Chapter 20

Adding, Removing, and Updating Software

If you looked around the universe before settling on Ubuntu Linux, one of the things that you were sure to notice is that "billions and billions" of Linux distributions are available, each with its own installer, favorite desktop environment and/or X Window system window manager, set of core applications, and some way of updating, expanding, and maintaining the software that comes with the distribution. When you come right down to it, the last item is the most significant aspect of a Linux distribution aside from the size and involvement of its user and development communities. There's actually an interesting loop between ease-of-use/ease-of-maintenance and the size of the user community. A Linux distribution that makes it easy to keep existing software up to date, install new software, and figure out what's on your system in the first place is a Linux distribution that more people are apt to use. (This assumes that someone is actually keeping the distribution up to date.)

As discussed in Chapter 1, I find the Ubuntu Linux community the most exciting, dynamic, and energetic Linux community that I've ever encountered. The Ubuntu community is a committed community of both users and developers. Updates and bug fixes to existing software are frequent, which is a good thing. A vast pool of software is available for easy download and installation.

Access to new and updated software is similarly easy. Ubuntu's Update Manager automatically lets you know when updates are available to software that's installed on your system, and keeping your system up-to-date requires three or four mouse clicks. Not too shabby! The `dpkg`, `apt-get`, `aptitude`, and `synaptic` tools make it almost easy to install new software, and Synaptic Package Manager makes it easy to search for software that's relevant to your interests. Who could ask for anything more?

The key to all of this simplicity and ease-of-use is the excellent DEB (for Debian) package format, developed for the Debian Linux distribution and therefore used by Ubuntu. The DEB package format put other software package formats, such as RPM, to shame — I personally would appreciate some electroshock therapy to

IN THIS CHAPTER

forget the needless hours I've wasted trying to untangle twisty mazes of dependencies and requirements between various RPM packages. If you don't know what RPM is, you're lucky. Let's keep it that way.

NOTE **If you're familiar with RPM-based Linux distributions such as Fedora Core, Mandriva, Red Hat, or Yellow Dog, you're probably thinking, "Silly Human, yum solves most RPM problems!" Although this is probably true in a court of law, I don't feel that providing additional layers of syntactic sugar on top of a frustrating package management system such as RPM is the "right thing" or an actual solution. I prefer using smarter package formats such as DEB and smarter utilities such as the ones discussed in this chapter that were designed to do the right thing in the first place rather than continually shaving JoJo the dog-faced boy to make him look more human.**

TIP **If you must deal with packages that were created for other Linux distributions and are therefore in other package formats, see the section entitled "Converting Packages from Other Package Formats" later in this chapter for information about converting package.**

Tools such as `dpkg`, `apt-get`, and `aptitude` are knee-deep in online manual pages and Web sites that explain every conceivable combination of command-line options, so I'm not going to bore you by repeating those in this chapter. This chapter explains how to do common tasks using these tools, focusing on helping you get your work done rather than on duplicating existing reference material. The first section of this chapter provides an overview of the package management utilities used on Ubuntu systems (and thus discussed in this chapter) so that you have a clear idea of "what does what" as you read through this chapter. The next section explains how the Ubuntu software that you can download from the repositories is organized, and provides some tips about other, unofficial repositories that you might find useful. After this, the hands-on sections of this chapter begin by explaining how to find out what's on your system, figure out what package provided specific files that are already on your system, and how to figure out what package you might want to install if you're looking for a specific file. This is most easily done from the command line, so that's our focus in this section.

Subsequent sections of this chapter focus on using the `apt-get` (command-line), `aptitude` (terminal-oriented), and `synaptic` (fancy graphics) tools to locate and install new packages on your system. The last two sections discuss how to use the Ubuntu Update Manager to keep your system up-to-date and how to identify and remove packages on your system that are no longer required because they were installed to satisfy dependency requirements for packages that are no longer installed.

Overview of Ubuntu Package Management Software

A standard Ubuntu Linux installation provides several different utilities for managing and querying Ubuntu software packages, ranging from low-level command-line tools to tools with sophisticated graphical interfaces. The basic user-level tools provided as part of a standard Ubuntu installation are the following:

- `apt-get`: A command-line utility that provides subcommands which enable you to install, remove, and manage packages on your system, both as individual packages and as components of a distribution.
- `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 quasi-graphical, menu and screen-oriented user interface and the ability to install, remove, query, and manage packages from the command-line.

- `dpkg`: The basic Ubuntu and Debian command-line tool for installing, removing, querying, and generally managing packages. This utility uses even lower-level utilities, such as `dpkg-deb`, to perform package installation, removal, and manipulation, and `dpkg-query`, to search for packages, but these lower-level packages are not discussed in this chapter. Although handy to use directly .0001% of the time, the rest of the time, I find it more convenient to think of the capabilities provided by these low-level utilities as functionality that is provided by the `dpkg` utility itself.
- `dselect`: A terminal-oriented front-end to the `dpkg` utility that provides quasi-graphical menus and an interactive display within the context of an xterm, GNOME Terminal, or other Ubuntu command-line environment. The `dselect` utility also accepts command-line options and arguments that you can use to initiate selected operations without selecting them from menus. This application isn't discussed in this chapter, because its functionality is available in other tools.
- `synaptic`: A graphical, X Window system tool for installing, removing, querying, and managing software packages on Ubuntu. This is my preferred package management utility for all day-to-day package installation operations.
- `update-manager`: A graphical, X Window system tool for identifying and installing updated versions of packages that are already installed on your Ubuntu Linux system.

NOTE **The `dpkg`, `apt-get`, and `aptitude` utilities all use different databases to store information about installed packages, and therefore all use different mechanisms for identifying and resolving conflicts and dependencies between software packages. The `dpkg` utility stores its information about installed and available packages in files and directories under `/var/lib/dpkg`. The `apt-get` utility stores the information that it uses in files and directories under the directory `/var/lib/apt`. The aptitude utility stores the information that it uses in files and directories under the directory `/var/lib/aptitude`. When using either `apt-get` or `aptitude`, it is important that you run either the apt-get `update` or aptitude `update` commands in order to ensure that the package state and availability information used by these applications is up-to-date. Similarly, if you want to experiment with `dselect` after having installed packages using `apt-get`, `aptitude`, or `synaptic`, you must use the dselect `update` command before using `dselect`, to ensure that `dselect` (and therefore `dpkg`) is aware of the state of all installed and available software.**

The `apt-get`, `aptitude`, and Synaptic Package Manager utilities also depend on the information about available repositories and associated packages maintained in a storage area which is generically known as the "apt cache". The apt cache actually consists of several files that, by default, are located in the directory `/var/cache/apt` on your Ubuntu Linux system.

This chapter focuses on discussing the `apt-get`, `aptitude`, `dpkg`, `synaptic`, and `update-manager` tools in the context of performing common tasks that each is best suited for, based on my experiences with them. As a fan of graphical tools that "do the right thing," most of the examples of installing packages throughout this book use the Synaptic Package Manager tool, which I find both powerful and easy to use. In general, however, the mantra for this chapter is "use the right tool for the job."

NOTE **Another popular graphical interface for package management on Ubuntu systems is `adept`, which is a graphical interface found on the Kubuntu Ubuntu derivative. The `adept` application is an excellent, easy to use application that is not discussed in this chapter because this book focuses on the standard Ubuntu Linux distribution, not variants such as Edubuntu, Kubuntu, or Xubuntu.**

All of the automated aspects of the package management utilities available on Ubuntu systems rely on obtaining packages from Ubuntu software repositories. The next section explains software repositories, how they are organized, the system configuration file that identifies them, and how to work with this file to ensure that your system has access to all of the latest and greatest software that is available for it.

Ubuntu Repositories and Components

A repository is exactly what the name suggests, a storage site for objects of some sort. In the case of the Ubuntu repositories, the objects stored therein are all of the source and binary packages that make up a variety of Ubuntu Linux distributions. When you download an ISO image (an image of a CD or DVD that conforms to the International Standards Organization 9660 standard for CD-ROM filesystem), the CD that you burn from this image and then use to install Ubuntu Linux provides the CD boot environment (known as *Isolinux*), an installer and the applications required to support it, a basic set of packages, and some documentation. As part of the install process, your system retrieves the majority of the Ubuntu Linux distribution from the Ubuntu repository over the Internet.

Making ISO images of Linux distributions available over the Internet is nothing new — it's the standard way of distributing most Linux distributions nowadays. Putting the Ubuntu repositories directly on the net has two primary advantages for Ubuntu fans:

- keeping the Ubuntu installation media down to a single CD and ISO image
- making every part of Ubuntu Linux instantly available to any computer that is connected to the Internet, anywhere, including the latest updates and additions

To organize its repositories along lines that are important to the Ubuntu folks, to many individuals, and to many corporations, the Ubuntu repositories are organized into four components, which are basically ways of classifying Linux software along support and licensing guidelines. The Ubuntu repositories contain four basic components:

- **main:** The main component contains binary packages (and source packages, in most cases) for Linux software that is officially supported by Canonical, Ltd., and can be freely redistributed. This does not mean that everything in the main repository component is GPL — the main repository component can include software that is distributed in binary format, such as firmware and binary fonts. The main portion of the repository is designed to provide everything that most people will need for a fully functional Linux desktop or server system that is fully supportable by Canonical, Ltd.
- **restricted:** The restricted component contains source and binary packages for commonly-used software that is not available under a completely free license. Packages in the restricted repository component are not guaranteed to be completely supportable by Canonical, Ltd., but are provided because they are necessary to use Linux on certain hardware. For this reason, some items from the "restricted" repository are included on Ubuntu installer CDs in subdirectories of `CD/dists/dist-name/restricted` and `CD/pool/restricted`. With the 6.06 (Dapper) release, restricted items include drivers for specific network interface cards (NICs) and video cards.
- **universe:** The universe repository component contains binary and source packages for the rest of the free and open source software that is commonly associated with Linux systems, but which is available under a variety of different licenses and is not guaranteed to be supported by Canonical, Ltd. There's simply too much software in the universe (pardon the expression) for Canonical to guarantee support for all of the software that it contains
- **multiverse:** The multiverse repository component contains binary and source packages for software that is not released under a license that meets the Ubuntu guidelines for free and open source software (`www.ubuntu.com/ubuntu/licensing`). This software is not supported by Canonical, Ltd., and it is your responsibility to verify that you satisfy the requirements imposed by software from the multiverse component.

NOTE **The official descriptions of the Ubuntu repositories are located at `www.ubuntu.com/ubuntu/components` — I've tried to distill them down to their essentials, but if you're a licensing fan, there's plenty of good reading there.**

One additional repository component also exists, known as the *backports* repository. The backports repository component contains updated or bleeding-edge versions of Linux software, including software that may already have an "official" version in one of the other repository components. Software in this repository component is not supported by Canonical but is provided because newer versions of existing software packages often provide some critical feature that users may require.

TIP **If you are curious about the packages that are installed or available in the Ubuntu repositories for different Ubuntu releases, see the Web page at `http://packages.ubuntu.com`. This page enables you to select the distribution that you are interested in, and then view a list of all of the packages, organized into various logical categories. It also provides a search capability so that you can search for specific packages.**

Enabling Additional Repository Components

The most common example of a situation in which you will want to access repository components other than the main and restricted components is when working with audio and video applications. The wide variety of CODECs (compressors/decompressors) used to encode digital audio and video, the platform-specific roots of many of these CODECs, and the hoops that many media companies make you jump through in order to actually play many digital audio and video formats on an Ubuntu Linux system make it necessary to push the boundaries of free software licensing.

By default, only the main and restricted components of the Ubuntu repositories are enabled when you install an Ubuntu Linux distribution. "Enabled" means that the online sources of these repository components are included in the default `/etc/apt/sources.list` file, which is the file that all of the package installation and management utilities discussed in this chapter consult when looking for new and updated software. This file is a text file that contains entries, which describe the location and name of different online repositories and the repository components that are available there. The general format of an entry in the `/etc/apt/sources.list` file is the following:

```
deb  URI distribution component-or-package(s)
```

The `deb` field identifies the package format, the URI is a universal resource identifier for the location where the repository can be found, and is typically an FTP or HTTP URL. The distribution field is typically the name of a standard Ubuntu distribution, but can be anything that is used to identify a subdirectory at the URI where associated packages can be found, and the component-or-package is one (or more) names of repository components or packages that are also used to navigate the directory tree at the URI and locate available packages. If multiple component-or-package names are provided, each identifies a different directory at the URI.

As examples, the entries in the `/etc/apt/sources.list` file for the main and restricted repository components of the 6.06 (Dapper) release of Ubuntu Linux are the following:

```
deb http://archive.ubuntu.com/ubuntu/ dapper main restricted
deb-src http://archive.ubuntu.com/ubuntu/ dapper main restricted
```

The first of these identifies the source of binary packages, while the second identifies the source of source packages. A similar pair of entries provides access to updated versions of the packages in these repository components:

```
deb http://archive.ubuntu.com/ubuntu/ dapper-updates main restricted
deb-src http://archive.ubuntu.com/ubuntu/ dapper-updates main
restricted
```

NOTE **The examples in this section reflect the `/etc/apt/sources.list` settings for the Ubuntu 6.06 (Dapper) release. If you are working with another Ubuntu distribution, you should replace the word "dapper" in these examples with the name of the Ubuntu release that you are using, such as breezy (the 5.10 release that preceded "dapper") or "edgy" (the release that follows "dapper"), and so on. You should not mix repository entries for different Ubuntu releases in an `/etc/apt/sources.list` file, because the repositories for a newer release will typically provide the same software as the repositories for older releases, and will be compiled against a set of system libraries that are found on the newer release. See the section entitled "Mixing Ubuntu and Debian Repositories" later in this chapter, for information about some cases in which you may want to have entries in your `sources.list` file that are not consistent with the Ubuntu release that you are using.**

Finally, a similar pair of entries provides access to security fixes for the packages in these repository components:

```
deb http://security.ubuntu.com/ubuntu dapper-security main restricted
deb-src http://security.ubuntu.com/ubuntu dapper-security main
restricted
```

Lines beginning with a hash mark (#) in the `/etc/apt/sources.list` file are comments. If you look through the entries in the default `/etc/apt/sources.list` file on your Ubuntu system, you will note that entries for access to the universe repository are already present in this file, but commented out. You can enable access to this repository by using `sudo` to edit this file in your favorite text editor. To do this, edit the `/etc/apt/sources.list` file using your favorite text editor, using a command like the following:

```
$ sudo emacs /etc/apt/sources.list
```

Remove the hash marks at the beginning of the lines that enable the universe repositories, add the word "`multiverse`" to the end of these lines, and save the updated file. The updated entries in a sample `sources.list` file would look like the following:

```
deb http://archive.ubuntu.com/ubuntu/ dapper universe multiverse
deb-src http://archive.ubuntu.com/ubuntu/ dapper universe multiverse
```

You could, of course, simply add these entries to the end of the lines that already provide access to the main and restricted components of the Ubuntu repositories, but I prefer to keep them separate just for clarity regarding licensing differences — and because that stays closest to the original format of the default `sources.list` file.

