



Installing and Removing Software

One of the fun things about running any operating system is the ability to expand it—to add in new software over time to improve your workflow or just for entertainment value.

Linux is blessed in this regard, because tens of thousands of software titles are available to meet just about every need. However, even if you’ve tracked down the ideal software title, there’s just one barrier to overcome: actually installing it on your system.

Installing software under Ubuntu isn’t the same as with Windows. Users are afforded a lot more power over what happens to their systems, but this comes at the expense of needing to take a little time to understand the terminology and techniques. That is what you’ll learn in this chapter.

Software Installation Basics

Installing programs on Windows is relatively easy. If you wish to use the WinZip archive tool, for example, you can browse to the web site, download the installer .exe file, and install the software. Although you might not realize it, a lot of work goes into making this seemingly simple task possible. Once the original software has been created by the programmers, it must be made into a form that you, the end user, can deal with.

The first thing to happen is that the software is *compiled*. This is the process of turning the source code created by programmers into an actual file (or set of files) that can be used on a daily basis. On most systems, compiling source code involves a lot of number crunching. This takes time—hours, in some cases—and this is why it isn’t normal practice to compile the source code every time you want to run the program.

Once the program files have been compiled, there needs to be a way they can be installed on various systems and easily transported across the Internet. This is where *packaging* comes into the equation. Programs usually consist of many files. To make each program file individually available would mean that some are sure to get lost or corrupted, and the program wouldn’t work. Therefore, the files are usually combined into a single archive file. In addition, third-party system files are added to ensure compatibility on all computers

and an extra program, called an *installer*, is added so that users can quickly get the files onto their systems.

All of this means that, to be able to install a program like WinZip on Windows, all you need to do is download the installer .exe file and run it once. No more work is necessary.

Linux is a little more involved, largely because it never assumes that users want their environment to be simplistic and with limited options. However, most Linux distributions still embrace the paradigm of packaging software into a single, easily transported file. We will explain how software packages work in Ubuntu in a moment, but first it's necessary to understand other typical software distribution file formats used in the world of Linux.

Formats of Linux Installation Files

If you visit the web site of a particular Linux application, you may find that it's available to download in a number of different formats. The program will almost certainly be available as *source code*—the original human-readable text that the developer created. But it might also be available as a binary or a package file.

Tip Linux isn't the only operating system for which open-source programs are created and used. There are open-source projects for both Windows and Apple Macintosh, many of which are hosted at the <http://sourceforge.net> web site. Many other, less widely used operating systems also rely on open-source software to a greater or lesser extent.

Here are the formats by which Linux software is normally distributed:

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

Binary files: You might find ready-made binary files are 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's ready for use as soon as it's downloaded. For example, this is how Linux versions of the Mozilla Foundation software, like Thunderbird and Firefox, are currently distributed if you download them directly from www.mozilla.com. 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.

Note In the cases of both source code and binary files, the files usually come in a *tarball*, which is a single archive file containing other files. A tarball isn't, by definition, compressed, but usually either the `bzip2` or `gzip` tools are used to shrink the file, to ease transportation across the Internet.

Self-installing binaries: Some larger programs are made available as self-installing binary files. This comes very close to the way Windows works, because when the file is executed, a GUI-based installation wizard takes you through installation. If you download OpenOffice.org from the official web site (www.openoffice.org), for example, you'll end up with a single 80MB+ file, which you then simply execute from the command line.

Package files: In many cases, you'll 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 Mandriva, among others).

Note As a blanket rule, an installation package created for one distribution won't be compatible with another. It's possible to use a program called `alien` under Ubuntu, which aims to convert packages between distributions and different package formats, but this should be seen as a last resort. You'll be better off simply obtaining a package specifically designed for your Linux distribution.

Package Management

Of all the preceding formats, packages are by far the most common and popular in the world of Linux. Ubuntu utilizes packages, as do nearly all other distributions. In fact, the Ubuntu DVD-ROM contains hundreds of packages.

A well-implemented package management system is able to install programs, upgrade them, and uninstall them, all with just a few keystrokes or clicks of the mouse. It vastly reduces the amount of work required to get new software onto your system and makes maintenance tasks such as upgrading software easy, too.

It's important to understand what an Ubuntu package file actually is and what it contains. With Windows, an installation `.exe` file is effectively a piece of software combined with an archive of files. When you run the executable, it triggers a small program contained within it that then unpacks the contents of the file and installs them to the hard disk.

In contrast, package files used by Ubuntu merely contain the program files along with a handful of configuration files to ensure the software is set up correctly. Package files are useless without the various pieces of software already installed on the system that are used

to manipulate them and do the hard work of installing, removing, and querying them. This software is known as the *package management system*. In the case of Ubuntu, the package management system has two components: `dpkg` and `APT`.

The use of a package management system has a number of benefits. The package management system builds its own database, so it knows exactly what programs are installed at any one time. Therefore, you can simply query the database rather than search the applications menu or hard disk. The package system also keeps track of version numbers. This gives the user much more control over the software on the system, and it makes updating easy.

The use of a package management system also means that if a program starts to act strangely, its configuration files can simply be refreshed using the package manager. There's no need to uninstall and reinstall the software, as is so often the case with Windows programs.

Dependency Management

One of the key features offered by any package management system is *dependency management*. Put simply, the package manager ensures that if you install a piece of software, any additional software it relies on to work properly is already present on the system. If the software isn't present, the package manager must either resolve the situation automatically or ask you what to do.

Sometimes, the software you want to install might depend on other programs on your system, but more often, the dependencies take the form of system libraries. It helps if you realize that not all packages contain software that you, as a user, will make direct use of. Some packages contain nothing but library files—shared pieces of code that are equivalent to `.dll` files under Windows. The key library on an Ubuntu system is the GNU C Library, without which the Linux kernel couldn't function, which is provided by the `libc6` package. But practically every program has its own needs when it comes to library files, and these requirements must be handled by the package manager.

Note One reason Windows installation files are often so large is that they typically come with all the system files they need, in case they're not already present on the system. This does not make dependency problems disappear, however. Third-party applications installers sometimes overwrite existing libraries with versions that are incompatible with the rest of the system.

Dependency management doesn't just mean adding in packages that a piece of software needs. It might also mean *removing* packages already present on your system. This might need to happen if they're incompatible with new software you want to install, something that's referred to as *package conflict*. In addition, sometimes you might want to remove a package that other packages rely upon, a situation known as *reverse dependency*. In such a case, the package manager must either stop you from removing that software,

to avoid breaking the software that depends on it, or remove the reverse-dependency packages, too. In most cases, it will ask you what you wish to do.

DEPENDENCY HELL

If you try to install certain software packages, you will very likely find that they depend on other packages, such as software libraries. These must be either already present on the system or installed at the same time for the software to work correctly. Ubuntu will attempt to take care of the latter automatically. In a similar way, removing software also means that other packages that rely on that particular software must also be removed, a situation known as *reverse dependency*.

Dependency hell comes about when chains of dependencies arise, which is to say, when a program you install or remove involves the installation or removal of several other, apparently unrelated pieces of software. For example, let's say you decide to manually install a program called `Oscar`. You download it and type the command to install it, but you are then told that this depends on another program called `BigBird`, which isn't installed. Fine, you think, I'll just download and add `BigBird` to the same installation command. But it then transpires that `BigBird` has its own dependency of `Snuffleupagus`. You download and add that, too. Alas! `Snuffleupagus` has its own dependency of `MrHooper`.

This can carry on for some time, and this is why it's not advised to manually install or remove software without using something like Synaptic Package Manager. In the preceding example, Synaptic would add in all the dependencies automatically and download and install them at the same time.

