Linux Fundamentals:
Compiling Software

This material is based on work supported by the National Science Foundation under Grant No. 0802551

National Science Foundation
WHERE DISCOVERIES BEGIN

Lesson Overview

In this lesson, you will explore package management systems found in Linux. You will have the opportunity to download, create, install, and remove software packages for various Linux distros. You will also practice these tasks in the command line environment.

Understanding the package management system will allow you to quickly respond to customers' software needs, apply security updates, and troubleshoot problems with automated tools. It is a core competency for Linux administrators.

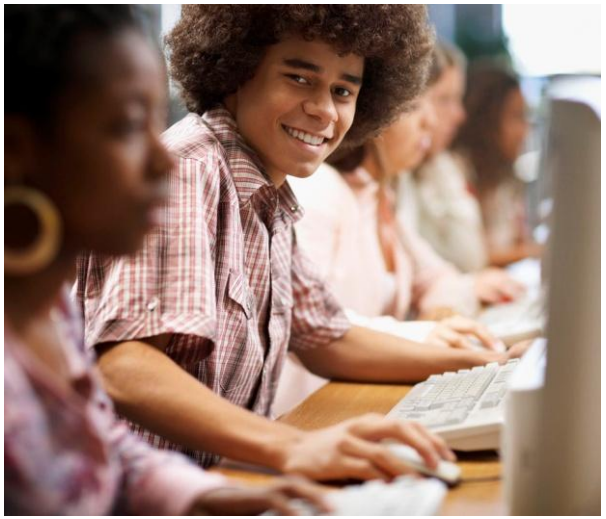
Student Expectations

You should know what will be expected of you when you complete this lesson. These expectations are presented as objectives.

Objectives are short statements of expectations that tell you what you must be able to do, perform, learn, or adjust after reviewing the lesson.

Objective

Given an update to an existing source package, the student will be able to download the appropriate source RPM or DEB package, update the file, recompile, and install the resulting RPM or DEB package correctly, using command-line tools.



Lesson Outline

During this lesson, you will explore:

- Package management
- The sandbox environment
- Software development tools
- RPM Package management
- DEB Package management
- Installation of the source package
- Updating and compiling a RPM package
- Updating and compiling a DEB package
- Installing the updated package
- Troubleshooting common installation problems including unmet dependencies

Creating the Debian Build Environment

Before you can build any software you need to create the build environment and install the build tools for your distribution. In this task, you will learn about the Debian build environment.

The steps to creating the build environment are demonstrated in the video to your right. These steps are:

1. Log in under your regular user ID.
2. Note: Press **ENTER** on your keyboard after each command
3. Open a terminal (console) window.
4. Type **sudo apt-get update** to update the package list.
5. Type **sudo apt-get upgrade** to upgrade the system to the latest versions of all installed software.
6. Type **sudo apt-get install build-essential** to install the build software.
7. Install the suggested software with the exception of the **debian-maintainers** package.
8. Create a **code** directory under your home directory using the command: **mkdir code**.

At this point you are ready to get the source package you would like to update.

Select **PLAY** below to view the Debian build video.

View Video VideoLesson8DebianBuildEnvironment(C2L8S14).mp4

Required Reading

- Debian new maintainers guide
- Debian building packages
- Updating Ubuntu Package

Installing Deb Source Package

At this point you have created your build environment on your Debian based machine. Now you will need to retrieve the source package for the application you wish to rebuild.

In this example, you are going to rebuild a package called **cvs-syncmail**.

1. Change to the *code* directory: **cd code**
2. Note: Press **ENTER** on your keyboard after each command.
3. Retrieve the source file by typing **apt-get source cvs-syncmail**
4. The *cvs-syncmail* package will install under your code directory.
5. Change to the *cvs-syncmail directory* using **cd cvs-syncmail***
6. Note: You must change the * to the version number of the source file you downloaded, as in **cd cvs-syncmail2.14-8**
7. Make any changes / updates/ adjustments needed.