NOTE **Just to be perfectly clear, by suggesting that you may want to add the universe and multiverse repositories to the places where your system looks for software, I am enabling you to add software that may have licensing requirements that you or the company that you work for find odious (such as MP3-related, or video-related software), for which support is not guaranteed by the Ubuntu folks (even if you purchase support, there are no commitments to support it), and which may be difficult to get support for anywhere. If you are configuring systems for enterprise use, you should make sure that your firm is comfortable with the licenses provided with non-free software in the universe and multiverse repositories. I use all of the software discussed in this book, and am therefore my very own test case for everything this book discusses, but your mileage (and your responsibilities) may vary.**

Enabling Additional Repository Sources Using a Text Editor

As you can see from the examples in the previous section, the standard Ubuntu repositories are all hosted on systems that are in the ubuntu.com domain. This makes perfect sense, because this is the only way that the Canonical folks can guarantee the integrity of their repository and the adherence of various components and their contents to the associated licensing guidelines. However, additional repositories of Ubuntu software

are also available on the Internet (I know that's a surprise), maintained by various groups and individuals. These alternate repository sites may simply mirror the contents of the official Ubuntu repositories to "share the load," or may provide software that is not found in the official Ubuntu repositories. In the latter case, this software may provide even newer versions of popular software than is found in the backports repository component, or may provide up-to-date versions of locally maintained software. As mentioned earlier, if you are interested in working with audio and video on your Ubuntu system, you may find that these alternate repository sites provide versions of related software with bleeding-edge features that you need.

Some well-known alternate repositories for Ubuntu and the software that they provided at the time that this book was written are the following:

- FreeNX: A repository of the latest version of the FreeNX client and server software. FreeNX is an alternative to the vnc-server, vino, and vnc-viewer software. The `/etc/apt/sources.list` entries for this repository are the following:

  ```
  deb http://mirror3.ubuntulinux.nl/ dapper-seveas freenx
  deb-src http://mirror3.ubuntulinux.nl/ dapper-seveas freenx
  ```

- Multimedia Support: A repository providing Windows CODECs and other software necessary for playing various digital audio and video files on your Ubuntu system. The `/etc/apt/sources.list` entries for this repository are the following:

  ```
  deb ftp://cipherfunk.org/pub/packages/ubuntu/ dapper main
  deb-src ftp://cipherfunk.org/pub/packages/ubuntu/ dapper main
  ```

NOTE **You will have to import the GPG key for this site to access and install the packages that it provides. You can retrieve this key from `http://subkeys.pgp.net`; the fingerprint of the appropriate GPG key is 33BAC1B3.**

- Penguin Liberation Front: This repository builds and provides packages that may be subject to patents, or other mechanisms for preventing free access to software. The `/etc/apt/sources.list` entries for this repository are the following:

  ```
  deb http://packages.freecontrib.org/ubuntu/plf dapper free non-free
  deb-src http://packages.freecontrib.org/ubuntu/plf dapper free non-free
  ```

- Wine: A repository providing the latest versions of WINE, the Windows API support environment for Linux that enables you to install and run much Windows software directly on your Linux system. The `/etc/apt/sources.list` entries for this repository are the following:

  ```
  deb http://wine.budgetdedicated.com/apt dapper main
  deb-src http://wine.budgetdedicated.com/apt dapper main
  ```

If you're interested in exploring other repositories and want to use a truly cool application, see `www.ubuntulinux.nl/source-o-matic`. This site provides a Web-based sources application that generates a sources.list file for you based on your selections of available repositories.

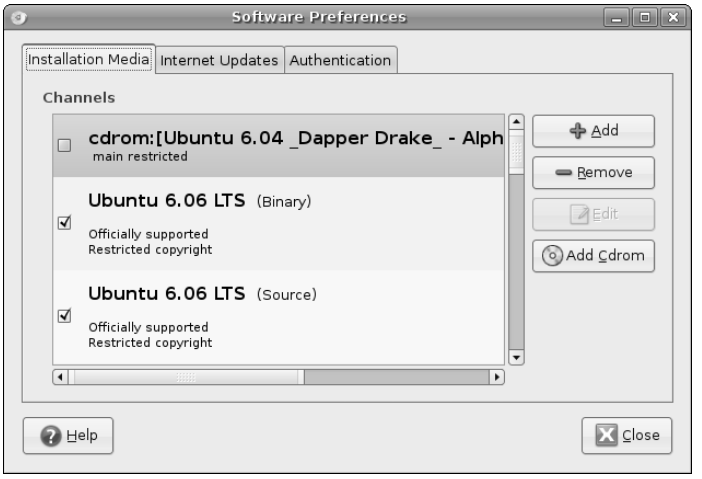
Enabling Additional Repository Sources Using the Software Properties Tool

Editing a text-format configuration file is pretty old school Linux, and is the sort of thing that you may not want to mention to your Windows or Mac OS X friends. (You'll still want to be able to do it, because it's fast and easy, but we'll keep that sort of thing our little Linux secret, OK?) For fans of graphical interfaces, Ubuntu provides a Software Preferences tool that gives you a graphical view of the entries in your `/etc/apt/sources.list` file and makes it easy for your to enable, disable, or add entries with a few mouse clicks.

To start the Software preferences utility, select the System ➪ Administration ➪ Software Properties menu item. After supplying your password to enable access to this administrative tool, the Software Preferences tool's main dialog displays, as shown in Figure 20.1.

FIGURE 20.1

The main Software Preferences dialog



As you can see in Figure 20.1, the main software preferences dialog displays all of the entries in your current `/etc/apt/sources.list` file, regardless of whether they are commented out or not. Each of these is identified as a channel. Entries that are commented out, such as the cdrom entry shown in Figure 20.1, are displayed as inactive — in other words, the checkbox beside their name is not selected. This gives you an convenient overview of all of the valid repository entries in your `/etc/apt/sources.list` file.

To add a new repository or repository component to the list of repositories that your system will search for software packages, click Add. This displays the dialog shown in Figure 20.2.

FIGURE 20.2

Adding a repository in the Software Preferences tool

The drop-down Channel menu shown in Figure 20.2 enables you to select any of the standard Ubuntu repositories and associated components — by default, only the main and restricted entries for any repository are enabled, but you can add the universe and multiverse components to any standard repository in which they are available by selecting the Community Maintained (Universe) or Non-free (Multiverse) checkboxes.

The dialog shown in Figure 20.2 also enables you to add entries for custom repositories by clicking the Custom button, which displays the dialog shown in Figure 20.3.

FIGURE 20.3

Adding a custom repository in the Software Preferences tool

Though not particularly elegant, this dialog makes it easy for you to cut and paste a repository identification entry from a mail message or browser session where you found it, and automatically add it to your `/etc/apt/sources.list` file. To return to the dialog shown in Figure 20.2 without adding a custom repository entry, click Cancel. To save a new entry after typing it in or cutting and pasting, click Add Channel, which adds the new entry and returns you to the dialog shown in Figure 20.1, which now contains an entry for the channel that you just added.

The Software Preferences tool provides three tabs that let you manage different aspects of your repositories and the update process. The first, the Installation Media tab, is the initial tab shown when you start the Software Preferences tool. The others, the Internet Updates and Authentication tabs, enable you to manage different aspects of system updates and how your system identifies valid repositories.

Figure 20.4 shows the Software Preferences tool's Internet Updates tab, which enables you to specify when your system checks for updates and how it responds to any available updates that it detects.

By default, your system automatically checks for updates on a daily basis, displaying an "Available Updates" icon in the GNOME panel at the top of your screen when updates are available (see Figure 20.28). If you are using a laptop or other system that only connects to the Internet infrequently, you may want to decrease the frequency with which your system checks for updates by selecting another value from the Check for updates automatically entry's drop-down menu, or disable this item entirely. If you disable this item, you can always check for updates manually using Ubuntu's Update manager, which is discussed later in this chapter in the section entitled "Using the Ubuntu Update Manager." If you are performing regular checks for updates and are using a system that is often (or always) connected to the Internet, you may want to select the Download updates in the background, but do not install them to make sure that packages are already available on your local system when you actually execute the update process. Similarly, you may want to select the Install security updates without confirmation option if you want to automatically retrieve security updates and install them so that your system is always secure against any known attacks or software exploits.

FIGURE 20.4

The Internet Updates tab in the Software Preferences tool

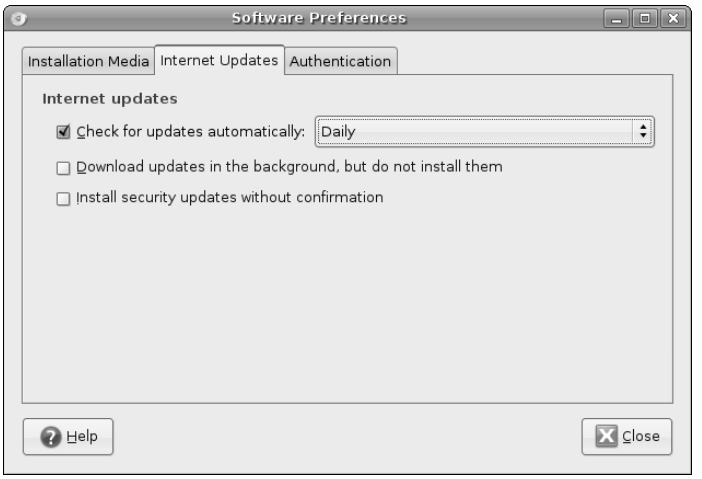
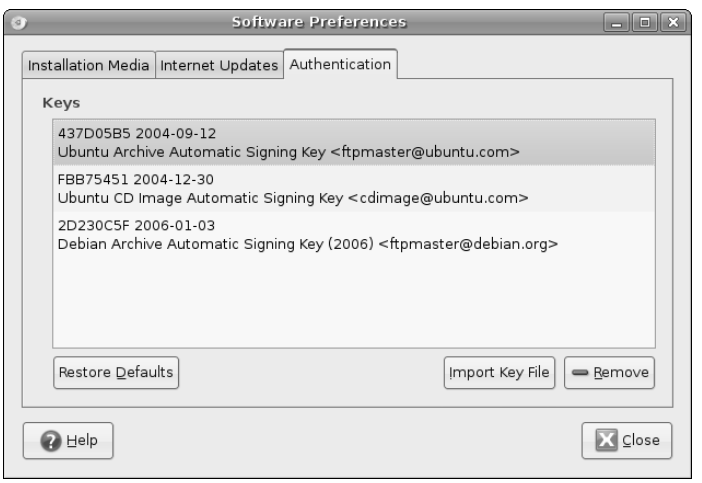


Figure 20.5 shows the Software Preferences tool's Authentication tab, which simplifies the process of importing identification keys for new repositories into the keyring used by Ubuntu's package management utilities.

FIGURE 20.5

The Authentication tab in the Software Preferences tool



Each package in any repository is signed with the key for that repository to verify its authenticity and validity. The Ubuntu package management tools will not retrieve packages from unauthorized or unidentifiable repositories. Adding the authentication key that identifies a nonstandard repository as a valid package source requires several steps from the command line, using hard-to-remember options to the `gpg` command in conjunction with the `apt-key` utility. Clicking the Import Key File button enables you to navigate to and select any file that contains an ASCII representation of a PGP authentication key and import it for use by Ubuntu's package management tools.

After making any changes that you desire to the list of repositories available to your system, the way in which system updates are performed, or the Authentication keys available to the Ubuntu package management utilities, click Close to exit the Software Properties application. If you have made any changes, the Software Preferences tool will display a dialog informing you that your channel (repository) information is out of date, as shown in Figure 20.6.

FIGURE 20.6

Change notification when exiting the Software Preferences tool

To automatically update the cache information used by Ubuntu's package management utilities, click Reload. To exit without updating the cache, click Close. If you exit without reloading the cache information, you will need to run `apt-get update` or use the Check option in Ubuntu's Update manager or the Synaptic Package Manager before you are guaranteed that your system is aware of all available packages in all valid repositories.

Problems Adding or Accessing Nonstandard Repositories

After modifying your system's `/etc/apt/sources.list` file, you may find that packages from new repositories that you've added aren't actually showing up in the list of available packages. First, make sure that you've run `apt-get update` or used Synaptic's Check button to refresh the list of available repositories and packages. This should identify any inaccessible repositories and outright syntax errors in the `sources.list file`.

Next, when running `apt-get update` to update your system's idea of repository contents, seeing messages like the following means that your system does not have the right authentication keys to add packages from a specified repository to its cache:

```
Ign http://ftp.us.debian.org sarge Release
Ign http://ftp.us.debian.org unstable Release
```

Similarly, you may see messages at the end of the output from `apt-get update`. These are related to the previous messages, but are somewhat less subtle, because they explicitly identify missing keys and associated repositories

```
W: GPG error: http://ftp.us.debian.org testing Release: \
     The following signatures couldn't be verified because the public
     key is not available: \
     NO_PUBKEY 010908312D230C5F
W: GPG error: http://ftp.us.debian.org unstable Release: \
     The following signatures couldn't be verified because the public
     key is not available: \
     NO_PUBKEY 010908312D230C5F
```

In these cases, the associated entries in `/etc/apt/sources.list` point to repositories whose keys are not present in the keyring used by the Ubuntu package management utilities. See the man page for the `apt-key` application for information on retrieving and installing keys into this keyring. For an example of the process of retrieving remote repository keys and integrating them into the keyring used by Ubuntu's package management system, see the next section.

