix-and-Internet-Fundamentals-HOWTO/) to be helpful.

You can read the GNU Project's online documentation for the GNU info system (also called "texinfo") at *GNU's texinfo documentation page* (http://www.gnu.org/manual/texinfo/).

One of the most famous development mailing lists is the "*Linux Kernel Mailing List*" (http://www.tux.org/lkml/).

Browse the *Linux newsgroup list* on the LDP site (http://www.tldp.org/linux/#ng), and the newsgroup archives at *Deja News* (http://groups.google.com/googlegroups/deja_announcement.html).

In the *Bash by example* article series on *developerWorks*, Daniel shows you how to use bash programming constructs to write your own bash scripts. This bash series (particularly Parts 1 and 2) is good preparation for the LPIC Level 1 exam and reinforces the concepts covered in this tutorial's "Tuning the user environment" section:

- *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*

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

If you're not familiar with the vi editor, check out Daniel's *Vi intro -- the cheat sheet method tutorial*. This 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 an intro to the Emacs editor, see the *developerWorks* tutorial, *Living in Emacs*.

Feedback

Please let us know whether this tutorial was helpful to you and how we could make it better. We'd also like to hear about other tutorial topics you'd like to see covered in *developerWorks* tutorials.

For questions about the content of this tutorial, contact the authors:

- Daniel Robbins, at *drobbins@gentoo.org*
- Chris Houser, at *chouser@gentoo.org*
- Aron Griffis, at *agriffis@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.

LPI certification 102 (release 2) 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. Before you start

About this tutorial

Welcome to "USB, secure shell, and file sharing," the last of four tutorials designed to prepare you for the Linux Professional Institute's 102 exam. In this tutorial, we'll introduce you to the ins and outs of using USB devices, how to use the secure shell (ssh) and related tools, and how to use and configure Network File System (NFS) version 3 servers and clients.

This tutorial is ideal for those who want to learn about or improve their foundational Linux USB, networking, and file sharing skills. It is particularly appropriate for those who will be setting up applications or USB hardware 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 you start with *Part 1* and work through the series from there.

By studying this series of tutorials (eight in all for the 101 and 102 exams; this is the eighth and last installment), 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.

For those who have taken the *release 1 version* of this tutorial for reasons other than LPI exam preparation, you probably don't need to take this one. However, if you do plan to take the exams, you should strongly consider reading this revised tutorial.

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

About the authors

For technical questions about the content of this tutorial, contact the authors:

- Daniel Robbins, at *drobbins@gentoo.org*
- John Davis, at *zhen@gentoo.org*

Daniel Robbins lives in Albuquerque, New Mexico, and is the Chief Architect 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.

John Davis lives in Cleveland, Ohio, and is the Senior Documentation Coordinator for *Gentoo Linux*, as well as a full-time computer science student at Mount Union College. Ever since his first dose of Linux at age 11, John has become a religious follower and has not looked back. When he is not writing, coding, or doing the college "thing," John can be found mountain biking or spending time with his family and close friends.

Section 2. USB devices and Linux

USB preliminaries

USB, or Universal Serial Bus, is a means of attaching devices and peripherals to your computer using cute little rectangular plugs. Commonly used by keyboards, mice, printers, and scanners, USB devices come in all shapes and sizes. One thing is certain: USB devices have arrived and it's essential to be able to get them running under Linux.

Setting up USB under GNU/Linux has always been a fairly easy, but undocumented, task. Users are often confused about whether or not to use modules, what the difference between UHCI, OHCI, and EHCI is, and why in the world their specific USB device is not working. This section should help clarify the different aspects of the Linux USB system.

This section assumes that you are familiar with how to compile your kernel, as well as the basic operation of a GNU/Linux system. For more information on these subjects, please visit the other LPI tutorials in this series, starting with *Part 1*, or The Linux Documentation Project homepage (see the Resources on page 17 at the end of this tutorial for links).

Modular vs. monolithic USB

Kernel support for USB devices can be configured in two ways: as modules or statically compiled into the kernel. Modular design allows for a smaller kernel size and quicker boot times. Statically compiled support allows for boot-time detection of devices and takes away the fuss of module dependencies. Both of these methods have their pros and cons, but the use of modules is suggested because they are easier to troubleshoot. Later, when everything is working, you may statically compile your USB device modules for convenience.

