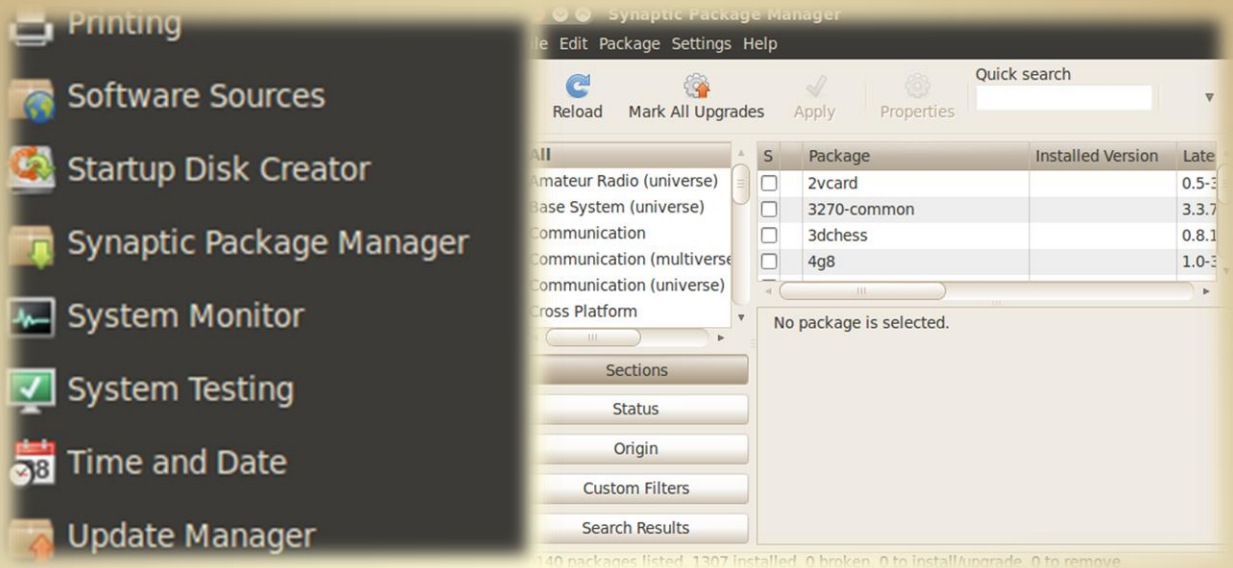


Package Management (CLI)

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

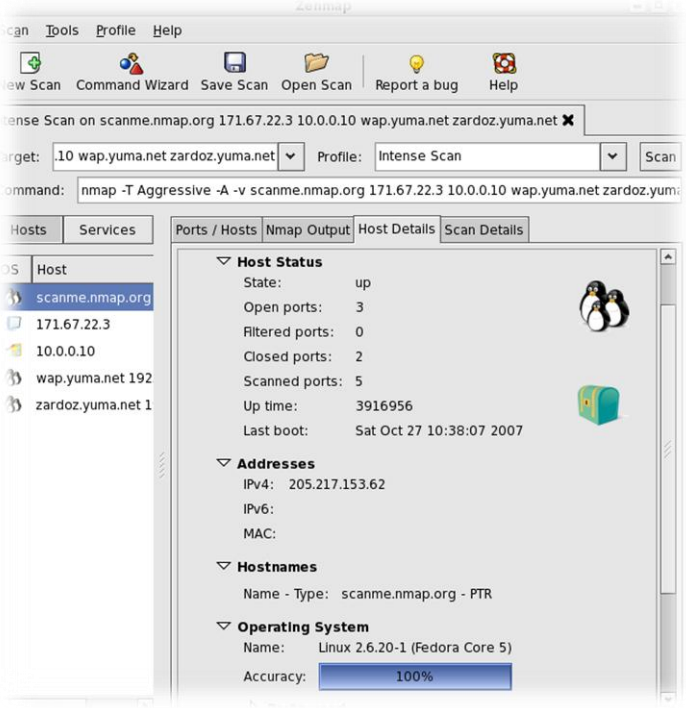
National Science Foundation
WHERE DISCOVERIES BEGIN

Lesson Overview

As Linux popularity increases, IT technicians will need to know and understand the processes involved with maintaining a functional Linux platform. These processes include updating, installing, removing, and managing applications through both desktop and command line environments. Additionally, technicians may be required to manually modify and build an application's source code.

