6.4.3 Shared Libraries Facts

The importance of shared libraries is best understood under the context of software installations.

This lesson covers the following topics:

- Software repository and package manager review
- Local repositories
- Installing locally compiled software
- Shared libraries
- Potential shared library complications
- Library management commands

Software and Repository and Package Manager Review

Installing software on a Linux system usually involves software repositories, software packages and package managers. Linux distributions are similar in that all of them originate from open source software. However, each distribution will compile its own set of software applications and software libraries. A Linux application complied for one distribution will likely not run on another distribution. Therefore, each distribution hosts their own software repository containing their own software packages.

A Linux software repository is a location, usually on the internet, that contains software packages.

- Each Linux distribution maintains its own software repository.
- A Linux may maintain a repository in multiple locations.
  - Each location contains exact copies of packages on other locations.
  - An alternate location can be used if a chosen location is busy.
- Each package in the repository contains software that has been compiled for the specific Linux distribution.
- A software package may contain a software application or a software library which is routines that can be called by software applications.
- Most software packages will one of two types.
  - Debian packages have the *.deb extension and are used by Debian and Ubuntu distributions.
  - RPM packages have the *.rpm extension and are used by Red Hat, Fedora and other distributions.
- Software packages may be compressed and given the extension of *.tar, *.tgz, or *.gz.

A package manager is a software utility that downloads and installs software packages from a software repository.

- The package manager checks version numbers for compatibility.
- Package managers like Red Hat Package Manager (rpm) and Debian Package (dpkg) install downloaded packages without checking for dependencies.
- Package managers like Yellowdog Updater, Modified (YUM) and Advanced Package Tool (APT) check for package dependencies, then download and install all interrelated packages.

Local Repositories

You can configure a package manager to access packages from a local repository. Packages are more quickly and easily transferred from a local repository than from an internet repository. To create and configure a local repository:

- Create a directory that can be accessed by your Linux systems.
- Populated the directory by pulling packages from an online repository using the **rsync**, **wget**, or **curl** utility.
- Add the local repository to the package managers list of repositories.

Installing Locally Compiled Software

Some Linux software may not have a compiled package for your Linux distribution, but may offer the source code. Other Linux software may only be offered as source code. In these cases, you can download and compile the software as an executable. The executable can be installed so that your Linux system will know how to run it. The basic steps for installing this software are:

- Download the source code.
- Compile and build the source code.
  - The **gcc** compiler is often used to create an executable.
  - The **make** utility can automatically determine which pieces of a large program needs to be compiled or re-compiled.
- Use the **make install** utility to install the software by copying the built program along with its libraries and documentation to the correct locations.

Shared Libraries

Shared libraries are precompiled code elements that are loaded into memory and can be reused by many different programs running on the same system. This allows the size of each individual program to be relatively small, as it can use a shared library code when necessary. In addition, shared libraries make updates easier. Instead of updating all applications when a change needs to be made, you only need to update the appropriate shared library. Updating the shared libraries allows each applicable program to immediately take advantage of the update.

There a two types of shared libraries:

Type	Description
Dynamic	*Dynamic libraries* are not directly integrated into the code of the application that uses it. Dynamic libraries: ▪ Are linked to the application that shares its code. ▪ Have a **.so** or **.so.*version*** extension (**.so** stands for shared object). ▪ Are typically stored in **/usr/lib/** and **/usr/local/lib/**. ▪ Can degrade program load time if the library is already in use by another program. ▪ Are similar to Dynamic Link Libraries (DLLs) in Windows. Be aware of the following management programs and files for dynamic libraries: ▪ **/lib/ld.so** is a program which finds and loads the shared libraries needed by a program. It also prepares the program to run and executes it. ▪ **/etc/ld.so.conf** is a file which contains a list of directories in which to search for shared libraries. Some lines in the file begin with the **include** directive which list files that are to be included as if they were part of the main file. ▪ **/etc/ld.so.cache** is a cached list of libraries found in the directories specified in **/etc/ld.so.conf**. The system uses this cached list instead of loading **/etc/ld.so.conf** every time a program runs. Use the following methods for configuring dynamic libraries on a Linux system: ▪ Modify **/etc/ld.so.conf** to add the path of the libraries. ▪ Use the **LD_LIBRARY_PATH** environment variable to specify additional directories to search for library files.
Static	*Static libraries* are integrated into the code of the application itself when the code is compiled. Static libraries: ▪ Have an **.a** filename extension. ▪ Are used when dynamic libraries are not available. ▪ Increase the size of the application. ▪ Eliminate dependency issues associated with dynamic libraries.

Potential Shared Library Complications

Using shared libraries can create the following software management complications:

- Changes to the shared library could potentially be incompatible with some of the programs that use the library.
- Programs may not be able to locate the shared library.
- A shared library can become inaccessible if it is overwritten or updated. For instance, problems may occur if two different packages that include the same shared library are installed.

Library Management Commands

Be aware of the following library management commands:

Command	Function
ldd	Discovers which libraries are used by another library (e.g., library dependencies). ▪ When using **ldd** to track down problems, check the complete dependency chain. ▪ Run **ldd** as root (recommended). Be aware of the following options: ▪ **-v** displays all information. ▪ **-u** displays unused direct dependencies. ▪ **--version** displays the version number of **ldd**.
ldconfig	Reloads the library cache every time libraries are added or removed, and update the symbolic links. This creates the necessary links and cache for the most recent shared libraries found in the directories specified on the command line, in the file **/etc/ld.so.conf**, and in the trusted directories (**/lib** and **/usr/lib**). Be aware of the following options: ▪ **-v** summarizes the directories and files it is registering as it reloads the cache. ▪ **-N** updates symbolic links, but does not update the cache. ▪ **-n** updates the links contained in the directories specified on the command line. ▪ **-X** updates the cache but does not update symbolic links. ▪ **-f** changes the configuration file from **/etc/ld.so.conf** default. ▪ **-C** changes the cache location for the **/etc/ld.so.cache** default.

	▪ **-r** treats a new directory as if were the root directory. This is helpful when you are recovering a badly corrupted system or installing a new OS. ▪ **-p** displays the current library cache, including all the library directories and their respective libraries.

TestOut Corporation All rights reserved.