Grab a kernel

If you do not already have Linux kernel sources installed on your system, it is recommended that you download the latest 2.4 series kernel from kernel.org or one of its many mirrors (see the Resources on page 17 for a link).

Look at your hardware

Before compiling support for anything USB into your kernel, it is a good idea to find out what kind of hardware your computer is running. A simple, very useful set of tools called **pciutils** will get you on the right track. If you don't already have pciutils installed on your system, the sources for the most recent pciutils can be found at its homepage, which is listed in the Resources on page 17 .

Enter lspci

Running `lspci` should produce output similar to this:

```
# lspci
00:00.0 Host bridge: Advanced Micro Devices [AMD] AMD-760 [IGD4-1P] System Controller (r
00:01.0 PCI bridge: Advanced Micro Devices [AMD] AMD-760 [IGD4-1P] AGP Bridge
00:07.0 ISA bridge: VIA Technologies, Inc. VT82C686 [Apollo Super South] (rev 40)
00:07.1 IDE interface: VIA Technologies, Inc. VT82C586A/B/VT82C686/A/B/VT8233/A/C/VT8235
00:07.2 USB Controller: VIA Technologies, Inc. USB (rev 1a)
00:07.3 USB Controller: VIA Technologies, Inc. USB (rev 1a)
00:07.4 SMBus: VIA Technologies, Inc. VT82C686 [Apollo Super ACPI] (rev 40)
00:08.0 Serial controller: US Robotics/3Com 56K FaxModem Model 5610 (rev 01)
00:0b.0 VGA compatible controller: nVidia Corporation NV11DDR [GeForce2 MX 100 DDR/200 D
00:0d.0 Ethernet controller: 3Com Corporation 3c905C-TX/TX-M [Tornado] (rev 78)
00:0f.0 Multimedia audio controller: Creative Labs SB Live! EMU10k1 (rev 08)
00:0f.1 Input device controller: Creative Labs SB Live! MIDI/Game Port (rev 08)
01:05.0 VGA compatible controller: nVidia Corporation NV25 [GeForce4 Ti 4400] (rev a2)
```

Enable the right host controller

As you can see, `lspci` gives a complete listing of all PCI/PCI Bus Masters that your computer uses. The lines that are highlighted should be similar to what you are looking for in your lspci readout. Since the example controller is a VIA type controller, it would use the UHCI USB driver. For other chipsets, you would pick from one of these choices:

Driver	Chipset
EHCI	USB 2.0 Support
UHCI	All Intel, all VIA chipsets
JE (Alternate to UHCI	If UHCI does not work, and you have an Intel or VIA chipset, try JE
OHCI	Compaq, most PowerMacs, iMacs, and PowerBooks, OPTi, SiS, ALi

```
# cd /usr/src/linux
# make menuconfig
# make modules && make modules_install
```

Those cute USB modules

When the building completes, load the modules with `modprobe`, and your USB system will be ready to use.

The following line loads core USB support:

```
# modprobe usbcore
```

If you are using an EHCI controller, execute this line:

```
# modprobe ehci-hcd
```

If you are using an UHCI controller, execute this line:

```
# modprobe usb-uhci
```

If you are using a JE controller, execute this line:

```
# modprobe uhci
```

If you are using an OHCI controller, execute this line:

```
# modprobe usb-ohci
```

USB peripherals -- mice

Perhaps the most commonly used USB device is a USB mouse. Not only is it easy to install, but it offers plug-and-play flexibility for laptop users who would rather not use the homicide-inducing trackpad.

Before you can start using your USB mouse, you need to compile USB mouse support into your kernel. Enable these two options:

Menuconfig location	Option	Reason
Input Core Support	Mouse Support *(Don't forget to input your screen resolution!)*	Enabling this will hone your mouse tracking to your resolution, which makes mousing across large resolutions much nicer.
USB Support/ USB Human Interface Devices (HID)	USB HIDPB Mouse (basic) Support	Since your USB mouse is a USB HID (Human Interface Device), choosing this option will en the necessary HID subsystems.