Mixing Ubuntu and Debian Repositories

Because the Ubuntu Linux distribution has its conceptual and packaging roots in the Debian project, the Ubuntu and Debian projects both use the `/etc/apt/sources.list` file to identify sites where packages for those distributions are located, and both use the DEB package format for distributing, installing, and maintaining packages. This makes it tempting to add Debian repositories to the `/etc/apt/sources.list` file on your Ubuntu system in order to get access to even more software. The best general rule for mixing and matching Ubuntu and Debian repositories can be summed up in a very simple rule: DON'T!

That said, the rest of this section explains how to mix Ubuntu and Debian repositories for the people who are going to ignore my warnings and know what they're doing (or think that they do). Just in case, here are a few more warnings: Mixing Ubuntu and Debian repositories may break your Ubuntu system, irritate both your Debian and Ubuntu friends, create a confused system on which the source of various software packages is remarkably unclear, and cause cancer in rats. I am not responsible if your once-pretty Ubuntu system turns into a smoking crater of unbootable software. You've been warned (again).

As noted in the previous section, you can add new repositories to your `/etc/apt/sources.list` file using Synaptic or by directly modifying the underlying configuration files used by Synaptic, `aptitude`, and `apt-get`. This section explains how to manually modify these files and make other changes necessary to use selected Debian repositories.

To add Debian repositories to the list of repositories where your system looks for software packages, do the following:

1. Add the appropriate entries for the repositories that you want to use to your `/etc/apt/sources.list` file, Debian repositories use Debian release or branch names and contain different components than Ubuntu repositories, named "main," "non-free," and "contrib." When adding Debian repositories to the repositories used on your Ubuntu system, you will probably want to add the testing and unstable Debian branches. Entries for these repositories in the form required by the `/etc/apt/sources.list` file are the following:

   ```
   deb http://ftp.us.debian.org/debian testing main contrib non-free
   deb-src http://ftp.us.debian.org/debian testing main contrib non-free
   deb http://ftp.us.debian.org/debian unstable main contrib non-free
   deb-src http://ftp.us.debian.org/debian unstable main contrib non-
   free
   ```

2. When adding nonstandard repositories, you will want to explicitly set the priorities that the `apt` and Synaptic utilities use when checking repositories for new and update packages. This information is stored in the file `/etc/apt/preferences`, which usually does not exist on a standard Ubuntu system. Create this file using `sudo` and your favorite text file, and enter the relative priorities of the repositories that are present in your `/etc/apt/sources.list` file. The values in mine are the following:

```
Package: *
Pin: release a=dapper
Pin-Priority: 900
 Package: *
Pin: release o=debian a=testing
Pin-Priority: 500
 Package: *
Pin: release o=debian a=unstable
Pin-Priority: 400
```

   Priority values below 1000 have special meanings to Ubuntu's package management utilities. The sample values shown above assign anything in a Dapper repository the highest priority, assign packages in Debian's testing branch a lower priority, and assign the lowest priority to packages found in Debian's unstable branch. Different priority ranges have special meanings in the preferences file. Because my goal is not to rewrite the man page for the preferences file (see `man apt_preferences`), the values for the Debian testing and unstable branches in the previous example prevent packages from these repository components from being installed if there is a version of the same package in Dapper or the currently installed version is newer.

3. Modify the file `/etc/apt/apt.conf` to increase the size of the memory-mapped file used for storing the data structures that hold package information. Adding entire Debian repositories to the standard Ubuntu repositories requires lots of storage. The default size of this file is 4MB (4194304 bytes). I'd suggest changing this to 16MB, just to be safe, by adding the following entry to `/etc/apt/apt.conf`:

```
APT::Cache-Limit "16777216";
```

4. You'll have to retrieve and import the PGP key for the Debian repositories into the keyring used by the Ubuntu package management utilities. Ubuntu will not retrieve packages from unauthorized repositories. Adding the keys that identify nonstandard repositories as valid repositories requires several steps. First, list the keys available on the Debian public key server using the `gpg` command's `--list-keys` option. The appropriate command and its output look like the following:

```
$ gpg -v --keyserver keyring.debian.org --list-keys
gpg: using classic trust model
----------------------------
[existing key output deleted]
 pub   1024D/2D230C5F 2006-01-03 [expires: 2007-02-07]
uid      Debian Archive Automatic Signing Key (2006)
<ftpmaster@debian.org>
```

NOTE **You can also retrieve the current Debian key in ASCII format using a command like `wget http://ftp-master.debian.org/ziyi_key_year.asc` (where `year` is the current year, such as 2006), and then import it into your keyring manually, but this isn't quite as secure as getting the key directly from a key server.**

5. Retrieve the specified key by its fingerprint (2D230C5F in the previous example) into your local keyring using the `gpg` command's `--recv-keys` option. The appropriate command and its output look like the following:

```
$ gpg -v --keyserver keyring.debian.org --recv-keys 2D230C5F
gpg: requesting key 2D230C5F from hkp server keyring.debian.org
gpg: armor header: Version: GnuPG v1.4.1 (GNU/Linux)
gpg: armor header: Comment: Key ID: 0x2D230C5F
gpg: pub  1024D/2D230C5F 2006-01-03  Debian Archive Automatic Signing
Key (2006) <ftpmaster@debian.org>
gpg: using classic trust model
gpg: key 2D230C5F: public key "Debian Archive Automatic Signing Key
(2006) <ftpmaster@debian.org>" imported
gpg: 2 keys cached (7 signatures)
gpg: 1 keys processed (1 validity counts cleared)
gpg: 3 marginal(s) needed, 1 complete(s) needed, classic trust model
gpg: depth: 0  valid:   1  signed:   0  trust: 0-, 0q, 0n, 0m, 0f, 1u
gpg: Total number processed: 1
gpg:               imported: 1
```

6. Once you've retrieved the key successfully, you can then export it into the keyring used by the Ubuntu package management utilities using the `gpg`, `sudo`, and `apt-key` commands together, as in the following example:

```
$ gpg --armor --export 2D230C5F | sudo apt-key add -
OK
```

   The `--armor` option wraps the key that you are exporting in the standard ASCII descriptors for a key file, while the `--export` option uses the argument that follows this option to identify the key that you want to export.

TIP **You can use the `gpg --armor --export` command to export any key in your keyring to an ASCII file, by simply redirecting its output into a text file. You can then use these files with tools that require ASCII key files, such as Ubuntu's Software Properties tool, described earlier in this chapter in the section entitled "Enabling Additional Repository Sources Using the Software Properties Tool."**

At this point, you can now execute the command `apt-get update` to update the Ubuntu package management utilities cache of repositories and packages. You can then install any packages that any of these repositories provide. Good luck!

TIP **If you have problems after changing the Preferences file, you can use the `apt-cache policy package-name` command to see information about the priority associated with the specified package from various sources. See the online reference information for the `apt-cache` command (using the command `man apt-cache`) for more information.**

Exploring Your System Using dpkg and Friends

Package management software not only simplifies software installation, removal, and updates, but also provides great opportunities for asking questions about the software that is installed, or could be installed, on your system. Some of the command-line package management utilities that are installed on your system make it easy to find out exactly what software is installed on your system, what a specific package contains, what packages provided certain files, and so on. The next few sections describe how to answer some basic questions about existing files and available packages on your Ubuntu system.

Listing the Packages that are Installed on Your System

The `dpkg` command's `-l` option provides the easiest way of listing the packages that are installed on your system. (This option can also be specified in verbose option format as `--long`.) You can execute this command with no other argument to produce a (long) listing of every package that is installed on your system, which looks something like the following:

```
$ dpkg -l
Desired=Unknown/Install/Remove/Purge/Hold
| Status=Not/Installed/Config-files/Unpacked/Failed-config/Half-
installed
|/ Err?=(none)/Hold/Reinst-required/X=both-problems (Status,Err:
uppercase=bad)
||/ Name              Version           Description
+++-=================-==================-
=======================================
ii  3dchess           0.8.1-11.1        3D chess for X11
ii  acpi              0.09-1            displays information on ACPI
devices
ii  acpi-support      0.84              a collection of useful events
for acpi
ii  acpid             1.0.4-1ubuntu11   Utilities for using ACPI power
mgmnt
ii  adduser           3.80ubuntu2       Add and remove users and groups
[much more output deleted]
```

The first few lines above the actual package listing identify the meaning of the various fields on each line and the symbols that they contain. This information is provided in a fairly cryptic fashion that is mandated by the terminal-oriented output produced by the `dpkg` utility. The entries for package name, package version, and package description are pretty straightforward, but the first field requires some serious explanation. The characters in the first field have the following meaning and possible values:

- The first character in the first field indicates the desired status of the package, which is the state that the packaging system thinks the package should be in. Available indicators in this position are `h` (the package is marked as being on hold, and cannot be updated or removed), `i` (the package should be installed), `p` (the package and all associated configuration information is supposed to be purged), `r` (the package is supposed to be removed, but associated configuration files will be preserved), and `u` (the package has never been installed on this system, so its state is unknown).
- The second character in the first field indicates the actual status of the package on your system. Available indicators in this position are `c` (the configuration files for the package are installed, but the package is not), `f` (the script used to complete the configuration of this package, known as a post-installation script, failed for some reason, and the package is therefore not guaranteed to be correctly installed), `h` (the package is partially installed because the installation process was interrupted), `i` (the package is correctly installed), `n` (the package is not installed), and `u` (the package was retrieved, unpacked, and is partially installed, but its post-installation script was not executed).
- The third character highlights any errors that are associated with the package. Available indicators in this position are a space (no errors, which looks remarkably like there is nothing in this position), `H` (the package has been marked as being on hold by a the package management system itself, which usually means that other packages that this package requires are not installed), `R` (reinstallation is required), and `X` (the package both requires re-installation and has been automatically put on hold by the package management system itself).

Now that I've explained what each character position in the first field of a package entry means, it's easy to see what the ASCII art in the `dpkg` output heading means. This is not exactly the same thing as being "intuitive," but that's hard to do in the limited number of characters that you can display and easily use from the command line.

Listing the Packages that are Available for Your System

The previous section explained how to figure out what packages are installed on your system. While that's interesting, a more interesting question perhaps is, "What packages are available for my system that I have not yet installed?" As mentioned in the previous section, the `dpkg` command's `-l` option with no other arguments shows the list of installed packages. You can supply an argument to this command to list available packages that match that argument, whether installed or un-installed. For example, the `dpkg -l emacs` command should list all of the packages that have the string `emacs` in their package name. Let's try that:

```
$ dpkg -l emacs
No packages found matching emacs.
```

Well, that seems odd, because I'm actually typing this in emacs on my Ubuntu system. The problem is that any argument that you supply to `dpkg` is used as a pattern match, and there are apparently no packages installed on my system whose exact name is "`emacs`." Let's try that again, using a standard Linux wildcard to say that you want to list any packages with a name that begins with emacs, as in the following example:

```
$ dpkg -l 'emacs*'
Desired=Unknown/Install/Remove/Purge/Hold
| Status=Not/Installed/Config-files/Unpacked/Failed-config/Half-
installed
|/ Err?=(none)/Hold/Reinst-required/X=both-problems (Status,Err:
uppercase=bad)
||/ Name                    Version        Description
+++-=======================-==============-
==================================
ii  emacs-chess             2.0b5-1        a client/library for playing
Chess
ii  emacs-chess-pieces      2.0b5-1        XPM images of chess pieces
ii  emacs-goodies-el        26.4-1         Miscellaneous add-ons for
Emacs
un  emacs-goodies-extra-el <none>         (no description available)
ii  emacs21                 21.4a-3ubuntu2 The GNU Emacs editor
ii  emacs21-bin-common      21.4a-3ubuntu2 The GNU Emacs editor's
shared...
ii  emacs21-common          21.4a-3ubuntu2 The GNU Emacs editor's
shared...
un  emacs21-el              <none>         (no description available)
un  emacs21-nox             <none>         (no description available)
un  emacsen                 <none>         (no description available)
ii  emacsen-common          1.4.17         Common facilities for all
emacsen
```

That's more like it! Note that when using wildcards in a `dpkg` specification, you have to protect them from expansion on the command line by enclosing your `dpkg` wildcard specification within quotation marks of some sort.

The previous output showed some emacs packages, but I could swear that there were others on my Ubuntu system. Let's try a more general wildcard that searches for packages that contain the string "emacs" anywhere in their names:

```
$ dpkg -l '*emacs*'
Desired=Unknown/Install/Remove/Purge/Hold
| Status=Not/Installed/Config-files/Unpacked/Failed-config/Half-
installed
|/ Err?=(none)/Hold/Reinst-required/X=both-problems (Status,Err:
uppercase=bad)
||/ Name                          Version         Description
+++-=========================-===============-==============================
ii  emacs-chess               2.0b5-1          a client/library for
Chess ii  emacs-chess-pieces            2.0b5-1          XPM images of
chess pieces
ii  emacs-goodies-el          26.4-1           Miscellaneous add-ons
un  emacs-goodies-extra-el    <none>           (no description
available)
ii  emacs21                   21.4a-3ubuntu2   The GNU Emacs editor
ii  emacs21-bin-common        21.4a-3ubuntu2   The GNU Emacs shared...
ii  emacs21-common            21.4a-3ubuntu2   The GNU Emacs shared...
un  emacs21-el                <none>           (no description
available)
un  emacs21-nox               <none>           (no description
available)
un  emacsen                   <none>           (no description
available)
ii  emacsen-common            1.4.17           Common facilities for
emacsen
un  xemacs                    <none>           (no description
available)
un  xemacs-support            <none>           (no description
available)
un  xemacs-widget             <none>           (no description
available)
ii  xemacs21                  21.4.18-1ubuntu1 highly customizable text
editor
[additional output deleted]
```

Even better! Thanks to the explanation of the characters in the first field of your dpkg output (provided in the previous section of this chapter), it's clear that I need to install `xemacs` on this system.