This lesson will review these different tasks and give you an opportunity to build an application from source files. This is an important skill to learn because it is a necessary part of an administrator's job.

In this lesson, students will download source code for the popular network tool NMAP. Students will build and compile the source code to create a working application.



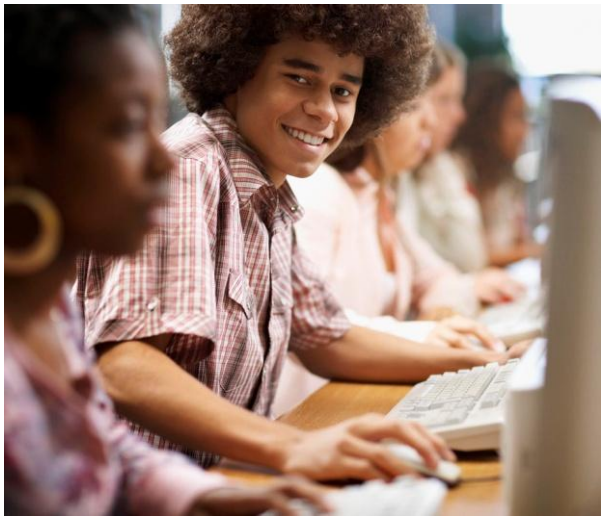
Student Expectations

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 the requirements to add new functionality to a Linux system, the student will compile source code into a running binary program.



Outline

In this lesson, you will explore:

- Packet Management in a CLI environment
- Compiling Source Code
- Installing from Source Code

Additional Needs:

To complete this lesson, you will also need NMAP.

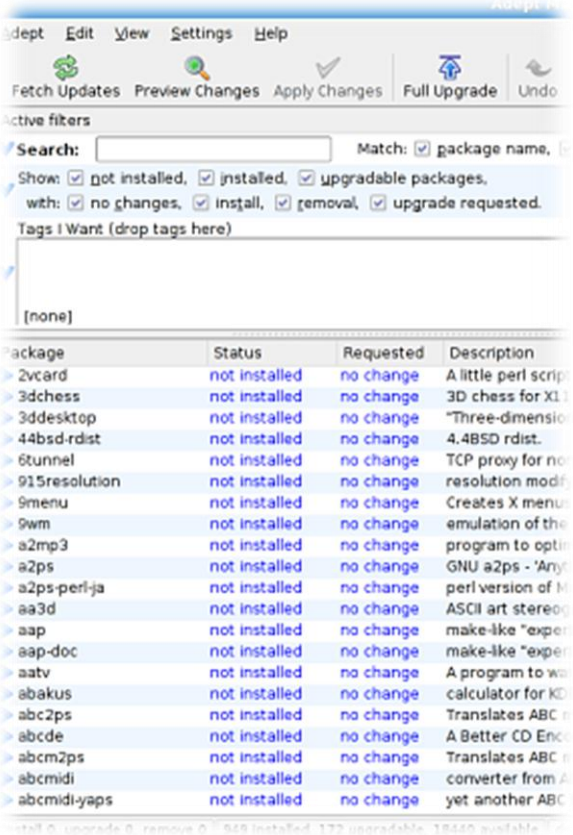
Note: Users should only run NMAP on systems in which they have permission to use. Some administrators and/or intrusion detection applications may see NMAP as a hostile action.

Download the NMAP application to your Linux system. Use the information provided in this lesson to install it when required.