USB mice, continued

Now, compile your new USB mouse-related modules:

```
# cd /usr/src/linux
# make menuconfig
# make modules && make modules_install
```

Once these options are compiled as modules, you are ready to load the `usbmouse` module and proceed:

```
# modprobe usbmouse
```

When the module finishes loading, go ahead and plug in your USB mouse. If you already had the mouse plugged in while the machine was booting, no worries, as it will still work.

Once you plug in the mouse, use `dmesg` to see if it was detected by the kernel:

```
# dmesg
hub.c: new USB device 10:19.0-1, assigned address 2
input4: USB HID v0.01 Mouse [Microsoft Microsoft IntelliMouse Optical] on usb2:2.0
```

When you have confirmed that your mouse is recognized by the kernel, it is time to configure XFree86 to use it. That's next.

USB mice and Xfree86

If you already have XFree86 installed and running with a non-USB mouse, not much configuration change is needed to use it. The only item that you need to change is what device XFree86 uses for your mouse. For our purposes, we will be using the /dev/input/mice device, since it supports hotplugging of your mouse, which can be very handy for desktop and laptop users alike. Your XF86Config file's "InputDevice" section should look similar to this:

```
Section "InputDevice"
   Identifier            "Mouse0"
   Driver               "mouse"
   Option        "Protocol"      "IMPS/2"
   #The next line enables mouse wheel support
   Option             "ZAxisMapping"  "4 5"
   #The next line points XFree86 to the USB mouse device
   Option             "Device"        "/dev/input/mice"
EndSection
```

Now, restart XFree86, and your USB mouse should be working just fine. Once everything is working, go ahead and compile your USB modules into the kernel statically. Of course, this is completely optional, so if you would like to keep your modules as modules, make sure they are loaded at boot time so that you can use your mouse after you reboot.

Configuring a USB digital camera

Yet another great feature in GNU/Linux is its digital imaging support. Powerful photo editing programs, such as the GIMP (see the Resources on page 17 for a link), make digital photography come alive.

Before any digital picture editing can take place, you'll need to retrieve the pictures that are going to be edited. Many times, digital cameras will have a USB port, but if yours does not, these instructions will work for your media card reader as long as the file system on your media card is supported in the Linux kernel.

USB Mass Storage works for anything that uses USB to access an internal drive of some sort. Feel free to experiment with other hardware, such as USB MP3 players, as these instructions will work the same. Additionally, note that older cameras with built-in serial ports are not compatible with these instructions.

USB storage -- the modules

Most USB Mass Storage devices use SCSI emulation so that they can be accessed by the

Linux kernel. Therefore, kernel support must be enabled for SCSI support, SCSI disk support, SCSI generic support, and USB Mass Storage support.

```
# cd /usr/src/linux
# make menuconfig
```

Enable the following options:

Menuconfig location	Option	Reason
SCSI support	SCSI support	Enables basic SCSI support
SCSI support	SCSI disk support	Enables support for SCSI disks
SCSI support	SCSI generic support	Enables support for generic SCSI devices, as well as some emulation
USB support/ USB Device Class drivers	USB Mass Storage support	Enables basic USB Mass Storage support; be sure to enable the options listed below it if you need support for any of that hardware

Build the USB storage modules

Since the options in the previous screen were compiled into your kernel as modules, there is no need to rebuild your kernel or reboot your computer! We just need to remake modules, and then load the newly compiled modules using `modprobe`.

```
# make modules && make modules_install
```

Please note that your third-party modules, such as NVIDIA drivers and ALSA drivers, may be overwritten by the module installation. You might want to reinstall those right after running `make modules_install`.

Did it work?