The next step will be to build the Debian package for distribution to other machines.

Select **PLAY** below to view the Debian package install video.

View Video VideoLesson8InstallDebianSource(C2L8S15).mp4

Required Reading

- Debian package

Building Debian Package

Since you have your source package installed from the last step and have made your changes, it is time to build the new installable packages.

1. Note: Press **ENTER** on your keyboard after each command
2. Go to your source directory.
3. Use the command: **dpkg-buildpackage**
4. Check any error messages for unmet dependencies and correct them using: **sudo apt-get install**
5. Once all dependencies are corrected and installed the *dpkg-buildpackage* command will run without errors.
6. In your *home/code* directory will be your new deb packages that can be installed using the **sudo dpkg -i** command.

Note: You may have to force the package to install if the new package does not have a version number upgrade that is higher than the existing package. For instance, you cannot install a version 1.01 package to an existing 1.01 package. However, a version 1.02 will have no problems updating a 1.01 package.

Select **PLAY** below to view the Debian build package video.

View Video VideoLesson8BuildingDebianPackage(C2L8S16).mp4

Updating Debian Packages

Since Linux is comprised of many open source projects with thousands of developers around the world working on applications, bug fixes and improved functionalities are sometimes delayed before they are available through official channels. Consequently, a Linux administrator will be required to update an existing package to bring it up-to-date with the upstream version (or the developer's latest release). The process is similar to recompiling and require the administrator to:

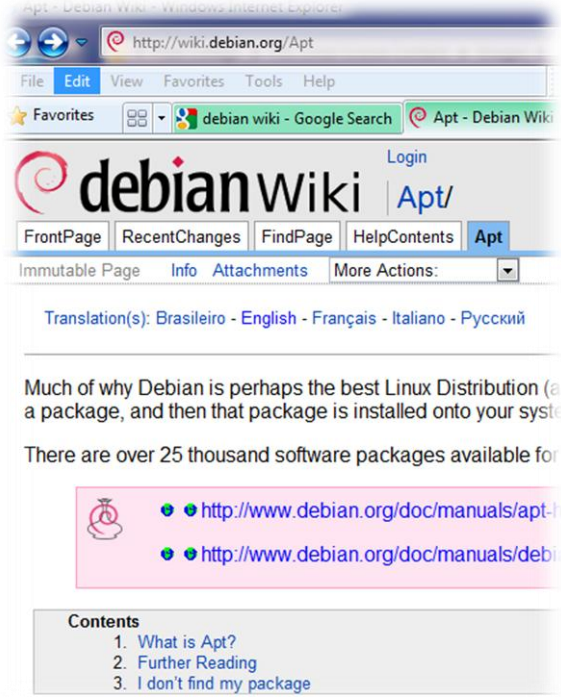
1. Update the build system, then access the command line.
2. Note: Press **ENTER** on your keyboard after each command.
3. Go to the *code* directory using **cd ~/code**
4. Get the source package using **apt-get source** <packagename>
5. (Note: replace <packagename> with a real package name above.)
6. Locate the new version of the applications source code in .tar.gz format.
7. Un-archive the downloaded source package in the *~/code* directory. Be careful not to over-write existing code.
8. Copy the source tar.gz file from the *download* directory and rename it to the standard Debian format. This name change will match the source (original) file that was placed into the directory with the source package from apt-get, but the version numbers will be different.
9. Change to the new source directory.
10. Enter the command **./configure**
11. Type **make**
12. Correct any errors.