Introduction

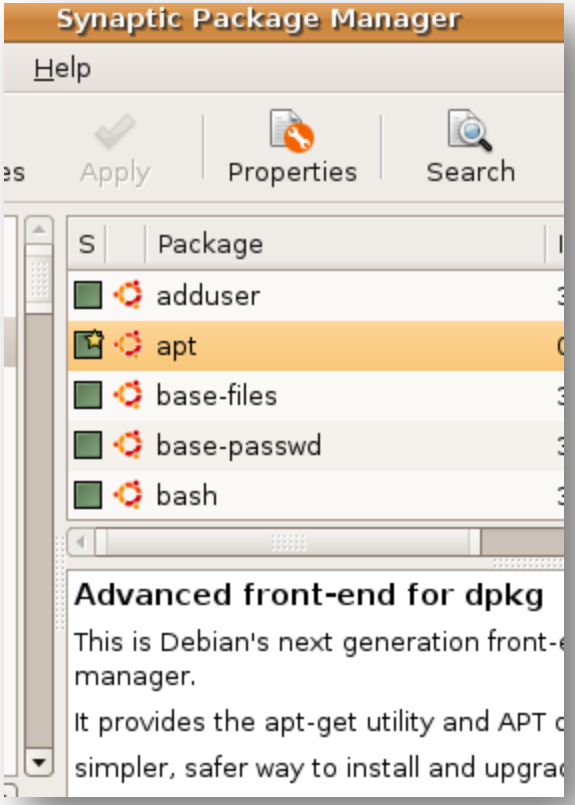
One of the advantages of using a Linux platform is ease with which the different “flavors” are updated. Like the Linux Development Community, the different distros are constantly changing. The frequency of updates allow for a “dynamic” update cycle that includes dynamic websites that change on the fly and dynamic IP leasing that allow IP addresses to be obtained on the fly. Various versions/flavors of Linux can also be updated dynamically using basic commands or GUI applets.

Linux is being developed in a “dynamic” manner, which forces end users and developers to update their systems regularly. Developers can make tweaks to existing applications and post them in the Linux repository while end users can use update tools to automatically check the repository and download/install the modified applications. Unlike other operating systems like Windows, Ubuntu changes on a daily basis , which creates a truly a truly dynamic and robust operating system.



Updating Linux Systems

Some of the tools used for updating are be GUI-based such as the update tool in Ubuntu. Access it by selecting **System>>Administration>>Update Manager** from the toolbar. It is an easy to use, self explanatory GUI tool. If you wish to add or remove software applications or services, you can access the Synaptic Package Manager by selecting **System>>Administration>>Synaptic Package Manager** from the tool bar on your desktop. Like the update manager, the Synaptic Package Manager offers an easy-to-use GUI.

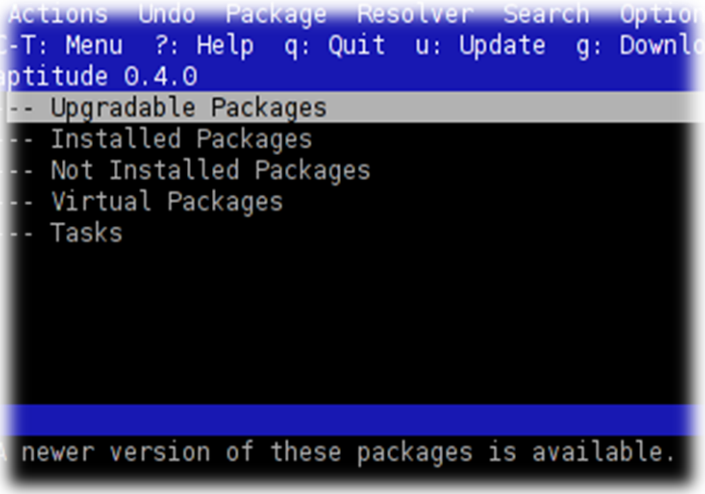


Synaptic is a full featured package manager that includes dependency management as well as package management. This combination helps ensure your Ubuntu installation will upgrade in a simple and reliable manner. Synaptic is probably the easiest way to keep your system up to date .

These options work well for GUI-based navigation, but those who opt for a command line environment may be in for a challenge. Well, not really… Linux, and especially Ubuntu versions, offer simple tools for updating, upgrading, deleting, and managing applications and services. Almost anything that can be done through a GUI, can also be done through the command line interface.