Because this is a Linux system, you can pipe your `dpkg` output through other commands to answer questions like, "What packages are not yet installed on my system whose names contain the string emacs anywhere in their names," which I can do using `grep` to look for lines that begin with the string "un," as in the following example:

```
$ dpkg -l '*emacs*' | grep '^un'
un  emacs-goodies-extra-el    <none>          (no description
available)
```

```
un  emacs21-el               <none>          (no description
available)
un  emacs21-nox              <none>          (no description
available)
un  emacsen                  <none>          (no description
available)
un  xemacs                   <none>          (no description
available)
un  xemacs-support           <none>          (no description
available)
un  xemacs-widget            <none>          (no description
available)
un  xemacs21-gnome-mule      <none>          (no description
available)
un  xemacs21-gnome-mule...   <none>          (no description
available)
un  xemacs21-gnome-nomule    <none>          (no description
available)
un  xemacs21-mule-canna-wnn  <none>          (no description
available)
un  xemacs21-nomule          <none>          (no description
available)
un  xemacs21-supportel       <none>          (no description
available)
```

Other Ubuntu package management utilities provide similar search features, particularly the aptitude utility. Though most commonly associated with it's terminal-oriented, quasi-graphical interface (discussed later in this chapter in the section entitled "Using aptitude to Add and Remove Software," aptitude also provide a command-line mode with many powerful commands. To locate packages, aptitude provides a search keyword that enables you to specify a substring to search for (no wildcards necessary), and displays information about any installed or available package that is present in any repository listed in your `/etc/apt/sources.list` file. Some sample output from a search for emacs using aptitude is the following:

```
$ aptitude search emacs
p   acl2-emacs             - A Computational Logic for Applicative Comm
p   aleph-emacs            - The Aleph programming language - emacs mod
p   cxref-emacs            - Generates latex and HTML documentation for
p   emacs                  - The GNU Emacs editor
i   emacs-chess            - a client and library for playing Chess fro
i   emacs-chess-pieces     - XPM images of chess pieces for emacs-chess
p   emacs-color-themes     - Color themes for Emacs
p   emacs-extra            - emacs configuration
i   emacs-goodies-el       - Miscellaneous add-ons for Emacs
i   emacs21                - The GNU Emacs editor
i   emacs21-bin-common     - The GNU Emacs editor's shared, architectur
i   emacs21-common         - The GNU Emacs editor's shared, architectur
[additional output deleted]
```

You'll notice that the aptitude utility finds more packages than the `dpkg -l search` command that was described earlier.

Another very convenient application for searching for packages is the `apt-cache` utility, which is a command-line utility that provides subcommands that enable you to search and manipulate the cache of packages that are installed or available for installation on your system and the metadata that is associated with those packages. You must enable the universe repository in order to install the `apt-cache` utility. The `apt-cache` utility can also be used to search the packages in any repository that is active in your `/etc/apt/sources.list` file, and find even more matching packages because it searches within the package name and both the short and long descriptions of all available packages, whether installed or not. Some sample output from an `apt-cache` search for our favorite string, emacs, is the following:

```
$ apt-cache search emacs
acl2-emacs - A Computational Logic for Applicative Common Lisp: emacs
interface
ada-mode - Ada mode for GNU Emacs and XEmacs
af - An Emacs-like mail reader and composer
aleph-emacs - The Aleph programming language - emacs mode
anjsp - A major mode to edit JSP and PSP code with Emacs
anthy-el - A Japanese input method (elisp fronted)
apel - portable library for emacsen
aplus-fsf-el - XEmacs lisp for A+ development
artist - Emacs Lisp drawing package
asn1-mode - Emacs mode for editing ASN.1 specification files
bhl - Emacs mode for converting brut text to HTML and LaTeX
bigloo-ude - Bigloo Unified Development Environment for Emacs
biomode - [Biology] An Emacs mode to edit genetic data
[additional output deleted]
```

As you can see, the `apt-cache` and `aptitude` utilities are easiest to use when searching for packages, and return the most verbose results.

Listing Information About a Package

Once you've found a package that you're interested in, there are an equivalent number of ways to get more detailed information about the package and its contents. However, because the `apt-cache` and `aptitude` utilities return the best search results, it makes sense to use them to display information about any packages that they've identified. For example, to find out detailed information about the `xemacs21` package that was listed in the previous section, you could use either the `apt-cache show xemacs21` or `aptitude show xemacs21` commands, both of which return almost identical information. The following example shows the output from the `aptitude show xemacs21` command, which is slightly more verbose:

```
$ aptitude show xemacs21
Package: xemacs21
New: yes
State: installed
Automatically installed: no
Version: 21.4.18-1ubuntu1
Priority: optional
Section: universe/editors
Maintainer: OHURA Makoto <ohura@debian.org>
Uncompressed Size: 49.2k
Depends: xemacs21-mule (= 21.4.18-1ubuntu1) | xemacs21-mule-canna-wnn
         (= 21.4.18-1ubuntu1) | xemacs21-nomule (= 21.4.18-1ubuntu1) |
```

```
          xemacs21-gnome-mule (= 21.4.18-1ubuntu1) |
          xemacs21-gnome-mule-canna-wnn (= 21.4.18-1ubuntu1) |
          xemacs21-gnome-nomule (= 21.4.18-1ubuntu1)
Conflicts: xemacs, xemacs-widget
Replaces: xemacs, xemacs-widget
Provided by: xemacs21-nomule, xemacs21-mule-canna-wnn, xemacs21-mule,
             xemacs21-gnome-nomule, xemacs21-gnome-mule-canna-wnn,
             xemacs21-gnome-mule
Description: highly customizable text editor
 XEmacs is a full fledged programming language with a mail reader, news
 reader, info browser, Web browser, calendar, specialized editor for
more
 programming languages and other formats than most people encounter in
a
 lifetime, and much much more.
  This package exists to cause the installation of the real XEmacs
packages.
```

This output probably provides more than you want to know about the specified package, but it's always better to err on the side of caution.

Listing the Contents of a Package

In some cases, you may want to list all of the files associated with a package that is installed on your system. To do this, you can use the `dpkg` command's `-L` option, followed by the name of the package whose contents you want to list. Sample output that lists the contents of the `emacs21` package on my Ubuntu system is the following:

```
$ dpkg -L emacs21
/.
/usr
/usr/bin
/usr/bin/emacs21-x
/usr/share
/usr/share/emacs
/usr/share/emacs/21.4
/usr/share/emacs/21.4/etc
/usr/share/emacs/21.4/etc/DOC-21.4.1
/usr/share/applications
/usr/share/applications/emacs21.desktop
/usr/share/doc
/usr/share/doc/emacs21
/usr/share/doc/emacs21/README.Debian.gz
/usr/share/doc/emacs21/copyright
/usr/share/doc/emacs21/changelog.Debian.gz
/usr/lib
/usr/lib/emacs
/usr/lib/emacs/21.4
/usr/lib/emacs/21.4/i486-linux-gnu
/usr/lib/emacs/21.4/i486-linux-gnu/fns-21.4.1-x.el
/usr/lib/menu
```

Being able to identify the packages provided by a package that is installed on your system can be handy if you are considering removing a package but want to make sure that doing so does not delete a file that you want to preserve. The next section describes how to find out what package does provide a specific file.

TIP **You can list the files that are provided by packages that are not yet installed by using the `apt-file` utility with a command such as `apt-file list package`, where `package` is the name of the package that you are interested in. The `apt-file` utility is described more detail in "Determining What Package Provides a Missing File" later in this chapter.**

Determining What Package Provides an Existing File

After working with Ubuntu for a while and installing, updating, and removing some number of packages, you may be curious about which package provides a certain file on your system. This is easy enough to determine using the `dpkg` command's `--search` option, as in the following example, where I'm curious about which package provides the `/usr/bin/ar` archiving utility:

```
$ dpkg --search /usr/bin/ar
binutils: /usr/bin/ar
```

If you know that a utility that you're looking for is in your path but you don't want to have to determine the directory in which it is physically located, you can use the cool shell trick of using the output of an existing command as the input to another one, in this case combining the which command, to determine the full pathname of an application and the `dpkg --search` command, which produces exactly the same output:

```
$ dpkg --search `which ar`
binutils: /usr/bin/ar
```

When searching for files, the `dpkg --search` command can't find symbolic links that are created by package post-installation scripts. If you are trying to find a file that you know exists but cannot locate it, make sure that the file is not a symbolic link and, if it is, search for the file that it actually points to. This can sometimes require a few tries, as in the following example:

```
$ dpkg --search /usr/bin/emacs
dpkg: /usr/bin/emacs not found.

 $ ls -l /usr/bin/emacs
 /usr/bin/emacs -> /etc/alternatives/emacs

 $ dpkg --search /etc/alternatives/emacs
dpkg: /etc/alternatives/emacs not found.

 $  ls -l /etc/alternatives/emacs
/etc/alternatives/emacs -> /usr/bin/emacs21-x

 $ dpkg --search /usr/bin/emacs21-x
emacs21: /usr/bin/emacs21-x
```

Though sometimes tedious, symbolic links are usually not an actual part of a package and you must therefore ferret out a file that actually is part of a package.

TIP **Another handy package management utility that you may want to use when determining what package provides a specific file is the `dlocate` utility. This utility is not installed on Ubuntu systems by default, but can easily be installed using `apt-get`, `aptitude`, or Synaptic Package Manager.**

After installing this command, you can search for the package that provides a file using the `dlocate -S` command, as in the following example:

```
$ dlocate -S /usr/bin/emacs
emacs21-bin-common: /usr/bin/emacsclient.emacs21
emacs21: /usr/bin/emacs21-x
emacs21: /usr/bin/emacs21
```

The `dlocate` utility's `-S` option uses the string that you specify as a substring to search for, and can therefore be a bit more helpful when looking for files on your system.

Determining What Package Provides a Missing File

Identifying the package that provides a file that is not installed on your system is somewhat tricky, but is a common question when you are trying to build software whose source code requires an include file that is not present on your system, or you are trying to link software that requires a missing library. Unfortunately, the utility that performs this type of search, the `apt-file` utility, is not installed on Ubuntu systems by default, but can easily be installed using `apt-get`, `aptitude`, or Synaptic Package Manager. After installing this command, you must first update its idea of the available packages in the repositories that you are using by running the command `sudo apt-file update`.

Once this command completes, you can search for the package that provides a file, even if that file is not installed on your system, using the `apt-file search` command, as in the following example, which looks for the package associated with the *libpowersave* library:

```
$ ls -l /usr/lib/*power*
ls: /usr/lib/*power*: No such file or directory
 $ apt-file search libpowersave.so
libpowersave-dev: usr/lib/libpowersave.so
libpowersave10: usr/lib/libpowersave.so.10
libpowersave10: usr/lib/libpowersave.so.10.0.3
```

Installing the `apt-file` utility can save you a tremendous amount of hair-pulling and general frustration when you are trying to build software and have no idea what package provides the missing include file or "the missing link." (Sorry, but I couldn't resist.)

TIP **You can also use the `apt-file` utility to list the contents of packages that are not yet installed by using the apt-file list package command, where package is the name of the package that you are interested in.**

Using apt-get to Add and Remove Software

The `apt-get` command is the fundamental user-level command provided as part of the Ubuntu package management suite. The `apt-get` command is a command-line tool that is fast and easy to use.

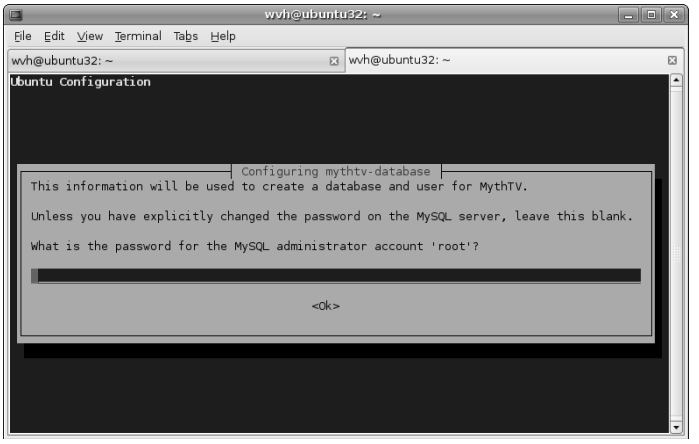
NOTE **As with any system administrative utility on your Ubuntu system, running the `apt-get` command requires the use of the `sudo` command or an equivalent, such as running `apt-get` under a shell that itself has been executed using the `sudo` command. To simplify examples and the readability of the text, the examples and discussion of using the `apt-get` command throughout this section do not include the `sudo` command.**

Much of the basic functionality provided by the `apt-get` command is quite straightforward. For example, you can install a package and any other packages that it requires using the `apt-get install package` command. Similarly, you can remove a package by using the `apt-get remove package` command, specifying the `--purge` option if you also want to remove any configuration files or other data associated with the package that you're removing. Before using `apt-get` to install new software or new versions of existing software, you should always first execute the `apt-get update` command to ensure that `apt-get` is aware of the latest software packages and versions of that software from all of the repositories listed in your `/etc/apt/sources.list` file.

When using the `apt-get` utility to install software, you may occasionally have to provide additional information during the installation process. In these cases, the `apt-get` utility displays a quasi-graphical screen in the terminal application from which you were running the `apt-get` utility, an example of which is shown in Figure 20.7.

FIGURE 20.7

Supplying configuration information during package installation



Whenever you see this sort of screen, you must answer any questions that it asks and supply any required information before the installation of your package can complete.