Updating Debian Packages Contd

After checking and correcting any errors:

1. Go to the *code* directory using **cd ~/code**
2. Note: Press **ENTER** on your keyboard after each command .
3. Remove the directory in which you built the code.
4. Un-archive the source tar.gz file and go to the directory.
5. Copy the Debian directory from the original code installed by the *apt-get* source command in step 4 in the previous screen.
6. Delete the old patch directory.
7. Use command **rm –rf debian/patches**
8. Update the changelog with **dch –I** and correct the version number on the top line and then your email address. Put a comment after the asterisk * in the top entry. Save the file.
9. Compile the package using **dpkg-buildpackage**.
10. Fix any dependencies that cause the script to report an error and try step 8 again.
11. At the end, you will get a message about signing the packages, ignore it for now.
12. Your new deb packages will be in your *~/code* directory.

The process is a little different on RPM based machines.



Required Reading

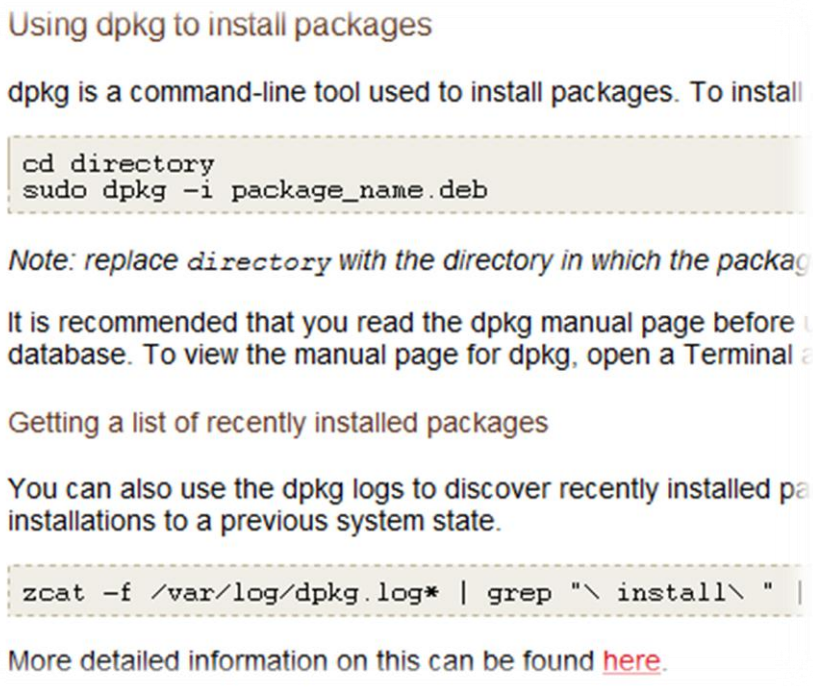
- Packaging for Ubuntu
- Packaging for Debian

Summary of Debian Packaging

To summarize the build process:

1. Update the build system.
2. Setup the build directories.
3. Install the build software.
4. Get the source packages.
5. Install the source packages.
6. Make necessary changes.
7. Build the packages, making sure to install required dependencies.
8. Test the install using *dpkg* to make sure it installs cleanly on a test machine.
9. Distribute.

The process is a little different on RPM based machines.



Using dpkg to install packages

dpkg is a command-line tool used to install packages. To install

```
cd directory
sudo dpkg -i package_name.deb
```

Note: replace `directory` with the directory in which the packag

It is recommended that you read the dpkg manual page before
database. To view the manual page for dpkg, open a Terminal

Getting a list of recently installed packages

You can also use the dpkg logs to discover recently installed pa
installations to a previous system state.

```
zcat -f /var/log/dpkg.log* | grep "\ install\ "
```

More detailed information on this can be found here.