Package Management (CLI)

Several options exist for updating and managing packages through the command line interface. CLI tools include **apt-get**, **aptitude**, **dpkg,** and **dselect**. Like their GUI-driven counterparts, command line tools go out to the Linux repository and check for updates to the various source and object files contained within the storage site. When properly used, these CLI tools can be quicker than their GUI counterparts and after a brief learning curve, most Linux administrators prefer updating using CLI methods.



CLI tools:

- **Apt-get** - a command line utility that provides subcommands, which enable you to install, remove, and manage packages on your Ubuntu system.
- **Aptitude** - A terminal oriented utility that serves as a front-end to lower level utilities such as apt-get and dpkg. The aptitude utility provides both a graphical and menu /screen oriented user interface to install, remove, query, and manage packages from the command line.
- **Dpkg** - The basic command line tool for installing, removing, querying, and managing packages.
- **Dselect** - A terminal-oriented front-end to the **dpkg** utility.

Required Reading:

- Apt-get
- Apt-get Howto
- Aptitude
- Dpkg
- Dselect

Apt-Get CLI Commands

Commands	Description
sudo apt-get install *packagename*	Used install an application
sudo apt-get remove *packagename*	Used to remove a package
sudo apt-get update	Used to retrieve a new list of available packages
sudo apt-get upgrade	Used to upgrade a system with new updates
apt-get help	Provides information on apt-get

Yum

Yum is another package management tool used primarily with Redhat , Fedora, and CentOS. There are a few other distros that use Yum, but the aforementioned are the three big hitters. Yum is used in the same way as Apt, and after a brief learning curve, administrators normally select it over the GUI driven management utilities as well.

Commands	Description
su -c 'yum install *nmap*'	To install the NMAP application
su -c 'yum update *nmap*'	To update the NMAP application *Note… if the old application is in use when you do an update, the old application will stay as is until you turn the service off and then restart it.*
su -c 'yum groupupdate "*MySQL Database*"'	To update all the packages in the MySql group
su -c 'yum remove *nmap*'	Removes the NMAP application

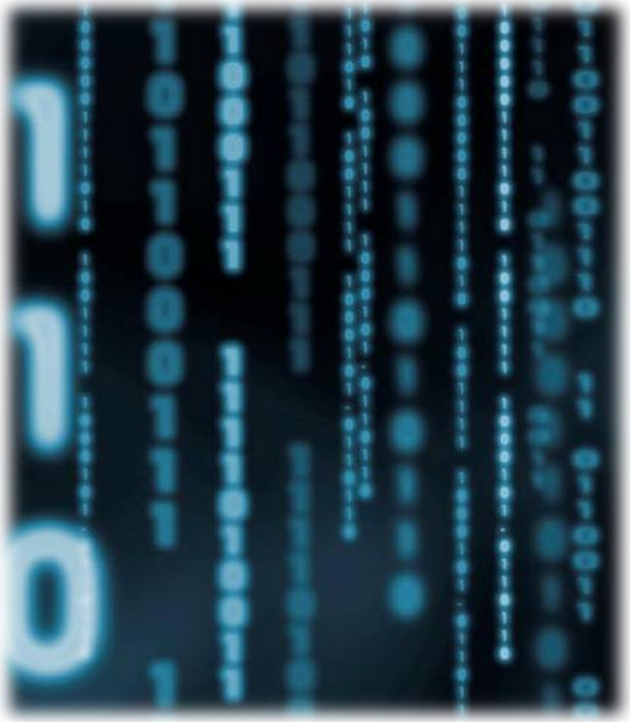
Installing From Source Code

Compiling Source Code

There will be times you will want to modify an application for your own personal use. Linux apps are created in such a way that users can access their source code and modify them to fit their needs.

The Linux philosophy and culture has been built upon years of individual users and developers modifying the source code and making the applications and fundamental services better with time. You are actually encouraged to experiment and redesign Linux as an end user.

Downloading and compiling source code is quite easy once you have been exposed to it. Source packages are downloaded in a similar way as the compiled packages, and the source code can be easily viewed with most text editors. You can modify the source code, recompile it, and test it to determine if your changes have the expected results.