Dependency chains like this are a by-product of any package management system. The solution is often simple—just don't remove the software package. After all, hard disks are extremely large nowadays, and space is rarely an issue, so there's little reason to not have software packages you no longer need hanging around.

Software Repositories

As mentioned previously, `dpkg` and `APT` take care of package management within Ubuntu. These tools are taken from Debian, on which Ubuntu is based.

Debian Package, or `dpkg`, is the most basic part of the system. It's used to install and uninstall software, and it can also be used to query any individual software packages. It's like the manager in a warehouse who is tasked with knowing exactly what boxes have been stored where. The manager doesn't know where the boxes come from, and he doesn't know anything about packages outside his warehouse. He just manages the boxes that are delivered to him and that are stored in his warehouse.

`dpkg` is aware of dependency issues and will refuse to fully install a package if the others it needs aren't already installed or supplied at the same time. But it doesn't have the means to fix the situation automatically. This is akin to the warehouse manager's inability to order more boxes if he needs them. That's not his job. He'll just tell you if boxes delivered to him are missing some of their components.

Because of this, there's an additional layer of software that sits on top of `dpkg` called the Advanced Package Tool, or APT. APT is very sophisticated. Its job is to handle dependency management. Try to install some software using APT, and any dependency issues will be worked out for you.

APT can do this because it's designed to work with *software repositories*. Users can search and install packages from these collections of software. More often than not, these software repositories are online, but that's not always the case. The DVD supplied with this book contains the base installation software repository, for example.

Note As you might already have guessed, the Synaptic Package Manager is simply a GUI front-end for the APT system. You can see this clearly when you're installing or removing a package. In the Apply Changes dialog box that appears after software has downloaded, click Terminal, and you'll see the output of various APT commands.

It's important to note that APT relies on the `dpkg` system to take care of the actual installation. Effectively, `dpkg` and APT are two sides of the same coin.

As you might have realized, the package management system means that Linux software installation/removal is a fundamentally different proposition than handling software under Windows or Mac OS. If you want to install new software, the first place to look is the Ubuntu software repositories. The online repositories contain most of the popular software available for Linux right now, all packaged for installation under Ubuntu.

It's comparatively rare for an Ubuntu user to visit a web site and download a package file for installation, as is often the case for Windows users, and the only time this normally happens is if you can't find what you're looking for in the official repositories.

Tip Software repositories don't have to be "official," or sanctioned by Ubuntu, to be used under Ubuntu. Sometimes, you might opt to add repositories that contain particular software, such as multimedia repositories. This may be necessary because multimedia formats are often licensed under terms that Ubuntu doesn't agree with, so it declines to offer this software from its official repositories.

Out of the box, Ubuntu comes with a couple of software repositories already configured. These allow you to download new software and also update the system online. Ubuntu software repositories are subdivided into various categories and components.

SOFTWARE VERSIONS

Because most Linux software is open source, a curious thing happens when it comes to software versions. Rather than there being just one “official” version of a program, such as with most Windows software (where you must download the official version of the file), many individuals and organizations take the source code, compile it, and make their own package files available for others to use.

For example, virtually all the software installed with Ubuntu has been compiled by Ubuntu developers. This means it can be quite different from what’s “officially” available at the programmer’s web site. In some cases, the source code is tweaked to fix notorious bugs or apply a different look and feel to the software, so it integrates with the distribution. Often, the configuration files are changed so that the software works properly under Ubuntu, such as integrating with other software packages.

The programmer behind the software doesn’t mind when such things happen, because this way of working is part and parcel of open-source software. In fact, the programmer is likely to encourage such tweaking.

Because of this, the first place to look if you want any additional software is not the developer’s web site but the Ubuntu software repositories. If the package is available in the Ubuntu main distribution, you’ll get an officially sanctioned Ubuntu release that will fit in with the rest of your system and won’t require much, if any, additional work to get it up and running.

Categories of Repositories

Regardless of whether they’re online or on a CD/DVD, Ubuntu repositories are strictly categorized according to the type of software they contain:

Main Distribution: This repository contains the packages that are required to install Ubuntu. This repository usually takes its name from the code name for the release, and is activated by default. For Ubuntu 8.04, the main distribution repository is called *hardy*, after the code name for the 8.04 release (Hardy Heron). In the previous release, the main distribution repository was called *gutsy*. (For more details on Ubuntu code names, see <https://wiki.ubuntu.com/DevelopmentCodeNames>.)

Security Updates: Sometimes, security flaws are so serious that they need to be fixed immediately, within as little as 24 hours of being discovered. If so, the packages concerned will be placed on this server. The Security Updates server isn’t about new versions or functionality. It’s about fixing security holes rapidly. This repository is also activated by default.

Recommended Updates: This repository contains newer versions of the packages in the main distribution repository. Like Security Updates, this category also offers bug fixes, but these fixes aren’t urgent and are often more substantial than quick patches to fix a critical bug.

Proposed Updates: This is a special category by which testing releases of updates are made available. There's no reason to use this category unless you want to test packages and help fix bugs (for more information, see <https://wiki.ubuntu.com/HelpingWithBugs>). This category is not activated by default.

Backported Updates: The Backports server allows access to software that's intended to go into the next version of Ubuntu but has been packaged for the current version. This software might not have been tested thoroughly and so is suitable only for neophiliacs or those who absolutely need the latest version (perhaps because of a vital new feature it offers). This category is not activated by default.

Repository Components

In addition to the categories listed in the previous section, the Ubuntu repositories are further split into *components* (effectively subsections) according to how essential the software is to a basic Ubuntu installation or the license that the software uses. Here are the components under which software is filed within a repository:

Main: This section contains nearly all the software that's featured in a basic Ubuntu installation. As such, it's all Free Software, and every package is supported by Canonical, the company that oversees the Ubuntu project. That means that updates are frequently provided to fix security holes or simply to keep up with latest releases.

Note *Free Software* refers to software that's licensed under the GNU Public License (GPL). It doesn't mean that the software is free of charge, although that's nearly always the case.

Restricted: Although Ubuntu is mostly Free Software, it must include some drivers released only in binary form (that is, closed source) and that, therefore, have license agreements that are not compatible with the goals of Free Software. That's what you'll find in this section. Some hardware simply won't work fully without software from the Restricted section.

Universe: This section might be referred to as "the rest," because it contains the majority of Free Software available at the present time. Much of it is borrowed from the massive Debian software repository, although the packages are sometimes tweaked to work correctly under Ubuntu before being made available (some people who create Debian packages also create the Ubuntu equivalents). Unlike Main and Restricted, the Universe section is not officially supported by the Ubuntu project, which means there's no guarantee that security flaws will be fixed. Nor is there any guarantee of updates, although most packages are usually updated regularly.

Multiverse: As with the Restricted section, here you'll find software that's released under a software license incompatible with either the letter or spirit of Free Software. However, unlike the software in the Restricted section, none of the software in Multiverse is considered essential to a default Ubuntu installation.

Source Code: This section contains source code packages. Unless you're a software developer, or are thinking of becoming one, this section won't be of much interest.

Now that we've covered the basics of Linux software installation, it's time to talk about the tools used to manage software. First, we'll look at graphical applications that can be used to manage software, and then we'll look at the command-line tools you can use.

The Synaptic Package Manager in Depth

The Synaptic Package Manager is effectively a graphical front end for the APT system. You can use it to search for and install software. To start this program, click System ► Administration ► Synaptic Package Manager.

Searching for Software

Before searching for software, it's always a good idea to refresh the package database. This database describes the software contained in the repositories, and it is held on your hard disk. Just click the Reload button on the Synaptic Package Manager toolbar to grab the latest package lists from the various repositories you're subscribed to (that are in your `sources.list` file). Reloading can take a few minutes on a slow connection, but it ensures that you have access to the latest software within the repositories.