TIP **If you are using `apt-get` to install new software and the `apt-get` utility informs you that other packages are recommended or suggested for installation, you may want to consider using `aptitude`'s command-line interface to install these packages rather than `apt-get`. As explained later in this chapter in the section entitled "Using aptitude to Install Recommended Software" `aptitude` provides options that can automate installing recommended software when installing a new software package.**

As mentioned throughout this book, I prefer to use the graphical Synaptic tool to search for, install, and remove packages whenever possible. The `apt-get` tool provides these same capabilities, but also provides some powerful capabilities that are not duplicated in the Synaptic tool, or are at least much more easily done using `apt-get` from the command line. Even if you're a Synaptic fan, the next few sections explain how and when you still may want to use `apt-get` to perform these advanced functions.

Upgrading Your System Using apt-get

The `apt-get` upgrade command searches all of the repositories in your `/etc/apt/sources.list` file for new versions of packages that are currently installed on your system, and downloads and installs those new versions. Before using this command, you should always first execute the `apt-get update` command to ensure that `apt-get` is aware of the latest software packages and versions of that software from all of the repositories listed in your `/etc/apt/sources.list` file.

During the upgrade process, the `apt-get upgrade` command will not change the installation status of any other packages on your system. This is the installation status of a package, not the version of a package. If a new version of a package requires a newer version of another package that is already installed on your system, the `apt-get` upgrade command will also install the updated version of that other package. However, if a newer version of an existing package requires that new packages be installed or that existing packages be removed, the `apt-get upgrade` command will not install the newer version of the existing package. To do this, you will need to use the `apt-get dist-upgrade` command, as described in the next section.

Smart System Upgrades Using apt-get

The `apt-get dist-upgrade` command searches all of the repositories in your `/etc/apt/sources.list` file for new versions of packages that are currently installed on your system, and downloads and installs those new versions. Before running this command, you should always first execute the `apt-get` update command to ensure that `apt-get` is aware of the latest software packages and versions of that software from all of the repositories listed in your `/etc/apt/sources.list` file.

The difference between the `apt-get upgrade` and `apt-get dist-upgrade` commands is that the `apt-get dist-upgrade` command will do what is known as a "smart upgrade." This means that it will do its best to handle any new package requirements or the removal of any existing packages that mandated in order to install the latest versions of all of the software on your system. For example, if a new version of a package requires that a new package be installed on your system, the `apt-get` dist-upgrade command will install both of these, while the `apt-get upgrade` command would not have updated the original software package. Similarly, if installing a new version of an existing package requires the removal of any other package(s) that are currently installed on your system, the `apt-get dist-upgrade` command will remove those packages in order to install the new version of the existing package.

Even if you are a Synaptic fan, you will occasionally need to execute the `apt-get dist-upgrade` command in order to completely upgrade all of the packages on your system. Synaptic provides a "smart upgrade" preference that tries to do this sort of thing for you whenever possible, but the `apt-get dist-upgrade` command is still occasionally required. When using Synaptic, symptoms of the need to use the `apt-get dist-upgrade` command are packages that are mysteriously identified as being held back and not updated — and, of course, the occasional pop-up message that explicitly tells you to run the `apt-get dist-upgrade` command.

Retrieving Package Source Code Using apt-get

Even in the wonderful world of constantly updated repositories with a rich collection of software, you may occasionally want to build your own versions of software packages for your system. The most common situation in which you'll want to do this is when you find patches to an existing software package that fix problems you're experiencing or add enhancements, and you just can't wait for them to appear in the official or backports repositories. You may also have your own ideas about changes that should be made to a package and want to test and work through your ideas. Linux and its ecosystem of utilities and other applications are open source, after all!

The `apt-get` command makes it remarkably easy to retrieve the source code for an installed package. The only requirement for this capability is that your `/etc/apt/sources.list` file includes a `deb-src` entry for the repository from which you retrieved the binary version of that software. For example, if you want to retrieve the source for a package in which the binary version is located in the multiverse repository component, you must have entries like the following (or equivalent entries) in your `/etc/apt/sources.list` file:

```
deb http://archive.ubuntu.com/ubuntu/ dapper multiverse
deb-src http://archive.ubuntu.com/ubuntu/ dapper multiverse
```

Assuming that the right entries are present in your `/etc/apt/sources.list file`, retrieving the source code for a specified package is easy using the `apt-get source` command. The following example shows the retrieval of the source code for one of my favorite packages, the MythTV personal video recorder package:

```
$ apt-get source mythtv
Reading package lists... Done
Building dependency tree... Done
Need to get 9836kB of source archives.
Get: 1 http://.../multiverse mythtv 0.18.1-5ubuntu3 (dsc) [1084B]
Get: 2 http://... dapper/multiverse mythtv 0.18.1-5ubuntu3 (tar)
[9817kB]
Get: 3 http://... dapper/multiverse mythtv 0.18.1-5ubuntu3 (diff)
[17.3kB]
Fetched 9836kB in 1m31s (107kB/s)
dpkg-source: extracting mythtv in mythtv-0.18.1
dpkg-source: unpacking mythtv_0.18.1.orig.tar.gz
dpkg-source: applying ./mythtv_0.18.1-5ubuntu3.diff.gz
```

When retrieving the source code for a package, the `apt-get source` utility actually retrieves multiple files. The extensions of these files are the following:

- `diff.gz`: a compressed file containing any available patches to the standard source archive for the package
- `DSC`: a description file for the Ubuntu package that identifies the contents of the package, associated binary packages, any package dependencies for that package, and so on
- `tar.gz`: a compressed archive file containing the source code for the official version of this package

After retrieving these files, the `apt-get source` command unpacks the compressed source archive, creating a working directory for your development efforts, and automatically applies any patches that are available in the (optional) compressed patch file.

Chapter 18 discussed how to install the basic packages required for software development on an Ubuntu system. If you are writing your own code or simply compiling existing applications on an Ubuntu system, that may be sufficient — except when you are building complex graphical software or when you are building your own DEB packages from the downloaded source for an Ubuntu package. In these cases, you will quickly encounter frustration such as the fact that today's graphical software has dependencies on libraries, include files, and utilities that you may not be familiar with or even have heard of before. Similarly, building DEB packages requires the use of billions and billions of utilities that may be new to you. Not to worry — you're not the first person to have encountered these issues, and the `apt-get` command provides a handy solution, as explained in the next section.

Satisfying Build Dependencies Using apt-get

Many of the software packages available for Linux have dependencies on other packages, which makes perfect sense — why reinvent the wheel when you can just link your code with it? However, if you decide to work on an existing Linux package, build-time dependencies can be frustrating. They're rarely well-documented, so you tend to encounter them as an iterative set of errors when you try to compile and link the application you're trying to build. This is also true when you begin to work with software packages in the formats used by different package management systems. The DEB packages used on Ubuntu and Debian systems (and derivatives) eliminate the dependency mumbo-jumbo that you often encounter with other package formats (such as RPM), but require the presence of an entire ecosystem of related utilities in order to build and package them successfully. Getting started with these can be tricky — you don't know what you're missing until your build process goes up in flames.

The `apt-get` tool provides a great, automated solution for these sorts of problems through its `build-dep` command-line option. Using this command-line option, followed by the name of the package that you want to build automatically identifies, retrieves, and installs all of the include files, libraries, and tools required to build the specified package. An example of the output from this command when retrieving build dependencies for the MythTV package used as an example in the previous section is the following:

```
$ sudo apt-get build-dep mythtv
Reading package lists... Done
Building dependency tree... Done
The following NEW packages will be installed
  build-essential g++ g++-3.4 g++-4.0 libartsc0-dev libasound2-dev
  libaudio-dev libdvb-dev libgl1-mesa-dev libglu1-mesa-dev
  libjpeg62-dev liblame-dev liblcms1-dev liblircclient-dev
  libmng-dev libmysqlclient14-dev libogg-dev libqt3-headers
  libqt3-mt-dev libstdc++6-4.0-dev libstdc++6-dev libvorbis-dev
  libxmu-dev libxmu-headers libxt-dev libxv-dev libxvmc-dev
  libxxf86vm-dev mesa-common-dev qt3-dev-tools x11proto-video-dev
  x11proto-xf86vidmode-dev
0 upgraded, 32 newly installed, 0 to remove and 0 not upgraded.
Need to get 14.2MB/17.9MB of archives.
After unpacking 63.4MB of additional disk space will be used.
Do you want to continue [Y/n]? Y
[much output deleted]
```

As you can see from this example, it would have taken me many, many iterative compilation attempts to retrieve all of the packages required to build MythTV. Identifying the packages that provide missing include files and libraries can also be time-consuming, even with the help of the `apt-file` application discussed earlier in this chapter in the section entitled "Determining What Package Provides a Missing File." The `apt-get build-dep` command eliminates these sorts of hassles, letting you focus on the software, not the infrastructure required for building it.

Using aptitude to Add and Remove Software

As mentioned earlier in this chapter, the `dpkg` utility provides the conceptual underpinnings of much of the package management software provided on Ubuntu (and Debian) systems. However powerful it may be, it is a command-line utility, and therefore doesn't provide any sort of graphical interface, so the `dselect` utility was developed in order to at least provide a quasi-graphical interface using terminal-oriented cursor-movement

libraries such as `curses` (now `ncurses`, for new curses). However, `dselect` couldn't quite let go of its command-line roots, and therefore also provides a command-line interface.

The `apt-get` application discussed earlier in this chapter provides a simpler, higher-level interface than that provided by the `dpkg` utility. However, `apt-get` is YACLU (Yet Another Command-Line utility), and therefore doesn't provide any sort of graphical interface either. The `aptitude` application was developed to solve this problem, but couldn't quite let go of the command line either. Thus `aptitude` (like `deselect` before it) provides both a powerful command-line interface and a terminal-oriented, quasi-graphical interface.

Personally, I find terminal-oriented graphical interfaces quaint, at best. I prefer not to use them unless absolutely necessary, especially when they've been replaced by actual graphical user interfaces such as that provided by the Synaptic Package Manager.

NOTE **When using the `aptitude` utility to install software, you may occasionally have to provide additional, configuration-related information during the installation process. In these cases, the `aptitude` utility displays the same sort of information requests as those displayed by the `apt-get` utility and shown earlier in Figure 20.7. Whenever you see this sort of screen, you must answer any questions that it asks and supply any requested information before the installation of your package can complete.**

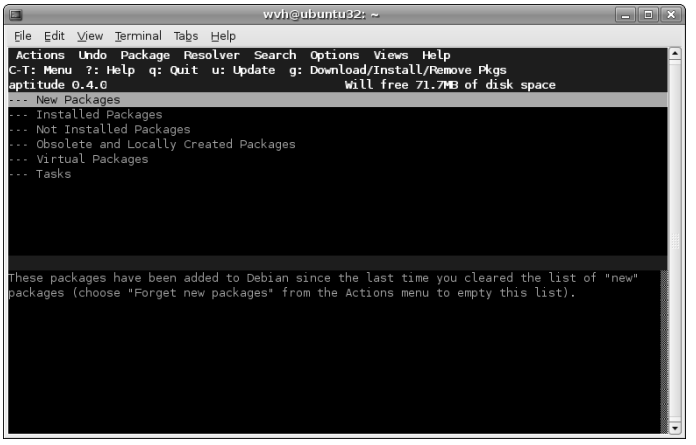
Software can easily be installed by `aptitude` using the command-line `aptitude install` command, and removed using the `aptitude remove` command. As discussed later in the section entitled "Using aptitude to Install Recommended Software," `aptitude` provides some convenient options that make it attractive to use `aptitude` from the command line in certain cases. Many people are also fans of the aptitude interface, primarily those who understand it. The next section discusses the absolute basics of the `aptitude` user interface, focusing on how to make it readable by mere mortals.

Tips and Tricks for Using the aptitude User Interface

Figure 20.8 shows the default `aptitude` interface that you see the first time that you start `aptitude` with no arguments in a GNOME Terminal.

FIGURE 20.8

The default aptitude interface in the GNOME Terminal



This interface looks essentially the same in an X Window system xterm or any other terminal application that supports color. (It's slightly more psychedelic in an xterm, but I won't bore you with a screen shot.)

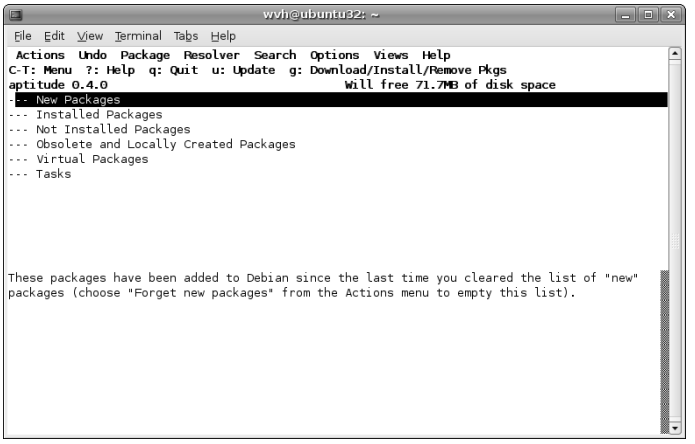
If you insist on using this interface but are having problems seeing the text or dealing with the color scheme, you're somewhat out of luck, because the `aptitude` application doesn't offer command-line options for changing the color scheme. However, all is not lost — you can make the interface easier to read by changing your terminal application's underlying notion of the type of terminal in which you are running `aptitude`. A good choice for this is the value `vt100`, which is a classic terminal from the late Digital Equipment Corporation that only supported black and white. To make this change, exit `aptitude` by pressing Q and using the tab key to emphasize that you do actually want to exit from `aptitude`. Next, type the following command in your terminal application:

```
$ export TERM=vt100
```

After executing this command, you can then restart `aptitude`, which will look like the screen shown in Figure 20.9.

FIGURE 20.9

The aptitude interface in black and white



Changing the color scheme doesn't make the `aptitude` interface any more attractive, but (for me) it does at least make it more usable.