Listing Packages

Command	Description
dpkg –l nmap	Will probably list nothing because it is using a pattern match. You should use a wild card
dpkg –l 'nmap*'	Should produce the desired results Note: You may need to add another wild card such as **dpkg –l '*nmap*'** to get any item that has nmap somewhere in its name.
aptitude show nmap	Listing information about a package… (nmap)
dpkg –L nmap	Listing the contents of a package… (nmap)

Software Distribution Formats

Linux software is distributed via four main sources: *source code, binary files, self-installing binaries,* and *packages*.

Source code: Programmers write software in various programming languages, such as C and C++. The resulting code is known as *source code*. To make source code usable, it must be *compiled* into a *binary*. The cornerstone of the Linux philosophy is that source code must be shared, consequently, you will usually find the source code of a program available at the developer's web site. You can then download and compile this source code on your own system (or, if you are so inclined, you may study the source code to further your understanding). Although compiling source code is not very hard to do, it is more convenient to download either a binary version of the program or a package.

Binary files: Ready-made binary files are sometimes available at the developer's web site. In other words, the programmer has taken his or her own source code and, as a service to users of the program, compiled it so that it is ready for use as soon as it's downloaded. Sometimes binary files come with scripts to help you install them. However, in most cases, you simply place the files in a convenient location on your hard disk, and then run them from there.

Software Distribution Formats (Contd)

Linux software is also distributed via:

Self-installing binaries: Some larger programs are made available as self-installing binary files. This process is similar to the way Windows works, because a GUI-based installation wizard takes you through installation process when the file is executed.

Downloading the nmap installer for Windows from the website www.NMAP.org works in this manner.

Package files: In many cases, you will find that a package file of the program is available. In this case, someone has compiled the software files and put them all together in a single, easily transportable file. Ubuntu package files end with *.deb* file extensions, but other Linux distributions use other package formats, such as .rpm (Fedora/Red Hat, SUSE Linux, and CentOS, among others). Deb files are much easier to work with and preferred by new Linux users while the *rpm* format can be extremely cumbersome for new Linux users as well as for those with experience.

Installing Process

Before installing an application from source files, you first need to download and build the dependency files. Open a terminal and enter:

- **sudo apt-get build-dep nmap**

- Review the *Download source* video on the right.
- Review the *Download source 2* video on the right.

- Download the source code files by entering:
- **Apt-get source nmap**

- Read the readme file – always, always, always read the *readme* files of any package before you install it. In all fairness, a good system administrator or IT technician will always read the extra documentation that is provided with any source code, driver package, new application, etc just in case there is something important that may need to be done for the object to function correctly.

- Review the *Configure source* video on the right.

Select **PLAY** below to review the instructions:

View Video VideoLesson6DownloadSource(C2L6S25V1).mp4

Download source

View Video VideoLesson6DownloadSource2(C2L6S25V2).mp4

Download source 2

View Video VideoLesson6ConfigureSource(C2L4S25V3).mp4

Configure source

Commands Used

Four important commands are used when converting and installing from source code:

- Configure
- Make
- Make-install
- Make clean

Each will be discussed in turn.

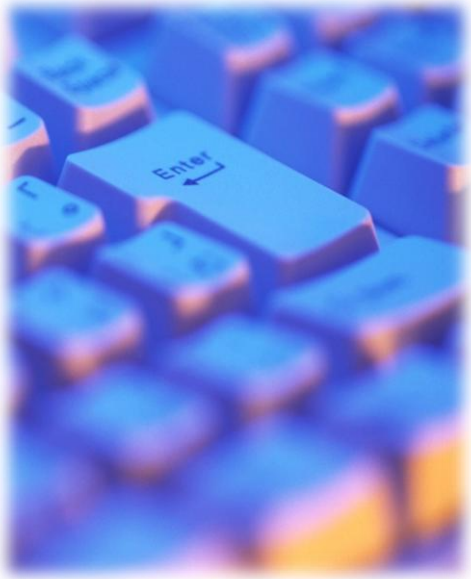
Commands Used: Configure

The *./configure* command tells the shell to run the script named ' *configure* ' which exists in the current directory. The *configure* script consists of many lines which are used to check details about the machine on which the software is going to be installed. This script also checks for dependencies on your system. For the particular software to work properly, it may require other software to exist on your machine already.