You can search for software in two ways:

Quick: For a quick search, click any entry in the list of packages, and then simply start typing. This will match what you type against the package names and sort the list dynamically, as you type.

In-depth: For an in-depth search, click the Search button on the toolbar. By default, this searches through both package names and the descriptions, for a higher chance of a match. You can type either the specific program name or a keyword that may be within the description. For example, if you are looking for graphics drivers for your nVidia card, but you don't know the name of the package that contains them, you can type *nvidia*.

Tip You don't need to type whole words in the search field. You can type part of a word or, more commonly, the word in a shortened or alternate form. For example, if you're looking for an e-mail client, it might be more fruitful to simply type *mail client* or even just *mail*. This will then return results containing *e-mail*, *mail*, *mailing*, and so on.

By clicking the Settings ► Filters button, you can enhance your search results by filtering out any packages that don't meet your requirements. You can filter by criteria such as whether the software is already installed, whether it's new in the repository, and much more. It's advisable to click the New button to create your own filter before starting, as shown in Figure 28-1, rather than editing one that's already there. Once a filter has been created, you can apply a filter to search results by clicking the Custom Filters button at the bottom left of the main program window, and then clicking the name of your filter in the list.

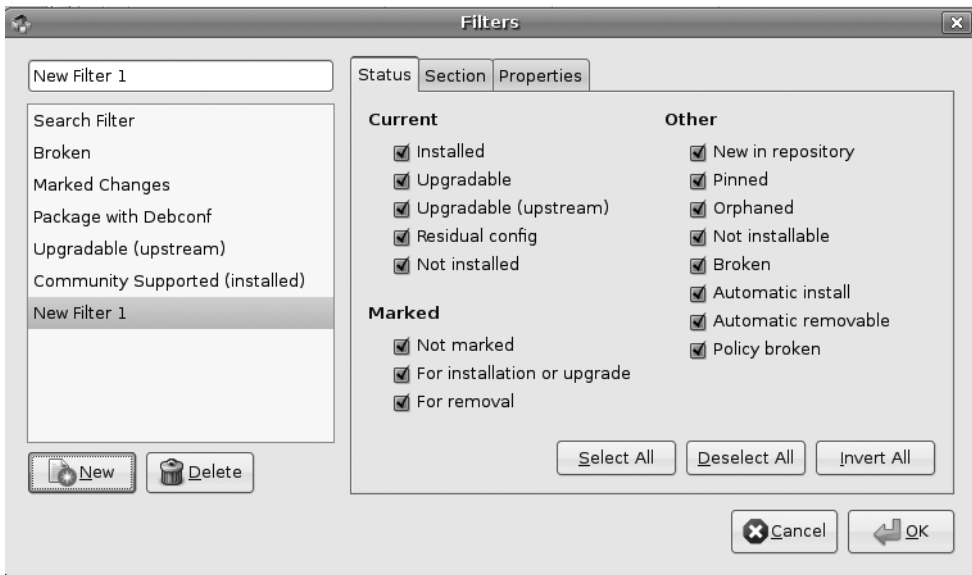


Figure 28-1. Filters can be used to trim the list of search results according to certain criteria.

One use of filtering is to remove the check alongside Installed so that you can remove from the search list any packages that might be already on your system.

Note Filtering can help reduce the number of search results if you use a generic search term, but don't forget to deactivate it when you're finished. To do so, click All at the top of the filters list.

In the search results, any packages with the Ubuntu symbol next to them are supported packages, which is to say, they're from the Main or Restricted software repositories, as opposed to Universe, Multiverse, or a third-party repository. Therefore, future updates are likely to be offered.

If the check box is green, that means the package is already installed. A star next to the check box means the package is new. You can view the complete range of Synaptic icons by clicking Help ► Icon Legend.

Installing Software

When you click the check box next to a piece of software in the search results and select Mark for Installation, the program will be queued for installation, which will take place as soon as you click the Apply button on the toolbar. If the program has any uninstalled dependencies, you'll also see a dialog box asking you to confirm installing those as well. If you agree, these will be automatically added to the list of packages to be installed.

Additionally, if you right-click the file and select Mark Suggested for Installation or Mark Recommended for Installation, you'll see a list of programs that, although not essential to the running of the program in question, will enhance its features to some degree. For example, if you choose to install the VLC media player program, it's also suggested that you install mozilla-plugin-vlc, so that VLC can be used as plug-in for playing media files in Firefox. You don't have to install these recommended programs; the software will run fine without them. But it can often be rewarding if you do so.

Note If the software in the recommended and suggested lists is grayed out, that means it's already installed. It's also possible that the package doesn't have any recommended/suggested packages.

Once you click the Apply button on the toolbar (bear in mind that you can install more than one piece of software at once), you'll see the Summary dialog box, as shown in Figure 28-2. Here, you're once again asked to confirm what needs to be installed. If any software needs to be removed in order to handle dependency issues, you'll be told about this, too. Additionally, under the Summary heading, you'll be shown the total size of the files that will be downloaded, as well as the space required on your hard disk.

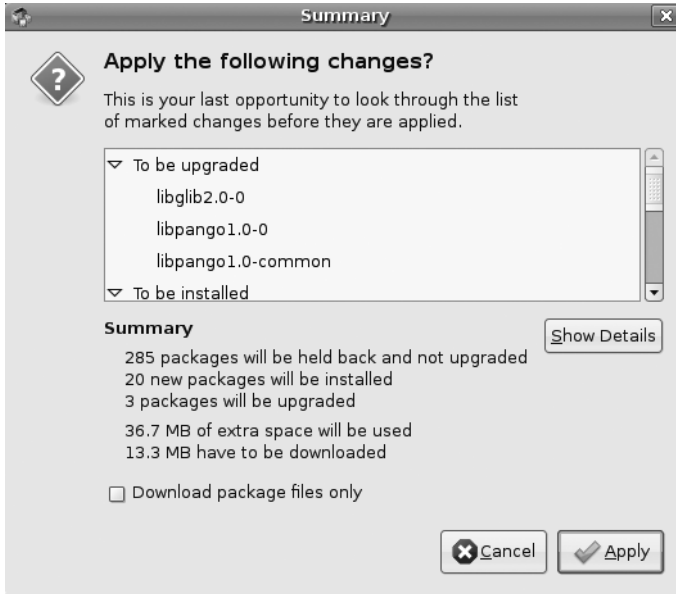


Figure 28-2. Before any software is installed by the Synaptic Package Manager, you'll be told what it is and asked to confirm the choice.

At the very bottom of the Summary dialog box, you'll see a check box marked Download Package Files Only. As it suggests, this will download but not install the packages. If you then select the package for installation again in the future, you won't need to download it, and installation will be almost instantaneous (unless a newer version of the package has been released; in which case, the newer version will be downloaded and installed).

If you see an Unchanged heading in the Summary dialog box, this means that there are several system updates available that you haven't selected for installation. To install the system updates, click Cancel, and then click Edit ► Mark All Upgrades. Then click Apply again. You will then see two separate headings in the Summary dialog box: one listing the upgrades and one listing the new packages you've selected to install.

Note Of course, you can opt to ignore the fact that updates are available and simply go ahead with installation. Installing updates as soon as possible is advised but not enforced.

METAPACKAGES

Software such as the GNOME desktop actually consists of a number of programs and system libraries, rather than one single piece of software. Therefore, you might be wondering how, as just one example, you might install the KDE desktop under Ubuntu 8.04. Is it necessary to install each component's package manually?