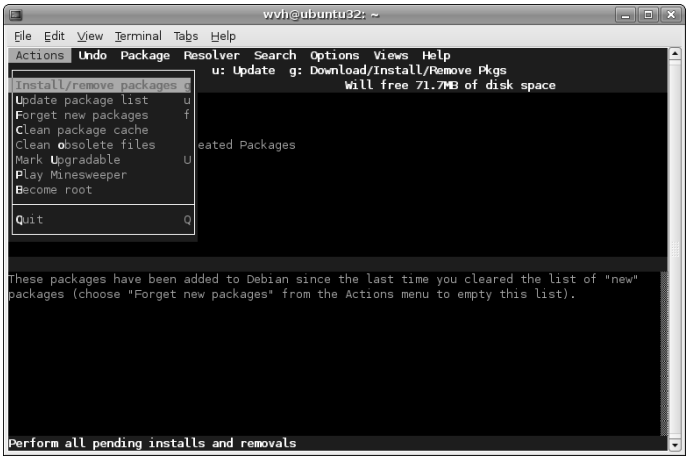
Once running `aptitude`, my favorite commands are the following:

- `Q`: Enables you to exit `aptitude`.
- `?`: Displays a panel containing summary information about using the `aptitude` interface.
- `Control-t`: Holding down the control key and pressing T displays `aptitude`'s menus, which you can navigate through using the arrow keys or by pressing the highlighted key shortcut to execute a specific command.

Figure 20.10 displays a menu in aptitude using `aptitude`'s default color scheme. Need I say more? This is the sort of thing that gives Linux applications a bad name.

FIGURE 20.10

A menu in the default aptitude interface



For more detailed (and less opinionated) information about using the `aptitude` user interface, see the online reference information for `aptitude` (`man aptitude`) or search the Web for relevant information.

Using aptitude to Install Recommended Software

If you have been using `apt-get` to install new software and the `apt-get` utility informs you that other packages are recommended or suggested for installation, you may want to consider using `aptitude`'s command-line interface rather than `apt-get`. The DEB packages used by Ubuntu identify various dependences, divided into five different classes:

- **Conflicts:** packages (or package versions) with which a package conflicts, and which cannot therefore be installed on a system when a specified package is being installed
- **Depends:** packages that are mandatory for the correct operation of a package, and which therefore must also be installed when a given package is installed
- **Enhances:** packages whose operation is improved or simplified as a result of installing a package, but between which there is no direct relationship
- **Pre-Depends:** packages that must be completely installed on your system before you can install a given package
- **Recommends:** packages that you should probably have, but which are not absolutely required for the correct operation of a given package
- **Suggests:** packages that may be useful, and which are normally installed on systems where a given package is installed

Requirements marked as Pre-Depends (which are rarely used or necessary) must be satisfied before you can install a given package. Requirements marked as Depends are automatically installed along with a given package. The `aptitude` utility enables you to automatically install packages that are recommended for installation by including the `--with-recommends` command-line option on your `aptitude` install command line.

The `aptitude` utility is the only command-line package installation utility that enables you to automatically install recommended packages along with required packages. Specifying the `--with-recommends` command-line option when installing packages from the command line can be quite convenient and can save you search time in the future if you discover that you want to use a recommended packages. A similar option to aptitude's `--with-recommends` option for `apt-get`, `--recommends`, and a related on for installing suggested packages, `--suggests`, have been "suggested" for `apt-get` for a long time, but are not available in any version of `apt-get` that I've ever seen. As I'll discuss in the section of this chapter on Synaptic, the Synaptic tool provides configuration options for considering recommended packages to be dependencies and automatically installing them; similarly, its interface provides a run-time option for manually selecting suggested packages and including them as part of the package installation process.

Advantages of Using aptitude to Install and Remove Software

Though I am not such a fan of the `aptitude` user interface, `aptitude` itself does provide some significantly useful capabilities. Its search capabilities, discussed earlier in this chapter in the section entitled "Listing the Packages that are Available for Your System," are easy to use and quite powerful — exactly what you want in a utility. Similarly, the current version of `aptitude` has some other advantages when installing and, specifically, when removing packages. The most significant of these are the fact that `aptitude` remembers dependency information when installing packages. Every Ubuntu package management utility understand requirements when installing packages, but `aptitude` remember this information and can therefore use that information should you decide to remove packages.

As an example, the following sample output shows an attempt to remove the PostgreSQL database system from one of my Ubuntu systems using `apt-get`:

```
$ sudo apt-get remove postgresql
Password:
Reading package lists... Done
Building dependency tree... Done
The following packages will be REMOVED
  postgresql
0 upgraded, 0 newly installed, 1 to remove and 0 not upgraded.
Need to get 0B of archives.
After unpacking 45.1kB disk space will be freed.
Do you want to continue [Y/n]? n
Abort.
```

Using `aptitude` to remove the same package is much more useful, because it also offers to remove packages that were installed along with the `postgresql` package, but which are not longer needed on that system if the `postgresql` package itself is removed, as in the following example:

```
$ sudo aptitude remove postgresql
Reading package lists... Done
Building dependency tree... Done
Reading extended state information
Initializing package states... Done
Building tag database... Done
The following packages are unused and will be REMOVED:
  postgresql-plperl-7.4 postgresql-plpython-7.4 postgresql-pltcl-7.4
```

```
The following packages will be REMOVED:
  postgresql
0 packages upgraded, 0 newly installed, 4 to remove and 0 not upgraded.
Need to get 0B of archives. After unpacking 651kB will be freed.
Do you want to continue? [Y/n/?] Y
[much output deleted]
```

For me, this makes `aptitude` a much more attractive command-line package installation solution than `apt-get`. Note that `aptitude` only remembers dependency information for packages that it has installed, so I tend to use `aptitude` from the command-line for both package installation and removal. Otherwise, if you actively install and remove large numbers of packages, your system tends to accumulate packages that you no longer need, but which you're not aware of. For information about ways of checking for unneeded packages and removing them, see the last section of this chapter, "Keeping your System Lean, Mean, and Pristine."

Using Synaptic to Add and Remove Software

The Synaptic Package Manager is a graphical, GNOME application for installing, removing, and generally managing Ubuntu software packages. In the unlikely event that this is only the second sentence you have read in this book, you may not know that I recommend using the Synaptic package management utility for almost all package management tasks on Ubuntu systems. It's true. Synaptic is the type of administrative application that users of other Linux distributions and other personal computer operating systems wish they had.

Synaptic was originally developed by the folks at Connectiva Linux, a Latin American Linux distribution that was eventually acquired by Mandrake Linux, which then changed its name to Mandriva to highlight that it wasn't your father's Mandrake Linux distribution anymore. Synaptic was originally written using the WINGS toolkit used by the Window Maker X Window system window manager (`www.windowmaker.info`), which attempted to provide a look and feel familiar to users of NeXT computer systems. Today, Synaptic is now built with the standard GTK+ toolkit used by GNOME. Interestingly, Connectiva was an RPM-based distribution, but Synaptic is now more commonly associated with the DEB packages used by distributions such as Ubuntu and Debian.

The Synaptic package management utility is installed by default as part of any Ubuntu installation. Select the System ➪ Administration ➪ Synaptic Package Manager menu item to start the Synaptic tool. As when trying to run any graphical administrative application on Ubuntu, you will be prompted for your administrative password before Synaptic actually starts. Figure 20.11 shows the dialog displayed when you first start Synaptic.

The left pane of the Synaptic dialog shown in Figure 20.11 shows the conceptual categories into which software is organized based on the default repositories enabled when you first install Ubuntu. Figure 20.12 shows an initial Synaptic dialog when additional repositories have been enabled.

FIGURE 20.11

The startup dialog for the Synaptic utility

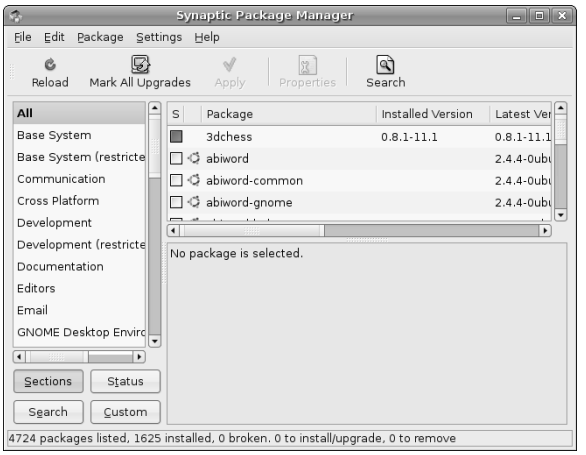
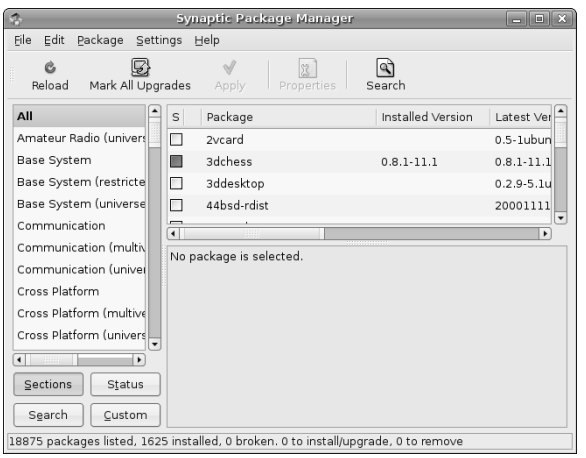


FIGURE 20.12

The startup dialog for the Synaptic utility



Note that the software listed in the left pane of the main Synaptic dialog is still organized into the same basic categories, but that entries for those categories from each available repository component are also listed. This enables you to identify the software packages that are associated with the various licensing requirements explained in "Ubuntu Repositories and Components" earlier in this chapter.

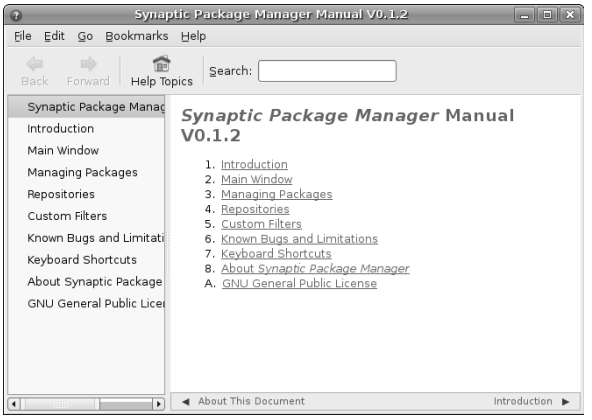
The basic package-related areas of Synaptic's main dialog are the following:

- **left pane:** Lists available software organized into various conceptual categories based on the repositories listed in `/etc/apt/sources.list`.
- **upper-right pane:** Lists the software packages that are available in the category that is currently selected in the left pane. By default, the All category is selected in Synaptic's left-pane, so that all available software packages are listed, As I'll discuss later in this chapter, performing a search in Synaptic creates a new category with the name of your search term(s) and hides all other default categories with the exception of the All category.
- **bottom-right pane:** Displays the description of any package that is currently selected in the upper-right pane.

Synaptic provides an excellent online manual, shown in Figure 20.13, that you can access by selecting the Help menu's Contents item.

FIGURE 20.13

The online manual for Synaptic



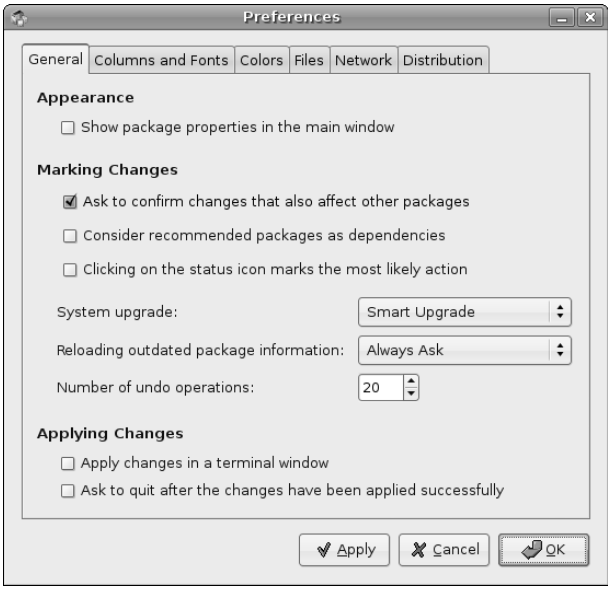
Because Synaptic includes thorough and up-to-date online help, I'm not going to bore you by repeating all of that. The next few sections focus on how to do common tasks in Synaptic, explaining how to configure Synaptic and how to perform basic package management tasks such as searching for and installing packages using Synaptic.

Configuring Synaptic Preferences

The entries on Synaptic's Settings menu enable you to set default values for using Synaptic, identify the repositories available to Synaptic, configure how searches work in Synaptic, configure how the icons are displayed in the Synaptic toolbar, and set internal variables used by Synaptic. Most of the settings that you will want to change are located on the Preferences dialog, which you can display by selecting the Settings menu's Preferences option. Figure 20.14 shows Synaptic's Preferences dialog.

FIGURE 20.14

The Synaptic Preferences dialog



The Synaptic preferences that you will most commonly want to change are all located on the general tab of Synaptic's Preferences dialog. These are the following:

- **Consider recommended packages as dependencies:** Selecting this check box causes Synaptic to always install any packages that are associated with a package that you are installing, but which are identified as recommended, but not required, for package installation. As discussed in more detail in this chapter in the section entitled "Using aptitude to Install Recommended Software," such recommended packages are ones that you will probably want to use with the package that you are installing.
- **System upgrade:** This item should always be set to "Smart Upgrade," which tries to identify other packages that must be installed, updated, or removed when installing or upgrading any package. Synaptic is much more conservative about this than other utilities such as `apt-get` (where the `apt-get dist-upgrade` command is a more powerful equivalent for this Synaptic option. Even if you are a Synaptic fan, you will occasionally need to execute the `apt-get dist-upgrade` command in order to completely upgrade all of the packages on your system. When using Synaptic, symptoms of the need to use the `apt-get dist-upgrade` command are packages that are mysteriously identified as being held back and not updated — and, of course, the occasional pop-up message that explicitly tells you to run the `apt-get dist-upgrade` command.
- **Apply changes in a terminal window:** Selecting this option causes Synaptic to always display package installation/update status messages and results in a separate terminal window. As mentioned when discussing installing software using the `apt-get` and `aptitude` utilities, you may occasionally have to provide additional, configuration-related information during the installation process. In these cases, the Synaptic utility displays the same sort of information requests as those

displayed by the `apt-get` and `aptitude` utilities, as shown earlier in Figure 20.7. Whenever you see this sort of screen, you must answer any questions that it asks and supply any requested information before the installation of your package can complete. The Synaptic utility displays this information in the dialog that you see only when you expand the software installation status window, so it's a good idea to always monitor this window. If you choose not to and notice that Synaptic seems to have stopped installing things, expand this window to see if Synaptic is waiting for additional information.