Once your modules are rebuilt, plug in your camera or media card reader and load the USB Mass Storage module:

The following line loads the SCSI disk support module:

```
# modprobe sd_mod
```

The following line loads the USB Mass Storage support module:

```
# modprobe usb-storage
```

Running `dmesg` should produce output similar to this:

```
# dmesg
Initializing USB Mass Storage driver...
usb.c: registered new driver usb-storage
scsi1 : SCSI emulation for USB Mass Storage devices
  Vendor: SanDisk   Model: ImageMate CF-SM   Rev: 0100
  Type:   Direct-Access                      ANSI SCSI revision: 02
  Vendor: SanDisk   Model: ImageMate CF-SM   Rev: 0100
  Type:   Direct-Access                      ANSI SCSI revision: 02
WARNING: USB Mass Storage data integrity not assured
USB Mass Storage device found at 2
USB Mass Storage support registered.
```

USB storage is go!

Congratulations! If you see something like the output in the previous screen, you're in business. All you have left to do is mount the camera or media card reader, and you can directly access your pictures.

On our machine, the card reader was mapped to /dev/sda1; yours might be different.

To mount your device, do the following (and note that your media's file system might not be vfat, so substitute as needed):

```
# mkdir /mnt/usb-storage
# mount -t vfat /dev/sda1 /mnt/usb-storage
```