In theory, dependency management should be able to help, and you should be able to select one key part of the KDE desktop system, such as the Konqueror file browser, and have the Synaptic Package Manager take care of the rest. After all, Konqueror will be dependent on other KDE packages.

Alas, this rarely works in reality. Installing Konqueror in this way will indeed install much of the KDE desktop suite, but not everything. Konqueror isn't reliant on Kate, for example, which is the default text editor under KDE. And although the packages will be installed, there's no guarantee that they'll be configured to work correctly as a desktop environment.

Metapackages provide the solution. These are packages that contain configuration files to ensure the full range of software is installed and configured correctly, and they also have extensive lists of dependencies that include the complete set of packages for the software in question. (The metapackage for KDE is *kdebase*, but if you want the full Kubuntu experience, you should install the *kubuntu-desktop* metapackage.)

Alongside desktop suites, other examples of Ubuntu metapackages include the OpenOffice.org office suite, where the metapackage ensures all the components of the suite can be easily installed, and the X.org graphical subsystem. To see what metapackages are available, simply search for *metapackage* using the Synaptic Package Manager.

Removing Software

To remove a piece of software, search for it by name, click the check box alongside it, and then select Mark for Removal. This will remove the software but leave behind any configuration files it created. This means you can install it again in the future, and it will function as it did before removal. However, you can also select Mark for Complete Removal, which will remove the configuration files.

As with installing software, the Synaptic Package Manager will attempt to manage dependencies when you remove software, but in this case, it will enforce the removal of any software that explicitly relies on that software.

Often, the solution is simply not to remove the software package. After all, modern hard disks have huge capacities, and it's unlikely the package will take up much room.

USING ADD/REMOVE APPLICATIONS

You can use another graphical package manager, accessed by selecting Applications ► Add/Remove Applications from the Ubuntu desktop, to easily install or uninstall programs without worrying about package dependencies.

The Add/Remove Applications window shows a list of application items. Each item has a check box that denotes whether the listed application is installed, with a brief description and a popularity rating. By selecting an item, you can view more information about the application at the bottom of the window. Selecting an item to install or uninstall is as easy as marking or unmarking the item's check box.

To find the software you are looking for, you can filter the items in the list by selecting a software category in the left column. The category choices are All, Accessories, Education, Games, Graphics, Internet, Office, Other, Programming, Sound & Video, System Tools, and Universal Access. You can also select which type of software to show using the Show drop-down list or by entering a query in the Search box. The software type choices are All Available Applications, All Open Source Applications, Supported Applications, Third Party Applications, and Installed Applications Only. If you're not sure which packages to try first, you can click the Popularity tab to rank items from five stars to one star.

Once you are satisfied with your choices, click Apply Changes to proceed. You'll see a confirmation dialog box, listing the items to be installed or uninstalled. Click the Apply button to confirm the changes, and you will be prompted for your password. After you've entered your password, the installation or uninstallation process will commence. The system will notify you when the process has been completed.

Package Management from the Command Prompt

Synaptic is one of the best examples of package management programs around, and there's little reason to shun it and choose to install packages from the command line. However, you may find occasions to use `dpkg` or the APT tools. For example, if you're already working at the command line, then this method is quicker than starting up the Synaptic Package Manager.

Using `dpkg`

The most basic package-manipulation command is `dpkg`. `dpkg` allows you to perform a lot of package-related tasks, such as building packages from scratch. Here, we'll look at simple package installation, removal, and query functions.

Note `dpkg` requires superuser powers to install or remove software, so the command must be preceded with `sudo`. If you simply wish to query the package database, `dpkg` can be run without superuser powers. The same is true of the APT tools discussed later in this chapter.

Installing Packages

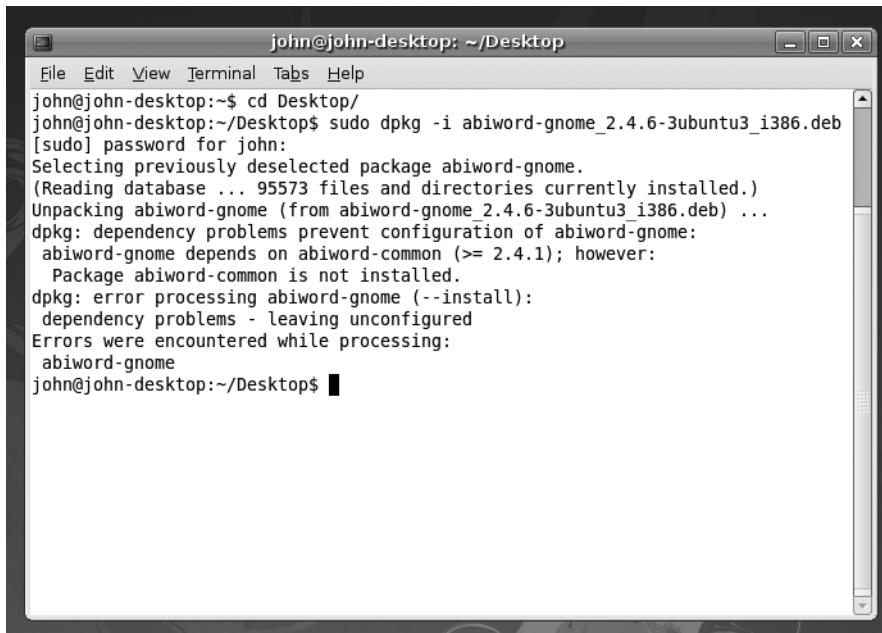
`dpkg` is useful when you've already downloaded a specific `.deb` package and would like to install it. Here is the command:

```
sudo dpkg -i packagename.i386.deb
```

You must specify the entire filename, rather than just the name of the program.

Note Be careful when downloading `.deb` package files. Not all of them are guaranteed to be 100% compatible with Ubuntu, because the `.deb` package format is used within a variety of distros, such as Debian and Xandros. Your first choice should be to download packages that are specifically created for your installed version of Ubuntu. These will probably have the word `ubuntu` in their filenames. If these are not available, you should try downloading those created for Debian. Package files created for other distros might work, but should be tried only as a last resort.

`dpkg` is quick and dirty, and although it will warn you about any dependency issues, it will still go ahead and install the package. After installation, it will run the package's configuration scripts. But if there are missing dependencies, it won't be able to configure the program to work on your system, because it probably won't be in a usable state, as shown in the example in Figure 28-3.



```
john@john-desktop: ~/Desktop
File Edit View Terminal Tabs Help
john@john-desktop:~$ cd Desktop/
john@john-desktop:~/Desktop$ sudo dpkg -i abiword-gnome_2.4.6-3ubuntu3_i386.deb
[sudo] password for john:
Selecting previously deselected package abiword-gnome.
(Reading database ... 95573 files and directories currently installed.)
Unpacking abiword-gnome (from abiword-gnome_2.4.6-3ubuntu3_i386.deb) ...
dpkg: dependency problems prevent configuration of abiword-gnome:
  abiword-gnome depends on abiword-common (>= 2.4.1); however:
    Package abiword-common is not installed.
dpkg: error processing abiword-gnome (--install):
  dependency problems - leaving unconfigured
Errors were encountered while processing:
  abiword-gnome
john@john-desktop:~/Desktop$
```

Figure 28-3. *dpkg* will warn about missing dependencies but will still install the package.

If this situation arises, it's up to you to install the dependencies that `dpkg` lists (several packages can be specified with the `dpkg` command, in any order; `dpkg` will sort them out itself).

Of course, some of these will have their own dependencies, which will also need to be installed. This cascade situation is known as *dependency hell*, as discussed earlier in the chapter, and is the main reason why the APT system was invented. As you'll see in the next section, APT effortlessly handles dependency issues.