Once you have change any setting on any of the tabs in Synaptic's Preferences dialog, click Apply to make sure that your changes are saved, and then click OK to close the Preferences dialog and return to the main Synaptic dialog.

Searching for Software in Synaptic

One of Synaptic's most powerful capabilities is providing a powerful and usable package search capability that doesn't involve a command line. To locate packages that are relevant to one or more keywords, click the Search button in the Synaptic toolbar. The search dialog displays, as shown in Figure 20.15.

FIGURE 20.15

Synaptic's Find dialog

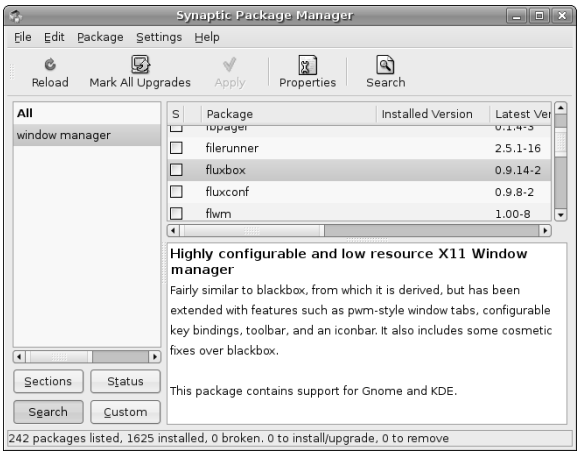
Most examples of installing software throughout this book have relied on Synaptic's search capabilities. To locate matching, available packages, you have to make sure that you're searching the right portion of the information that Synaptic has about packages. In almost all cases, you will want to make sure that the Look in value in Synaptic's search dialog is set to Description and Name so that the search looks for matches for your keyword(s) in both package descriptions and package names. However, you may occasionally want to search other portions of the package information that is available via Synaptic. Other search possibilities are the following:

- **Name:** searches only package names for your keyword
- **Maintainer:** looks for a package maintained by one your favorite Ubuntu rock star
- **Version:** finds specific versions, which can be useful when looking for all packages that take their version number from higher-level packages, such as GNOME versions
- **Dependencies:** identifies packages that depend on a given package
- **Provided Packages:** identifies packages based on other packages that they also install.

To search for packages in Synaptic, enter one or more keywords in Synaptic's search dialog and click Search (or press Return). Synaptic will create new category in the left pane with the name of your search term(s) and hides all other default categories with the exception of the All category. Once the search completes, the new category is selected and any matching packages display in the upper-right pane. Figure 20.16 shows the results of a search for "window manager."

FIGURE 20.16

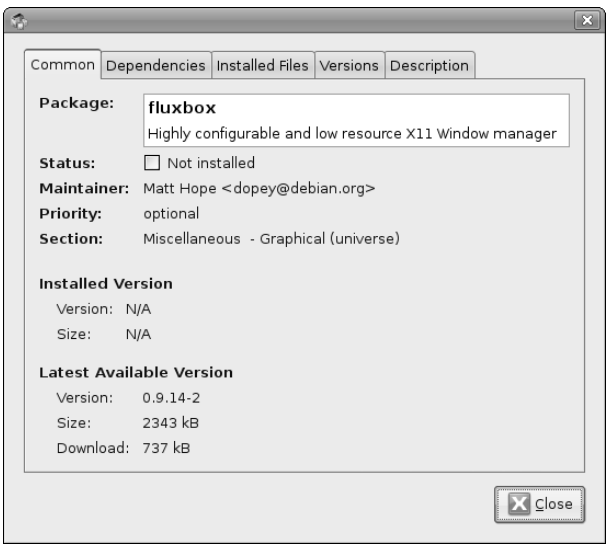
Search results in Synaptic



To select a package for installation, scroll down in the upper-right pane until you see the name of the package that you want to install. You can select the name of any package in the upper-right pane to see the description of that package, and can also click the Properties button in Synaptic's toolbar (or select the properties item from the pop-up menu after right-clicking its name) to display a Properties dialog that shows detailed information about the selected item. Figure 20.17 shows the Properties dialog for the package that provides the fluxbox window manager, a lightweight window manager that I like to use on laptops.

FIGURE 20.17

A package Properties dialog in Synaptic



The Common tab, shown by default when you display a package properties dialog, provides general information about the package. The Dependencies tab provides useful information about required, recommended, suggested, and conflicting packages. The Installed Files tab is also very useful, but only contains information on packages that are already installed. (To see information about the files that are included in a package that is not installed, you can use the apt-file application, which is discussed earlier in this chapter in the section entitled "Determining What Package Provides a Missing File.")

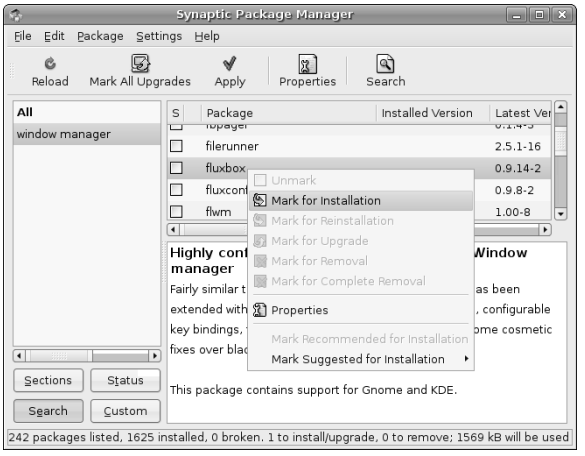
After examining any properties of a selected package, click Close to close the properties dialog and return to the main Synaptic dialog.

Installing Packages in Synaptic

After locating a package that you want to install, right-click on its name in the upper-right pane and select Mark for installation from the pop-up menu, as shown in Figure 20.18.

FIGURE 20.18

Marking a package for installation in Synaptic



If installing the selected package requires that other packages be installed, Synaptic displays a dialog like that shown in Figure 20.19.

This dialog lists all packages whose installation is required by the package that you have selected. If you checked the Consider recommended packages as dependencies item in Synaptic's Preferences dialog, this list will also include any packages that are recommended, but not required, for installation. Click mark to accept the installation of these other packages and close this dialog.

Next, right-click the package name again and check whether any packages are listed on a sub-menu of the Mark Suggested for Installation command at the bottom of Synaptic's pop-up menu, as shown in Figure 20.20.

FIGURE 20.19

Identifying other packages required for installation

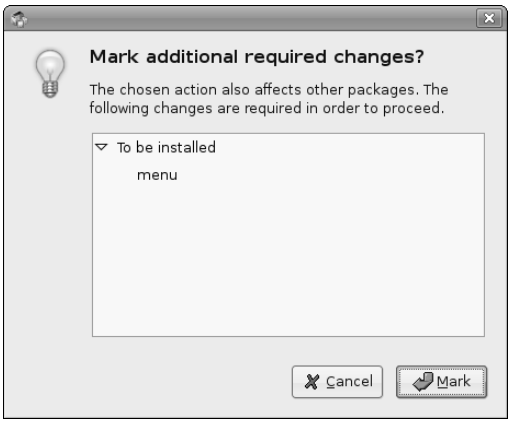
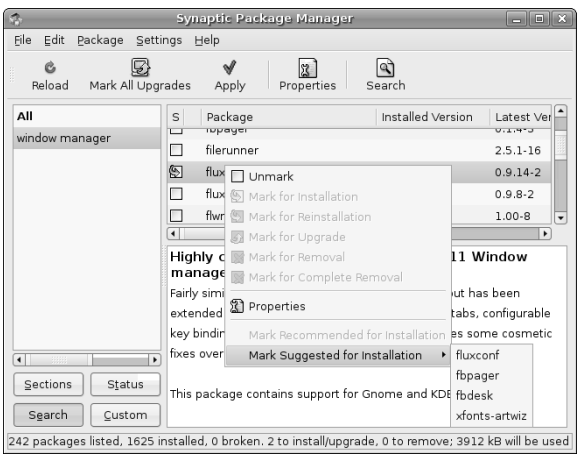


FIGURE 20.20

Marking suggested packages for installation in Synaptic

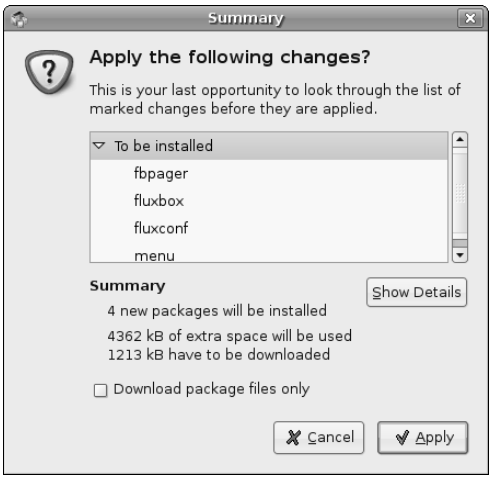


This sub-menu identifies packages that are normally installed with the primary package that you have selected for installation, but which are not required or expected. You can select any of the additional packages listed on this menu (one by one) to also schedule them for installation.

Once you have selected any suggested packages that you want to install with the primary package that you have selected for installation, click Apply in the Synaptic toolbar to begin the actual installation process. This displays the dialog shown in Figure 20.21, which summarizes all of the activities associated with installing the selected package(s).

FIGURE 20.21

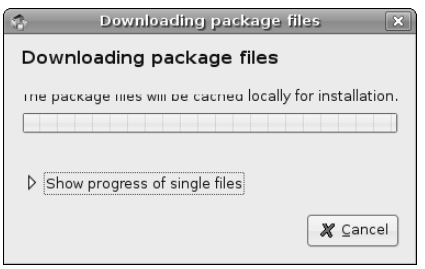
A package installation summary in Synaptic



To proceed, click Apply. At this point, Synaptic begins retrieving the selected packages from the repository where they are located, displaying the progress dialog shown in Figure 20.22.

FIGURE 20.22

The download status dialog during Synaptic package installation



Once the download process completes, Synaptic begins installing and configuring the downloaded packages. If you did not select the Apply changes in a terminal window item in Synaptic's Preferences dialog, Synaptic displays a simple status dialog, as shown in Figure 20.23.

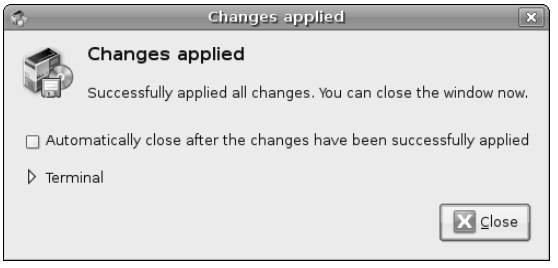
You can display a small terminal window that lists the status of Synaptic's installation and configuration processes by clicking the arrow to the right of the Terminal option on this dialog. When the installation and configuration of your packages completes, the dialog shown in Figure 20.24 displays.

FIGURE 20.23

A simple installation and configuration status dialog in Synaptic

FIGURE 20.24

A simple completion dialog in Synaptic

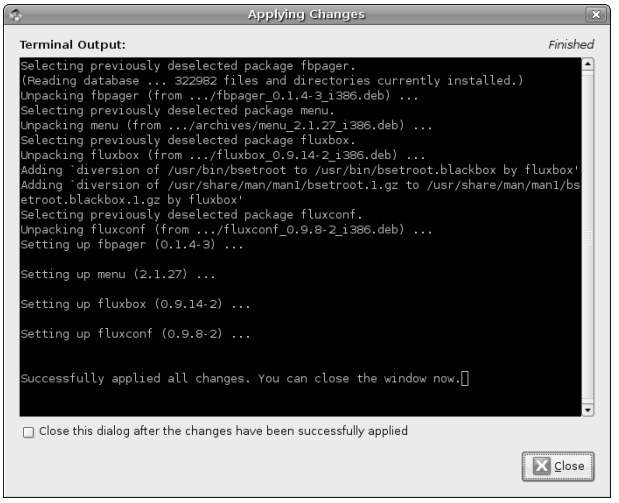


When using the Synaptic utility to install software, without having configured Synaptic to always display configuration status in a Terminal window, it is useful to display a Terminal to see information about the status of software installation on your system. You may occasionally have to provide additional, configuration-related information during the installation process. In this case, the Synaptic utility displays the same sort of information requests as those displayed by the `apt-get` and `aptitude` utilities, as shown earlier in Figure 20.7. Whenever you see this sort of screen, you must answer any questions that it asks and supply any requested information before the installation of your package can complete. The Synaptic utility displays this information in the dialog that you see only when you expand the software installation Terminal window, so it's a good idea to always monitor this window. If you choose not to and notice that Synaptic seems to have stopped installing things, expand this window to see if Synaptic is waiting for additional information.

If you selected the Apply changes in a terminal window item in Synaptic's Preferences dialog, Synaptic displays a single large terminal window during its installation and configuration phases, shown in Figure 20.25, instead of the dialogs shown in Figures 20.23 and 20.24.

FIGURE 20.25

A terminal window showing installation and configuration status



Regardless of which of the dialogs you see when package installation and configuration completes, click Close to close that dialog and return to Synaptic's main dialog.

Once you have finished installing any packages that you are interested in, you can exit from Synaptic by selecting the File menu's Quit command (or pressing the Control-q key sequence).