Section 3. 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. Why? 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 see! Obviously, tools such as telnet were designed with the assumption that the network was secure and unsniffable and 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 `ssh`. A 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 very difficult or impossible 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-around" 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 Internet popularity, `ssh` is a valuable tool for enhancing network security when using Linux systems. Most security-savvy network admins discourage or disallow the use of `telnet` and `rsh` on their systems 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@remotebox
```

Above, you instruct `ssh` to log in as the "drobbins" user account on remotebox. Like `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 your distribution's openssh package by 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` (secure copy). You can use this command to securely copy files to and from various systems on the network. For example, if you wanted to copy ~/foo.txt to our home directory on remotebox, you could type:

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

Note the trailing colon -- without it, `scp` would have created a local file in the current working directory called "drobbins@remotebox." However, with the colon, the intended action is taken. After being prompted for the password on remotebox, the copy will be performed.

If you wanted to copy a file called bar.txt in remotebox's /tmp directory to the current working directory on our local system, you could type:

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

Again, the ever-important colon separates the user and host name from the file path.

Secure shell authentication options

Openssh also has a number of other authentication methods. Used properly, they can let you 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 Daniel's openssh key management articles on *developerWorks* (see the links in the Resources on page 17 at the end of this tutorial).

Section 4. 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 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 file system trees and have them fully integrated into a system's local file system. NFS' transparency and maturity make it 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" file systems. When a file system is exported, it is made available for access by other systems on the LAN. Then, any authorized system that is also set up as an NFS client can mount this exported file system using the standard `mount` command. After the mount completes, the remote file system is "grafted in" in the same way that a locally mounted file system (such as /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 not an issue to any standard Linux application. Everything simply works.

Attributes of NFS

Shared NFS file systems have a number of interesting attributes. The first 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 dependent on the amount of NFS data being transferred over the network, rather than on the number of machines that happen to be requesting that data.

NFS version 3 under Linux

When you set up NFS, we recommend 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 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 able to bypass 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 3* of the 102 series for more on this).

NFS users and groups

When setting up NFS, it's important to ensure that all NFS machines (both servers and clients) have identical user and group IDs in their user and group databases. Why? Because NFS clients and servers use numeric user and group IDs internally and assume that the IDs correspond to the same users and groups on all NFS-enabled machines.

Because of this, using mismatched user and group IDs with NFS can result in security breaches -- particularly if two different users on different systems happen to be sharing the same numerical UID.

So, before getting NFS set up on a larger LAN, it's a good idea to first set up NIS or NIS+. NIS(+), which stands for "Network Information Service," allows you to have a user and group database that can be centrally managed and shared throughout your entire LAN, ensuring NFS ownership consistency as well as reducing administrative headaches.

NIS and NFS combined

When NIS+ and NFS are combined, you can configure Linux systems on your network so that your users can log in to any box on your LAN -- and access their home directories and files from that box. NIS+ provides the shared user database that allows a user to log in anywhere, and NFS delivers the data. Both technologies work very well together.

While NIS+ is important, we don't have room to cover it in this tutorial. If you are planning to take the LPI exams -- or want to be able to use NFS to its full potential -- be sure to study the "Linux NFS HOWTO" by Thorsten Kukuk (see the Resources on page 17 for a link).

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 our 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 file systems that will be made available for export as well as:

- What hosts will be able to access these file systems
- Whether they will be exported as read/write or read-only
- Other options that control NFS behavior

But before we look at the format of the /etc/exports file, a big implementation warning is needed! The NFS implementation in the Linux kernel only allows the export of *one local directory per file system*. This means that if both /usr and /home are on the same ext3 file system (on /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 error 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 file system, 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 file system. Now that you understand this implementation quirk of Linux NFS, let's 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 we use on our 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 the /etc/exports file is a comment. On the second line, we select our root ("/") file system for export. Note that while this exports everything under "/", it

will not export any other local file system. For example, if our 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 our /etc/exports file. On this line, we export /mnt/backup; as you might guess, /mnt/backup is on a separate file system from /, and it contains a backup of our 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 file systems writeable as well as readable by NFS client systems ("rw",) and instructs the NFS server to allow the remote NFS client to allow a superuser account to have true "root" access to the file systems ("no_root_squash".)

Another /etc/exports file

Here's an /etc/exports that will export the same file systems as the one in the previous panel, except that it will make our exports available to all machines on our 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 the above example /etc/exports file, we use a host mask of /24 to mask out the last eight bits in the IP address we specify. It's 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 you 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 you can use to start NFS -- type `/etc/init.d/nfs start` or `/etc/rc.d/init.d/nfs start` to use it -- or use "restart" instead of "start" if your NFS server was already started at boot-time. 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 file systems.

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 file systems

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 file system 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, you can type:

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

Inventor's root file system 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 file system, 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 file system 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. Summary and resources

Summary

This wraps up this tutorial and the LPI 102 series. We hope you've enjoyed the ride! You should now be well versed on the use of ssh, NFS, and USB. To expand your Linux knowledge even further, see the Resources on page 17 on the next panel.

Resources

Although the tutorial is over, learning never ends and we recommend you check out the following resources, particularly if you plan to take the LPI 102 exam:

For more information on USB under GNU/Linux, please check out the *official Linux USB project page* for more information.

If you do not have *pciutils* installed on your system, you can find the source at the *pciutils project homepage*.

Get more information on *XFree86 configuration* at Xfree86.org.

Visit the *project homepage for the venerable GIMP*, or GNU Image Manipulation Program.

Daniel's OpenSSH key management series of articles on *developerWorks* is a great way to gain a deeper understanding of the security features provided by OpenSSH:

- *Part 1 on RSA/DSA authentication*
- *Part 2 on ssh-agent and keychain*
- *Part 3 on agent forwarding and keychain improvements*

Also be sure to visit the *home of openssh*, which is an excellent place to continue your study of this important tool.

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 quite good *Linux NFS HOWTO*, by Thorsten Kukuk.

We didn't have room to cover another important networked file-sharing technology: Samba. For more information about Samba, we recommend that you read Daniel's Samba articles on *developerWorks*:

- *Part 1 on key concepts*
- *Part 2 on compiling and installing Samba*
- *Part 3 on Samba configuration*

Once you're up to speed on Samba, we recommend that you spend some time studying the *Linux DNS HOWTO*. The LPI 102 exam is also going to expect that you have some familiarity with Sendmail. We didn't have enough room to cover Sendmail, but (fortunately for us!) Red Hat has a good *Sendmail HOWTO* that will help to get you up to speed.

In addition, we recommend the following general resources for learning more about Linux and preparing for LPI certification in particular:

Linux kernels and more can be found at the *Linux Kernel Archives*.

You'll find a wealth of guides, HOWTOs, FAQs, and man pages at *The Linux Documentation Project*. Be sure to check out *Linux Gazette* and *LinuxFocus* as well.

The Linux System 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:

- *Part 1 on fundamental programming in the Bourne-again shell*
- *Part 2 on more bash programming fundamentals*
- *Part 3 on 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 excellent as well.

If you're not very 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*.

Feedback

Please send any tutorial feedback you may have to the authors:

- Daniel Robbins, at *drobbins@gentoo.org*
- John Davis, at *zhen@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.

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.