RPM Packaging

Creating Build Environment for Redhat

As with Debian, before you can build any software on a Redhat system, you need to create the build environment and install the build tools for your distribution. In this task, you will learn about the RPM build environment. The steps to creating this environment are demonstrated in the video to your right. These steps are:

1. Log in to your Redhat system using your regular user id.
2. Note: Press **ENTER** on your keyboard after each command.
3. Type **sudo yum upgrade** to update your build system to the latest version of software.
4. Type **sudo yum install rpmdevtools** to install the build software.
5. Type **sudo yum groupinstall "Development Tools"** to install all of the development tools.
6. Type **sudo yum install rpm-build** to install the build scripts.
7. Create the build directory structure.
8. Type **cd** to make sure you are in your home directory.
9. Type **mkdir -p ~/src/redhat** to create your Redhat build directory.
10. Type **cd ~/src/redhat** to go to that directory.
11. Type **mkdir BUILD RPMS SOURCES SPECS SRPMS tmp** to create the required directories. (Note: the names such as BUILD and RPMS are case sensitive and the build system expects to find these exact names.)

Select **PLAY** below to view

View Video VideoLesson8RedhatBuildEnvironment(C2L8S21).mp4

Creating Build for Redhat Contd

To continue the Redhat build:

1. Type **cd** to return to your home directory.
2. Note: Press **ENTER** on your keyboard after each command.
3. Using the curl command, you need to retrieve the default .rpmmacros file using the command:

   curl http://lms.lincs.pscit.org/content/PSC2/lesson8/rpmmacros.txt > .rpmmacros

4. Using any text editor (like vi or pico) to edit the .rpmmacros file you just retrieved.
5. Change the FIXME in the home directory and the FIXME in the email address line to reflect the correct information. Your home directory is normally your user id and your email address needs to be fixed as well. Don't change anything else in that file.
6. Save and exit back to your home directory.

Required Reading

- Building RPM Packages Pt 1
- Building RPM Packages Pt2
- Building RPM packages Pt3

At this point you are ready to get the source package you would like to update.

Installing RPM Source Package

At this point you have created your build environment on your RPM based machine. Now you will need to retrieve the source package for the application you wish to rebuild. In this example we are going to rebuild a package called "mc" which is a command line file utility.

1. Go to to the home directory using **cd**.
2. Retrieve the source file by typing **yumdownloader --source mc**
3. The mc source rpm will now be in your home directory.
4. Verify that the mc package is installed on the build system using **sudo yum install mc**.
5. Allow any dependencies to install.
6. Install the source rpm by typing **rpm -i mc*.*.rpm** where * and * are the version and architecture for the rpm you downloaded.
7. Go to the ~/src/redhat directory using **cd ~/src/redhat**.
8. The source code and any required patches or files are in the SOURCES directory and the build directions are in the SPECS file under the name **mc.spec**.
9. Make any changes to these files that you need to make.
10. Update the release information in the spec file and save. Use any text editor to do this. Always put company or personal initials after the release information so you don't confuse your packages with the official release versions.
11. If you are fixing bugs, please consider submitting your fixes to the development team so they may incorporate your fixes in the upstream package.

You are now ready to re-build the rpm packages.

Select **PLAY** below to view the Debian package install video.

View Video VideoLesson8InstallRPMSource (C2L8S23).mp4

Required Reading

- Create RPM packages

Building the RPM Package

The next step will be to build the RPM packages for distribution to other machines now that the source package is installed.

1. Note: Press **ENTER** on your keyboard after each command.
2. Change into your *~/src/redhat* directory and use the command: **rpmbuild -ba –-target=i686 SPECS/mc.spec**
3. Substitute the **i686** for any other system architecture for which you are building.
4. Substitute the **mc.spec** for any other package spec file name you are using.
5. Check any error messages for unmet dependencies and correct them using **sudo yum install**
6. Once all dependencies are corrected and installed, the **rpmbuild** command will run without errors.
7. In your *src/redhat/RPMS* directory will be your new rpm packages that can be installed using the command: **sudo yum –nogpgcheck localinstall**
8. Note: *You may have to force the package to install if the new package does not have a version number upgrade that is higher than the existing package. For instance, you cannot install a version 1.01 package to an existing 1.01 package. However, a version 1.02 will have no problems updating a 1.01 package.*
9. Your modified source RPMs will be found in the *~/src/redhat/SRPMS* directory and should be saved in case you need to make future changes.