If the dependencies aren't met after a `dpkg` installation, whenever you run the Synaptic Package Manager or attempt to use the APT tools, you'll be told of "broken" packages or unmet dependencies. APT will refuse to install any other software until the problem is fixed. Synaptic Package Manager will attempt to fix the problem automatically—the missing packages will be selected for installation and will be installed alongside any software you choose. At the console, you can type `sudo apt-get -f install`. This will install all of the missing dependencies on the system.

Tip Within the Synaptic Package Manager, you can click the Custom Filters button at the bottom left and then click the Broken entry in the filter list in order to see any packages that have unmet dependencies.

Uninstalling Packages

To remove a package, type the following:

```
sudo dpkg -r packagename
```

Note that you simply need to type the name of the program, without its version number or the `.i386.deb` file extensions.

In this case, `dpkg` is slightly better behaved than when installing software. If there are any reverse dependencies (other packages that depend on the one you're trying to remove), you'll be stopped in your tracks with a couple of error messages. You'll need to resolve the reverse dependencies first; of course, they might also have their own reverse dependencies. Welcome back to dependency hell!

Note The `dpkg -r` command will remove the package but leave behind its configuration files. This is handy if you install the software again in the future. In order to remove the configuration files in addition to the software, type `sudo dpkg -P packagename`.

Querying Packages

`dpkg` includes a couple of query facilities that display details about packages. You can find out if a package is installed by typing this:

```
dpkg -l packagename
```

If you want to find out every bit of information about an installed package, including what dependencies it has, use the following command:

```
dpkg -s packagename | less
```

This example pipes the output of `dpkg` into `less` so you can read it more easily, because it's likely to fill several terminal window screens.

You can also use `dpkg` to query an installation file you've just downloaded:

```
dpkg -I packagename.i386.deb | less
```

All said, `dpkg` is an often undervalued tool that's capable of some handy low-level package management tasks. Take a look at its man page to learn more.

Using the APT Tools

`dpkg` is the only option if you want to install a package file you've just downloaded. However, if you wish to utilize software repositories at the command line, you'll need to use the APT tools `apt-get` and `apt-cache`. These still use `dpkg` in the background to install and remove packages, but they also feature intelligence to handle dependency management.

Note If, while using `dpkg` or APT, you get an error message along the lines of, “Can’t get a lock,” make sure that the Synaptic Package Manager or Update Manager program isn’t open. Only one piece of software can access the package database at any one time.

Installing and Removing Packages

The most basic APT command is `apt-get`. You can use this command to install or remove packages contained within the repositories as follows:

```
sudo apt-get install packagename  
sudo apt-get remove packagename
```

You should specify the program name without the version number. You can specify two or more programs to be installed and/or removed at the same time. Just separate the package names with a space:

```
apt-get install package1 package2 package3
```

Note On very rare occasions, there are several different versions of the same software in the repository. In such cases, version numbers sometimes are used. But this isn't something you should worry about.

To install the `links` web browser, for example, you just need to type the following command:

```
sudo apt-get install links
```

Figure 28-4 shows the results. As you can see, `apt-get` will check dependencies, download the software, and then install it. It's a much better way of working compared with `dpkg`.

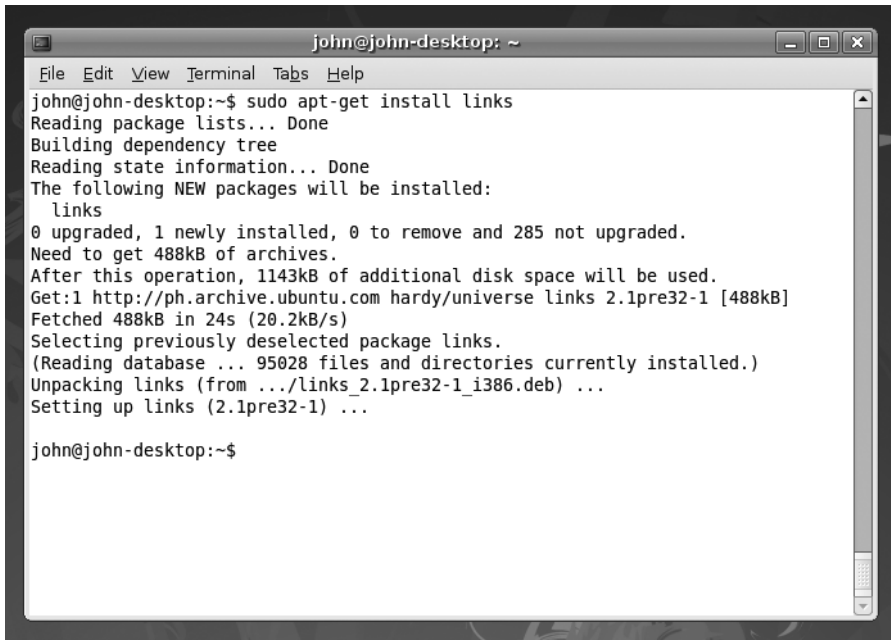
A screenshot of a terminal window titled 'john@john-desktop: ~'. The terminal shows the command 'sudo apt-get install links' and its output. The output includes: 'Reading package lists... Done', 'Building dependency tree', 'Reading state information... Done', 'The following NEW packages will be installed: links', '0 upgraded, 1 newly installed, 0 to remove and 285 not upgraded.', 'Need to get 488kB of archives.', 'After this operation, 1143kB of additional disk space will be used.', 'Get:1 http://ph.archive.ubuntu.com hardy/universe links 2.1pre32-1 [488kB]', 'Fetched 488kB in 24s (20.2kB/s)', 'Selecting previously deselected package links.', '(Reading database ... 95028 files and directories currently installed.)', 'Unpacking links (from .../links_2.1pre32-1_i386.deb) ...', 'Setting up links (2.1pre32-1) ...', and finally 'john@john-desktop:~\$'.

Figure 28-4. You can use `apt-get` to install, remove, and update packages at the command line.

It's a similar situation when it comes to uninstalling software. For example, suppose you tried to remove the Evolution e-mail client, like so:

```
sudo apt-get remove evolution
```

`apt-get` would also mark for removal `evolution-exchange` and `evolution-plugins`, two packages that depend on the e-mail client. But before doing anything, it will tell you what it is about to do and ask you to confirm it.

Similarly, if you tried to install the AbiWord word processor, like so:

```
sudo apt-get install abiword-gnome
```

you would be informed that an additional package needs to be installed: `abiword-common`. It will be automatically added to the list of packages that are to be installed.

As with the Synaptic Package Manager, `apt-get` will also list suggested and recommended packages that will complement the software you wish to install but aren't vital. However, if you wish to install those packages, you'll need to do that later, in a separate `apt-get` command.

Tip An alternative to the command-line APT tools is `aptitude`. This can be used like APT tools such as `apt-get`, but can also take into account suggested and recommended packages. For more information, see its man page.