Removing Packages in Synaptic

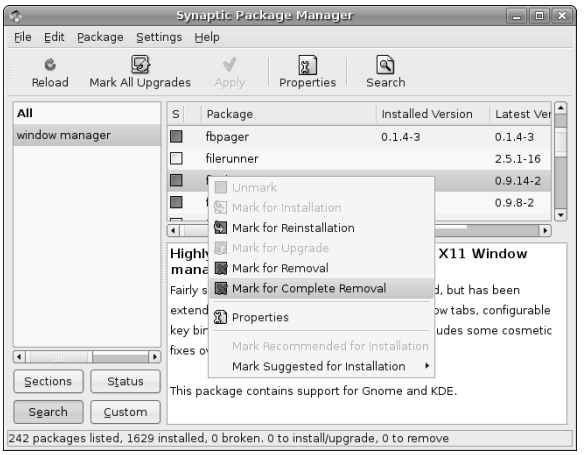
Removing software in Synaptic is very similar to the installation process. You must first locate the package(s) that you want to remove by selecting the appropriate category from Synaptic's left pane and scrolling through the package lists in the upper-right pane, or by searching for that package and then selecting it from the upper-right pane. Either way, once you have located the package that you want to remove, right-click on its name to display the pop-up menu shown in Figure 20.26.

Marking a package for complete removal removes the package and any configuration files for the application that you may have modified, while Mark for Removal removes the package but leaves the configuration files on your system.

TIP **Unfortunately, Synaptic does not automatically remove other packages that were required by a package that you are selecting for removal, or which you installed as a result of their being suggested for installation. You must select all of these packages manually in order to remove them. You can also not worry about them and use the process described in the last section of this chapter, "Keeping your System Lean, Mean, and Pristine" to remove them at some point in the future.**

FIGURE 20.26

Marking a package for complete removal in Synaptic



Once you have selected any packages that you want to remove and marked them for removal, click Apply to proceed with the installation process. This displays the confirmation dialog shown in Figure 20.27.

FIGURE 20.27

The confirmation dialog for package removal in Synaptic

Click Apply to proceed. As the removal process proceeds, Synaptic dialog displays status dialogs similar to those shown during the package installation process, as described in the previous section. Once package removal completes, click Close to close the final status dialog and return to Synaptic's primary dialog.

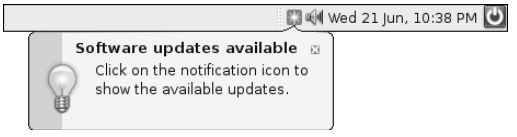
Once you have finished removing packages from your system, you can exit from Synaptic by selecting the File menu's Quit command (or pressing the Control+q key sequence).

Using the Ubuntu Update Manager

Ubuntu's Update Manager checks the active repositories listed in your `/etc/apt/sources.list` file and notifies you if updated to any installed packages are available. The Update Manager uses an associated panel applet to let you know when updates are available by displaying a small orange icon and a pop-up dialog when updates are found. Figure 20.28 shows this pop-up and the icon that is associated with the availability of system updates.

FIGURE 20.28

The icon and dialog that notify you of available updates



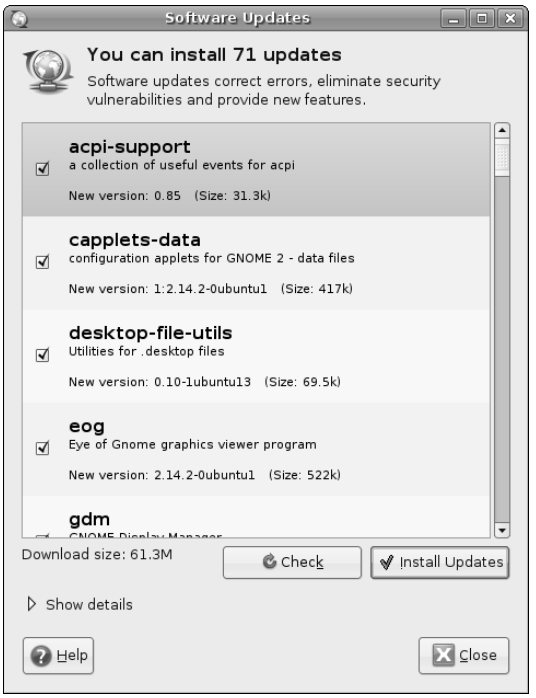
Clicking on this icon starts the Update Manager itself, which is shown in Figure 20.29.

Using the Update manager is simple — just click Install Updates. The Update manager then automatically retrieves, installs and configures all available updated packages, displaying progress and status dialogs that are very similar to those used by the Synaptic Package Manager. When the update process completes, click Close to terminate the Update Manager and begin doing real work.

TIP **As when using the Synaptic utility, you should watch for configuration questions or requests for additional information that are required during the package update process. If the update process seems to have halted or is taking a long time, click the Terminal button in the Update manager's status dialogs to see if any configuration questions are pending.**

FIGURE 20.29

The Ubuntu Update Manager



Converting Packages from Other Package Formats

Though the Ubuntu repositories contain most of the add-on software packages that anyone could ever want, there are cases when you must deal with pre-packed software that is provided in the package formats used by distributions other than Ubuntu. This is often the case with commercial software whose Linux releases are distributed in formats such as RPM and LSB. Luckily, an easy solution to this sort of problem is provided by the `alien` application (`www.kitenet.net/~joey/code/alien.html`), which is a Perl script that can covert between different package formats. Most importantly, `alien` can convert most package formats to the DEB package format that is used by the Ubuntu and Debian installation and update tools that are discussed in this chapter. Depending on the format of the package that you are converting from, `alien` may need some additional software to be installed on your Ubuntu system — more about that in a second.

You must run `alien` as root (preferably using the `sudo` program) so that it can correctly set ownership and file permissions on the files in the converted package.

The alien application can convert packages to and from the following formats:

- **DEB:** When converting packages to DEB format, the `debhelper`, `dpkg`, `dpkg-dev`, `gcc`, and `make` packages must be installed on your system. (The `gcc`, `make`, and `dpkg-dev` packages are installed as requirements for the build-essential package, as discussed in Chapter 18. Installing `alien` installs the `debhelper` package.) If you're stuck with using some distribution other than Ubuntu on one or more of your systems, you can use `alien` to convert DEB packages to the appropriate format for your other distribution(s) until you can replace them.
- **LSB:** The package format used to distribute applications and entire distributions that are compliant with the Linux Standard Base specification (`www.freestandards.org/en/LSB`). Because the LSB specification mandates RPM packages, LSB packages are a superset of RPM packages, the RPM Package Manager software must be installed on your Ubuntu system in order to convert LSB packages. (The rpm software is automatically installed for you when you install `alien`, because it is listed as a requirement for the `alien` package.) If you are converting an LSB package that depends on other packages, the conversion process will correctly translate those dependencies, but will also introduce a new dependency on the `lsb` package, which is available in the Ubuntu repositories. You should also install the `lsb-rpm` package, which is suggested but not required by the `alien` package, if you are converting from LSB packages.
- **PKG:** The package format used by Solaris. Converting PKG files into other package formats requires Solaris-only tools such as `pkginfo` and `pkgtrans`. If you also install the rpm software from one of the Sun Free Software CDs, you can then use `alien` on your Solaris system to convert PKG packages into RPMs. This is probably only useful for packages composed of scripts, source code, or command files, since you can't execute Solaris binaries on a Linux system.
- **RPM:** The package format used by distributions such as Red Hat, Fedora Core, Mandrake, Yellow Dog, and so on. To convert packages from the RPM format, the RPM Package Manager, formerly known as the Red Hat Package Manager, software must be installed on your Ubuntu system. (The rpm software is automatically installed for you when you install `alien`, because it is listed as a requirement for the `alien` package.)
- SLP: The package format used by the extinct Stampede Linux distribution, which ceased development in 2002. I can't believe that this option is still actively supported, but since Stampede Linux is dead, I don't suppose much is changing in its package format.
- TGZ: The software distribution format used by most open source software projects and the Slackware Linux distribution, a TGZ file is a `tar` archive that has been compressed using the `gzip` compression tool, whether from the command line or by `tar` itself when creating the archive. Compressed tar archives are often simply referred to as *compressed tarballs*. Package files that are generated from TGZ files simply contain the same files as the compressed tarball, plus the package database information for the target package format.

TIP **For more information about Linux package formats and a general comparison of their capabilities, see** `http://kitenet.net/~joey/pkg-comp`**.**

Using `alien` to convert from one package format to another is easy. The `alien` application provides options such as `--to-deb`, `--to-lsb`, `--to-rpm`, and `--to-tgz` to specify the target output format. The following is an example of converting the RPM package for Adobe Acrobat Reader, which some of you may have heard of, to DEB format:

```
$ sudo alien --scripts --to-deb AdobeReader_enu-7.0.0-2.i386.rpm
Password:
adobereader-enu_7.0.0-3_i386.deb generated
```

Not much to see there, actually. The `--scripts` option tells `alien` to convert any pre- or post-installation scripts found in the package, though the validity of the converted scripts can't be guaranteed — you should watch carefully to make sure that nothing goes amiss when you install the converted package. The `--to-deb` option tells `alien` to generate a Debian package from the input package. If you want to see detailed output, you can add the `-v` option to the `alien` command line, but you're likely to see way more output than you actually care about.

Before installing a converted package, you can verify its integrity using `dpkg`, as in the following example:

```
$ dpkg --info adobereader-enu_7.0.0-3_i386.deb
 new debian package, version 2.0.
 size 37932742 bytes: control archive= 7436 bytes.
     484 bytes,    11 lines      control
   20061 bytes,   207 lines      md5sums
    1340 bytes,    60 lines   *  postinst             #!/bin/sh
     980 bytes,    45 lines   *  postrm               #!/bin/sh
     211 bytes,     7 lines      shlibs
 Package: adobereader-enu
 Version: 7.0.0-3
 Section: alien
 Priority: extra
 Architecture: i386
 Installed-Size: 92316
 Maintainer: root <root@vmdesktop>
 Description: Adobe Reader for Linux. An application that reads a PDF
document.
  Adobe Reader 7.0.0 can read documents in PDF format. Adobe Reader
also allows you to search within PDF files, search for PDF files on the
internet and participate in collaborative document reviews.
  .
  (Converted from a rpm package by alien version 8.64.)
```

This matches quite nicely with information about the original RPM package that was produced on one of my other systems:

```
$ rpm -q --info -v -p AdobeReader_enu-7.0.0-2.i386.rpm
Name        : AdobeReader_enu              Relocations:
/usr/local/Adobe/Acrobat7.0
Version     : 7.0.0                             Vendor: Adobe Systems,
Incorporated
Release     : 2                              Build Date: Mon 28 Mar 2005 06:34:42
AM EST
Install date: (not installed)                Build Host:
acrolinux2.corp.adobe.com
Group       : Applications/Publishing        Source RPM: AdobeReader_enu-
7.0.0-2.src.rpm
Size        : 97613446                          License: Commercial
Signature   : (none)
Packager    : Adobe Systems, Incorporated
URL         : www.adobe.com
```

```
Summary     : Adobe Reader for Linux. An application that reads a PDF document.
Description :

Adobe Reader 7.0.0 can read documents in PDF format. Adobe Reader also
allows you to search within PDF files, search for PDF files on the
internet and participate in collaborative document reviews.
Distribution: (none)
```

For more information about the `alien` package, see its online reference information, which is available by typing the man `alien` command (after installing the package, of course).

WARNING **You should never use `alien` in order to install packages containing low-level system utilities or system libraries. Though they may convert and install successfully, these packages will probably not work correctly on your Ubuntu system and you risk reducing your Ubuntu system to a dysfunctional heap of slag that will not boot correctly.**

Keeping your System Lean, Mean, and Pristine

If you like to play with different software packages as much as I do, you'll find yourself installing lots of random packages on your system, playing with them for a while, and then forgetting about them unless they solve some major problem for you. You may occasionally even be thoughtful enough to remove packages that you aren't planning on using anymore. Unfortunately, this doesn't remove packages that were required, recommended, or suggested by the packages that you've removed (unless you use `aptitude` to both install and remove those package).

Every now and then, it is therefore useful to run a utility that scans all of the packages that are installed on your system and looks for packages that are not used or required by any other package on your system, and which are also not a part of some system installation process. Good examples of programs that you can use to check your system in this way are the `deborphan` and `debfoster` packages. Neither of these packages is installed by default on an Ubuntu system, but you can easily install either of them using your favorite package installation utility.

The core difference between these two packages is that the `deborphan` utility simply produces a list of packages that are not used or referred to by any system package on your system, while the `debfoster` package builds its own database of which packages are required, tries to be smart about things that you may not need, and also asks lots of questions the first time you run it. For all of these reasons, I prefer the `deborphan` package, but I suggest that you investigate both and see which utility best suits your needs and modus operandi.

As mentioned previously, the `deborphan` utility simply lists packages that are no longer required by or associated with any other package. To do something useful with this list, you can either redirect it into a file and use that file as the basis for a shell script that will remove the packages that you actually want to remove. Another solution, much more wizardly, is to feed this list to the `apt-get remove` command using the Linux `xargs` command, as in the following example:

```
$ sudo deborphan | xargs apt-get remove --purge -y
```

Before running this sort of command, you should examine the `deborphan` output to make sure that it isn't removing anything that you still want.

Running either the `deborphan` or `debfoster` commands periodically will help you reduce the amount of software package detritus that tends to accumulate on any running system. Keeping your system as free of

unneeded packages as possible can help guarantee that the maximum amount of free disk space is available on your system, which you can then devote to "real work" such as your online music archives or collection of artistic digital photographs.

Summary

Package management is a wonderful thing, enabling your system to identify installed packages and, with the help of excellent package formats such as DEB, identify and satisfy any requirements for successfully installing a completely functional software package. This chapter discussed Ubuntu's four primary package management utilities (`dpkg`, `apt-get`, `aptitude`, and `synaptic`), and discussed how you can use these and related utilities to install and query packages, figure out other packages that you need, and many more interesting package management tricks.

Chapter 21 continues with system-related topics, discussing how to create and manage users and groups on your Ubuntu system, and why this is important. It also discusses advanced topics related to file and directory protection, such as Access Control Lists, that are above and beyond the traditional user/group protection model but give you even finer control over who can access your data (and don't require system administration privileges to do so).