When you run the *configure* script you will see several lines of output on the screen , with a question and answer style and a respective *yes/no* as the reply. If any of the major requirements are missing on your system, the *configure* script will exit and you will not be able to proceed with the installation until the required files are installed.

The main job of the *configure* script is to create a 'm*akefile*'. This is a very important file for the installation process. Depending on the results of the tests (checks) that the configure script performed, it would write the various steps that need to be taken (while compiling the software) in the file named *makefile*.

If you get no errors and the *configure* script runs successfully (if are errors the last few lines of output will list them) then you can proceed with the next command which is the *make* command.

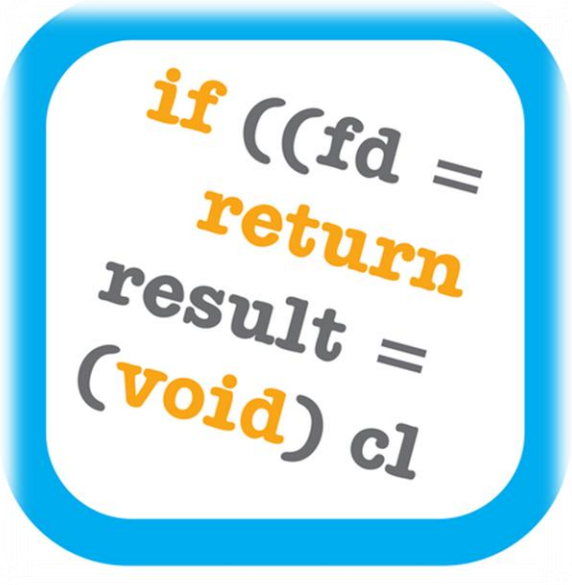


Commands Used: Make

The ' *make* ' command is actually a utility which exists on almost all Unix systems. In order for the *make* utility to work, it requires a file named *makefile* in the same directory in which you run *make*. Remember , the *configure* script's main job was to create a file named *makefile* to be used with the *make* utility.

The *make* utility compiles all your program code and creates the executables. For particular sections of the program to complete, other parts of the code must be ready to go. *Makefile* ensures this process works by setting the sequence of events so that your program does not complain about missing dependencies.

If *make* runs successfully, then you are almost finished with the installation. Only the last step remains which is *make-install*.



Commands Used: Make-Install

When you run *make* without parameters, the instructions in the *makefile* are executed from the start as per the rules defined within the *makefile*. When *you run make* with *install* as the parameter, the *make* utility searches for a label named *install* within the *makefile*, and executes only that section of the *makefile*.

The *install* section happens to be the only a part of the script where the executables and other required files created during the last step (i.e. make) are copied into the required final directories on your machine. E.g. the executable that the user runs may be copied to the /usr/local/bin so that all users are able to run the software. Similarly, all other files are copied to the standard directories in Linux. Remember, when you ran *make*, all the executables were created in the temporary directory where you unzipped your original tarball. So when you run *make install*, these executables are copied to the final directories.

- To run *configure* (you usually have to type **./configure** as most people do not have the current directory in their search path). Running *configure* builds a new *makefile*.
- Type **make** to build the program. That is, when *make* is executed, it will look for the first target in *makefile* and perform the listed instructions. The expected end result would be to build an executable program.
- As root, type **make install**. This action again invokes *make*, which will find the target install in *makefile* and follow the instructions to install the program.
- Run the **make clean** command to remove extra source code no longer required for installation. The *make clean* command cleans up your system and provides more disk space for regular use.

Lesson Summary

Linux systems are updated regularly and include several tools to help with this task. You may choose to update using a GUI environment or you can invoke a command terminal and use the CLI environment to perform updates. Most users consider the CLI to be a faster option.

Popular tools for Linux updates include Apt-get, Aptitude, Dpkg, Yum, and Dselect.

To install applications and updates, you may have to compile source code. Linux is well optimized for this task and uses commands such as configure, make, make-install, and make clean to compile source code, create the necessary executables, and install the updates.