Querying Packages and Repositories

To search the repository databases for particular software packages, use the `apt-cache` command:

```
apt-cache search packagename
```

Both descriptions and package names are searched. The list of results will show the package name on the left and a description on the right. Sometimes, the results can scroll off the screen, so it's useful to pipe the output into `less`:

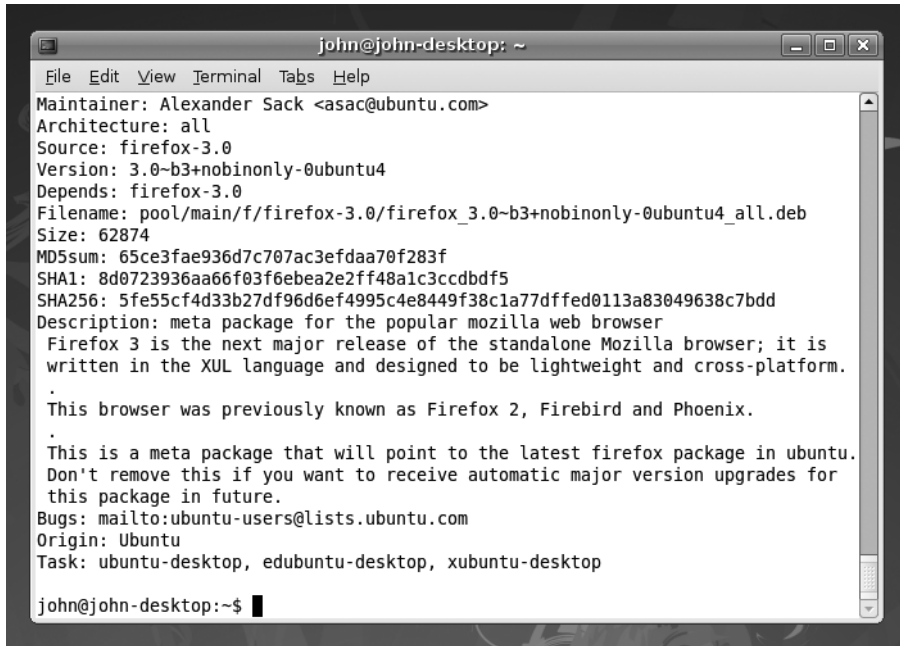
```
apt-cache search packagename | less
```

You can also find out about dependencies and suggested packages using `apt-cache`. Once again, it's a good idea to pipe the output into `less`, because the dependency list may run off the screen:

```
apt-cache depends packagename | less
```

You can read the program description for a package file by typing the following (see Figure 28-5 for an example):

```
apt-cache show packagename | less
```



```

john@john-desktop: ~
File Edit View Terminal Tabs Help
Maintainer: Alexander Sack <asac@ubuntu.com>
Architecture: all
Source: firefox-3.0
Version: 3.0~b3+nobinonly-0ubuntu4
Depends: firefox-3.0
Filename: pool/main/f/firefox-3.0/firefox_3.0~b3+nobinonly-0ubuntu4_all.deb
Size: 62874
MD5sum: 65ce3fae936d7c707ac3efdaa70f283f
SHA1: 8d0723936aa66f03f6e2e2ff48alc3ccdbdf5
SHA256: 5fe55cf4d33b27df96d6ef4995c4e8449f38c1a77dffed0113a83049638c7bdd
Description: meta package for the popular mozilla web browser
 Firefox 3 is the next major release of the standalone Mozilla browser; it is
 written in the XUL language and designed to be lightweight and cross-platform.
.
 This browser was previously known as Firefox 2, Firebird and Phoenix.
.
 This is a meta package that will point to the latest firefox package in ubuntu.
 Don't remove this if you want to receive automatic major version upgrades for
 this package in future.
Bugs: mailto:ubuntu-users@lists.ubuntu.com
Origin: Ubuntu
Task: ubuntu-desktop, edubuntu-desktop, xubuntu-desktop
john@john-desktop:~$

```

Figure 28-5. You can query any package using `apt-cache show` to learn more about it.

None of these commands makes a distinction between installed and uninstalled packages—you're accessing the details held in the repository database.

Before searching for packages, it's a good idea to make sure you have the latest package lists in the database (the equivalent of clicking the Reload button in the Synaptic Package Manager). To refresh the repository data, use this command:

```
sudo apt-get update
```

Updating the System

`apt-get` can also perform various types of system updates. To update all the packages on your system to the latest versions contained within the repositories, type the following:

```
sudo apt-get upgrade
```

This is the command-line equivalent of using the Update Manager function of the Ubuntu desktop.

To upgrade the system to the latest version of Ubuntu (if there is one), type this:

```
sudo apt-get dist-upgrade
```

Updating your system can take a long time, depending on the number and size of files to be downloaded. In addition, each package will need to configure itself during installation, and this can also take a long time.

DECODING PACKAGE FILENAMES

Although the filenames of packages might seem like cryptic mumbo-jumbo, they actually tell you a great deal about the file. Let's take a look at the package file of the Eye of GNOME image viewer to explain this:

```
eog_2.21.92-0ubuntu1_i386.deb
```

The first element of the filename is the name of the program. In this case, Eye of GNOME has been abbreviated to `eog`. Abbreviations like this are quite common, because they decrease the length of the filename. But it's important to note that they will be consistent. For as long as Eye of GNOME is supported as a package under Ubuntu, its package filename will always begin with `eog`.

Following the name of the package is the version number of the program in question: `2.21.92-0`. This is almost always the version number that will appear if you click Help ► About when the program is running and is the version number decided on by the developer who created the software.

After the version number is the word `ubuntu`, which indicates that this is a package that has been created specifically for the Ubuntu distribution of Linux. Then you see the build version number of the package: `1`. This is Ubuntu's own version number, indicating how many times the package has been built (created) by the Ubuntu team. Sometimes, it's necessary to release an updated build of the same version of a program in order to correct an error that was accidentally introduced in the last build version. Sometimes, the program is patched by the Ubuntu team to support a new function.

After Ubuntu's build version number is the platform on which the package will run. In this case, `i386` indicates that the package will run on all x86-based processors, from 80386 upward (the 486, Pentium, Pentium II, AMD processors, and so on). Sometimes, you might see `i686`, which means that the package has been optimized for Pentium Pro chips and above (Pentium II, III, IV, and AMD's Athlon range of chips). If the package is created for 64-bit desktop processors, then `amd64` will appear there.

Optimized versions of packages for particular processors are used only when they might bring a performance boost. For example, there are `i686` versions of the Linux kernel and the `libc6` library. Even ordinary programs, like OpenOffice.org, can be optimized for their architectures, but the majority of packages that are used under Ubuntu have the `i386` designation.

Managing Software Repositories

It's unlikely that, in general use, you'll need to add, remove, or otherwise manipulate the list of software repositories, above and beyond what we've described in Chapter 18, when you enabled the Skype repository.

The list of repositories is held within the `/etc/apt/sources.list` file and also in the `/etc/apt/sources.list.d/` directory. You can administer this either by using the Software Sources program or by directly editing it. Although it's not a complicated file to understand, we don't advise making manual edits; in most cases, the Software Sources program will do all you need. However, we explain both approaches in the following sections.

Using Software Sources

The Software Sources program can be found on the System ► Administration menu. Although you can use it to add and remove third-party repositories, it's designed to let you tweak settings relating to the official repositories, which are set up by default.

The program contains five tabs, which offer the following functionality:

Ubuntu Software: This tab lets you choose the repository components. You can choose to activate Main (Canonical-supported open-source software), Universe, Multiverse, Restricted, and Source Code components. Additionally, you can choose whether Ubuntu connects to a regional repository server (that is, one in or near your country) or the main Ubuntu server. Simply select the one you want from the Download From drop-down list, or try the Select Best Server button for automatic configuration. Connecting to a regional server is likely to result in faster service. If you're outside the United States, you can also opt to connect to the Server for United States.

Third-Party Software: This tab lets you add your own selection of repositories. These can be online or on a CD/DVD-ROM. When you click Add, you'll be asked to supply the APT line, which should be the line as it would appear in the `/etc/apt/sources.list` file, discussed in the next section. However, once a line has been added, you can double-click it in the list to see a more user-friendly dialog box, which splits the category and component fields into separate text boxes.