Select **PLAY** below to view the Debian package install video.

View Video VideoLesson8BuildingRPMPackage(C2L8S24).mp4

Updating RPM Package Versions

One of the most frequent tasks of a Linux administrator is to update packages to more recent versions. These updates occur frequently as upstream developers fix bugs, make modifications, and add features to packages. Consequently, administrators must become familiar with the process of updating package versions.

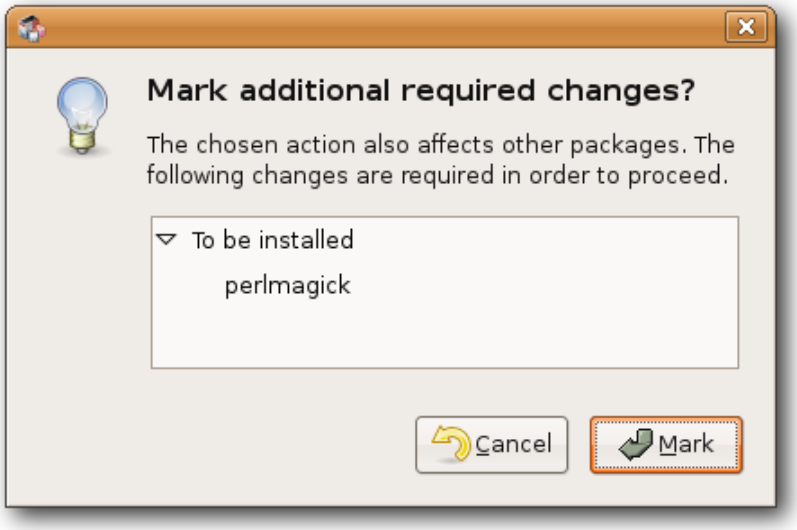
1. From a terminal window, update your system to make sure all software is current.
2. Note: Press **ENTER** on your keyboard after each command.
3. Install the source package from Yum using **yumdownloader –source packagename**
4. Download the new source into **~/src/redhat/SOURCES/** using any software you wish.
5. Edit the SPEC file in **~/src/redhat/SPECS/** for that package name.
6. Change the version number and release number.
7. Fix the packager name and email address.
8. Comment out all patches by adding a # sign in front of the line since you may not need them for the new version.
9. Update the change log entry near the bottom of the spec file.
10. Save and exit the editor.
11. Use **rpmbuild –ba packagename** to build the packages.
12. Correct any dependency issues using **sudo yum install packagename** to install missing packages.
13. Redo steps 11 and 12 to correct errors.
14. After a successful build, RPM files will be stored in the RPMS directory and source RPMS in the SRPMS directory.

Note: test your builds on non-critical machines before releasing for distribution!

Summary of RPM Packaging

To summarize the RPM build process:

1. Update the build system
2. Setup the build directories
3. Install the build software
4. Get the source packages
5. Install the source packages
6. Make any changes
7. Build the packages, making sure to install any dependencies
8. Test install using **sudo yum –nogpgcheck localinstall** to ensure it installs cleanly on a test machine
9. Distribute



Lesson Summary

Building packages is the primary method of installing software on Linux machines. Without packages, it would be difficult to maintain version control over large numbers of machines. This confusion would introduce software instability and unreliability to an environment that must remain stable.

Six popular versions of Linux include Redhat, Suse, Ubuntu, Fedora, Centos, and Debian. Some of these OS versions share packages, but others are incompatible. The Deb package is used on Debian and Ubuntu systems, but the RPM package is used on Redhat, Suse, Fedora, and Centos. Both RPM and Deb packages serve the same purpose but are built in different methods.

All software installed on a Linux machine should be built into the packages. One sure sign of a poor or inexperienced Linux administrator is the choice to update software or settings without using the package management system.