Updates: This tab lets you select the types of updates you would like to download. Effectively, it lets you choose a repository category. You can choose to connect to Proposed Updates and Backported Updates, in addition to Security Updates and Recommended Updates, as shown in Figure 28-6. You can also set when to check for updates, and whether you want to be informed of all new releases of Ubuntu or just the long-term support releases every few years. Additionally, you can select to automatically install security updates without being prompting first. This is not a bad idea, considering many are considered vital for the safety of your system.

Note On the Updates tab, it's not possible to add or remove the Main category. It's assumed that you'll want to be connected to that in order to download new software.

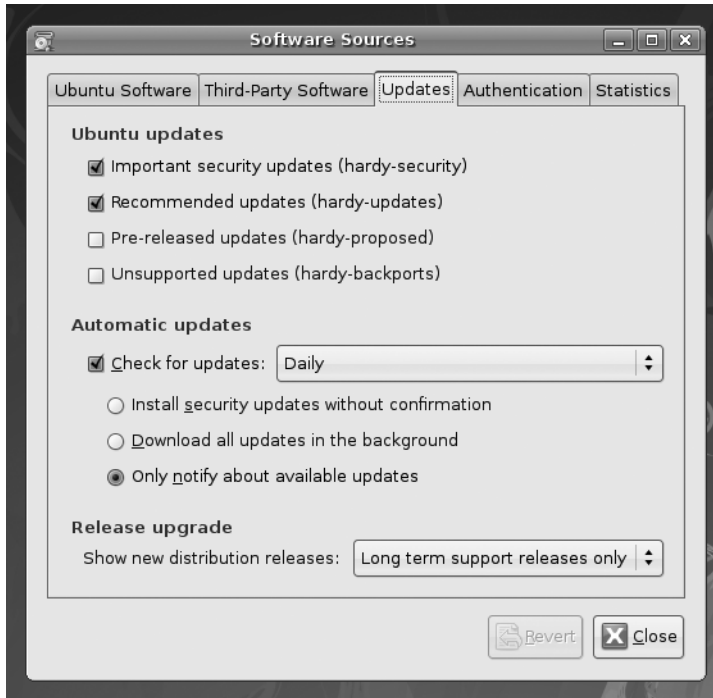


Figure 28-6. The *Updates* tab lets you choose which repository categories you wish to use.

Authentication: Package files within repositories are usually digitally signed by their creators. This is a way of proving that they haven't been tampered with. Your computer can check the digital signature, but it needs a copy of the signing key for the relevant repository. If the key is missing, you'll be warned within Synaptic Package Manager that the package can't be authenticated when you try to install it (although it's still possible to install the package after this). Two signing keys are added to Ubuntu by default—one for the installation DVD-ROM and one for the online archive—but you can add more, such as those for third-party repositories that you've added manually. The Authentication tab lets you do this. Using a web browser, download the key file from the server (right-click the file and select Save As), and then click the Import Key File button. In the Open dialog box, select the file and click OK. The key will then be instantly imported.

Statistics: Here, you can opt into an attempt by the Ubuntu developers to monitor which packages are most popular. This will help shape future releases of Ubuntu and also improve the rank of the applications you choose in the Add/Remove Applications program. Simply put a check in the Submit Statistical Information box if you would like to participate.

Adding/Removing a Repository at the Command Line

The `/etc/apt/sources.list` file is at the heart of the APT system and contains details of all the package repositories you're subscribed to, as well as the individual categories within each repository.

You can edit the file within `vim` by opening a terminal window (Applications ► Accessories ► Terminal) and typing the following:

```
sudo vim /etc/apt/sources.list
```

The `sources.list` file usually contains two types of entries: those beginning with `deb`, which indicate a standard repository containing binary files, and those beginning with `deb-src`, which indicates a source file repository. `deb-src` entries are largely for developers.

Here's an example line from `sources.list` on a test system:

```
deb http://us.archive.ubuntu.com/ubuntu/ hardy main restricted
```

As you can see, the first component of the line is the address of the server. Then the repository category is listed: `hardy`. If the server offered updates, this might read `hardy-security` or `hardy-updates`. Following this, the repository components are listed: `main` and `restricted`.

A hash symbol (`#`) at the beginning of a line in `sources.list` means that it's ignored. This can be useful for disabling a repository in a situation where you don't necessarily want to remove it from the file. In addition, as you can see within the file, some explanatory comments from Ubuntu developers are inserted into the file and are preceded by a hash symbol, so that APT doesn't attempt to interpret them.

Adding a new repository is easy. For example, to add the Skype repository, in order to download the Skype VoIP software, switch to insert mode within `vim` (by typing `i`), and then type the following:

```
# added by Keir, 4 May 08
deb http://download.skype.com/linux/repos/debian/ stable non-free
```

As you can see, we've added a comment to identify who added the line and when.

You'll notice something interesting about the actual `deb` line. The Skype server's repository category and components aren't like the others in the `sources.list` file. The category is `stable`, and the component is `non-free`. In this case, the Skype server uses the Debian method of naming repository categories and components, which is different from Ubuntu's way.

Once you've added a new repository, don't forget to refresh your local list of packages. If you don't do this, APT won't be aware of the software contained in the new repository. Refresh the list by typing the following at the command prompt:

```
sudo apt-get update
```

This should also be done after you remove or disable a repository within `sources.list`.

Installing from Source

Back in the old days of Unix, the only way to install many software packages was from source code, a process known as *compiling*. This was because most people edited the source code themselves, or at least liked to have the option of doing so. Nowadays, innovations such as the Debian package management system make compiling all but redundant for the average user. But knowing how to compile a program from source is still a good Linux skill to have. In some cases, it's your only option for installing certain programs, because you may not be able to find a packaged binary.

Program compilation is usually handled at the command prompt. It's not the kind of thing you would do via a GUI program.

Installing the Compiler Tools

Before you can compile from source, you need to install several items of software: the `make` program, which oversees the process of creating a new program, and the GNU Compiler Collection (GCC), which does the hard work of turning the source code into a binary. In addition, if the software relies on certain library files, you'll need to install development (`dev`) versions of them, as well as the libraries themselves if they're not already installed. For example, if you're compiling a program to run under the GNOME desktop, you'll need development versions of the GTK2+ libraries.

Under Ubuntu, it's possible to install all the program-compilation tools you need by installing the `build-essential` metapackage. You can use the Synaptic Package Manager or the following `apt-get` command at the command prompt:

```
sudo apt-get build-essential
```

Unpacking the Source Tarball and Solving Dependencies

Let's take a look at installing a program from source. Dillo is a stripped-down web browser that's designed for speed and small file size. It's a fun little program that's good to have around in the event of your main browser developing a glitch that you can't fix. The Dillo home page is www.dillo.org, so head over there, and choose to download the latest version of the source code.

Note Okay, you got us. If you use `apt-cache` or the Synaptic Package Manager to look through the repositories, you'll see that Dillo is available as a ready-to-install package. But Ubuntu's package repositories are so comprehensive that, frankly, we couldn't find anything to demonstrate program compilation that wasn't already in there!

The first thing to do is to unpack and uncompress the tarball (if you wish to learn more about the `tar` command, see Chapter 31):

```
tar jxf dillo-0.8.6.tar.bz2
```

Of course, you should replace the filename with that of the version you downloaded.

Next, you'll need to switch into the source code directory and take a look at the `README` file. This will tell you what dependencies Dillo has and also any caveats you may need to take into account in order to compile Dillo on a Linux system:

```
cd dillo-0.8.6
less README
```

Note Unlike binary packages, source code is rarely designed with one specific Linux distribution in mind. For example, Dillo is able to compile on all types of Unix, including Linux, Solaris, BSD, and others. With a little work, it might even be possible to compile it under Windows!

First, you see that Dillo needs the `glib` libraries. This is a given on nearly all Linux systems, but in order to compile, Dillo will need the `dev` version of `glib`, which isn't part of the default Ubuntu installation.

Next, you read that it also needs the `GTK+ 1.2` libraries. These are present on the majority of GNOME-based Linux desktop systems, but once again, the `dev` versions will need to be installed.

Beneath that in Dillo's list of requirements is support for JPEG and PNG image formats, which are definitely installed on the average Linux system, and the `Wget` download tool,

which is also included with most versions of Linux (although it's a good idea to use the Synaptic Package Manager or `apt-cache search` to check that it's installed).

After finding out about dependencies, you should scroll down the README to look for any notes about compiling under Linux. It turns out there might be some issues with older 2.4 versions of the Linux kernel, but Ubuntu uses 2.6, so this isn't an issue.

So, in short, before you can compile Dillo, you need to install dev versions of the `glib` and `GTK+` 1.2 libraries. You can install these via the Synaptic Package Manager or `apt-get`. It will help cut down the search results if you realize that system library packages under Ubuntu are usually prefaced with `lib`. So, search for the dev versions of `libgtk` and `libglib`. Doing so on our test system returned three likely packages: `libglib1.2-dev`, `libglib2.0-dev`, and `libgtk1.2-dev`. There are two `libglib` entries, because our system has both `glib2` and the older `glib1.2`. To ensure compatibility, we decided to install dev versions of both. Since you're working at the command prompt, install the packages via `apt-get`:

```
sudo apt-get install libglib1.2-dev libglib2.0-dev libgtk1.2-dev
```

As soon as we typed this, it turned out that `libgtk1.2-dev` came with a host of dependencies in the form of X server dev libraries. The reasoning is that if the `GTK+` dev library files are needed, these other libraries are often needed, too. Whatever the case, there's no harm in installing them.

Compiling

Now comes the exciting process of compiling the program! This is done via three commands, issued in sequence:

```
./configure  
make  
sudo make install
```

The first command starts the `configure` script, created by the Dillo programmer, which checks your system to ensure that it meets Dillo's requirements. In other words, it checks to make sure the `glib` and `GTK+` libraries are present. It also checks to make sure you have the correct software that's required to actually compile a program, such as `GCC` and `make`.

It's when the `configure` script is running that something is most likely to go wrong. In that case, more often than not, the error message will tell you that you're missing a dependency, which you must then resolve.

Note Some `configure` scripts are very thorough and check for components that the program you're trying to install might not even need. Because of this, you shouldn't worry if, as the text scrolls past, you see that various components are missing. Unless `configure` complains about it, it's not a problem.

The next command, `make`, takes care of the actual program compilation. When it's run, the screen will fill with what might look like gibberish, but this is merely the output of the GNU compiler. It provides a lot of valuable information to those who know about such things, but you can largely ignore it. However, you should keep your eyes peeled for any error messages. It's possible that the `configure` script did not check your system thoroughly enough, and you might be missing an important system component; in which case, `make` will halt.

Note We saw an error message at the end of the `make` session when compiling Dillo. We were able to follow up with the `sudo make install` command, which also reported error messages. However, we found that Dillo worked fine! The moral of the story is that software compilation is something of a black art, with error messages designed for programmers, and not all error messages are fatal.

Alternatively, the program simply might not be able to compile on your system without some tweaking to the `Makefile` (the file that `make` uses). If such a situation arises, the best plan is to visit the web site of the developer of the software and see if there's a forum you can post to. Alternatively, check if the developer has an e-mail address you can contact to ask for help.

Eventually, the compilation will stop with a number of exit messages. Then the final command must be run: `make install`. This needs to be run with superuser powers, because its job is to copy the binary files you've just created to the relevant system directories. In addition, any documentation that comes with the program is also copied to the relevant location on your system.

Once the three commands have completed, you should be able to run the program by typing its name at the command prompt. If you've been playing along at home and have compiled Dillo, you can run it by typing `dillo`. Figure 28-7 shows Dillo running under Ubuntu.

Note You'll probably need to add your own icon for Dillo to the desktop or Applications menu. Source packages are usually designed to be installed on any version of Unix running a variety of desktop managers. In the past, it was difficult for the developer to know where to create desktop shortcuts, but now organizations like freedesktop.org (<http://freedesktop.org>) are standardizing the process.

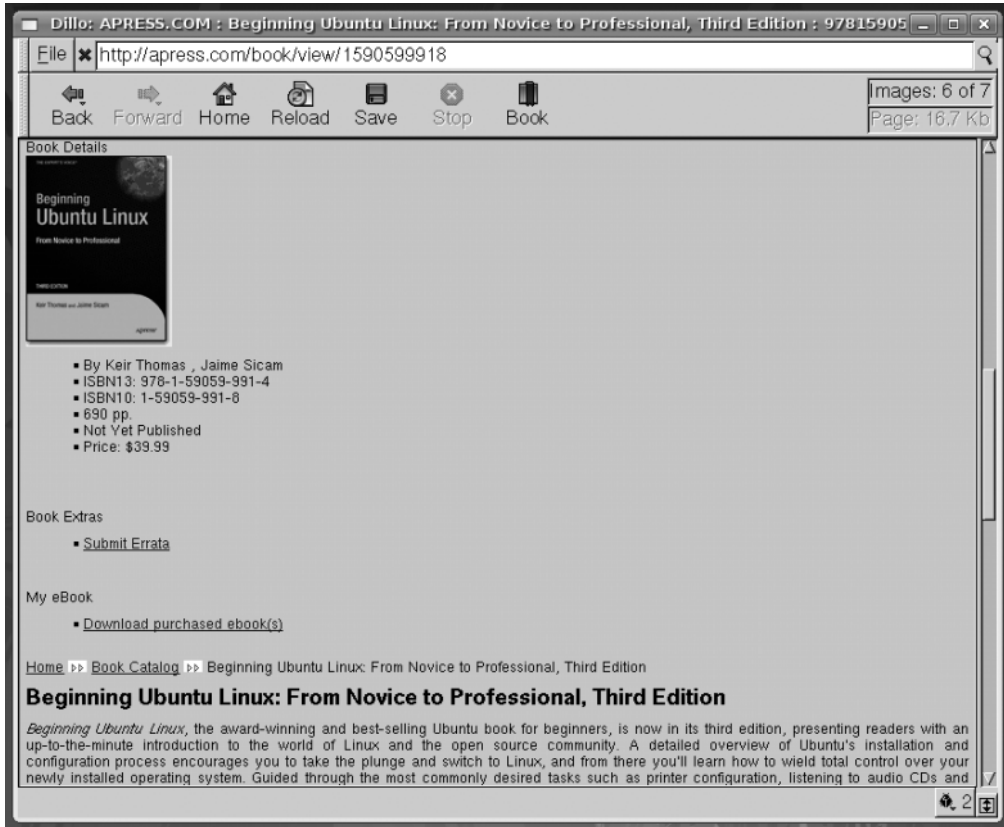


Figure 28-7. *Dillo in action—a certain satisfaction comes from compiling a program from source code.*

Summary

This chapter described how to install software under Ubuntu. We've looked at how this differs from Windows software installation, and how the Debian package management system is designed to make life easier.

You learned how to use the Synaptic Package Manager to install software under the GUI, and how to use the `dpkg` and `APT` tools to install software at the command-line prompt. Finally, we looked at how programs can be compiled from their source code, which is a fundamental process for all versions of Linux.

In the next chapter, we'll look at how to administer the user accounts on the Ubuntu system.