Learn Linux, 101: RPM and YUM package management

Add new software and keep your system current

Ian Shields
Senior Programmer
IBM

19 March 2012
(First published 11 May 2010)

Learn how to install, upgrade and manage packages on your Linux® system. This article focuses on the Red Hat Package Manager (RPM) developed by Red Hat, as well as the Yellowdog Updater Modified (YUM) originally developed to manage Red Hat Linux systems at Duke University's Physics department. You can use the material in this article to study for the LPI 101 exam for Linux system administrator certification, or just to explore the best ways to add new software and keep your system current. [*Typographical errors noted by alert readers (see Comments at the end of this article) have been corrected, thanks! --Ed.*]

View more content in this series

About this series

This series of articles helps you learn Linux system administration tasks. You can also use the material in these articles to prepare for Linux Professional Institute Certification level 1 (LPIC-1) exams.

See our developerWorks roadmap for LPIC-1 for a description of and link to each article in this series. The roadmap is in progress and reflects the latest (April 2009) objectives for the LPIC-1 exams: as we complete articles, we add them to the roadmap. In the meantime, though, you can find earlier versions of similar material, supporting previous LPIC-1 objectives prior to April 2009, in our LPI certification exam prep tutorials.

Overview

In this article, learn to use the RPM and YUM tools to manage the packages on your Linux system. Learn to:

- Install, reinstall, upgrade, and remove packages using RPM and YUM
- Obtain information about RPM packages including version, status, dependencies, integrity, and signatures
- Determine what files a package provides, as well as find which package a specific file comes from.

This article helps you prepare for Objective 102.5 in Topic 102 of the Linux Professional Institute's Junior Level Administration (LPIC-1) exam 101. The objective has a weight of 3.

Prerequisites

To get the most from the articles in this series, you should have a basic knowledge of Linux and a working Linux system on which you can practice the commands covered in this article. Sometimes different versions of a program will format output differently, so your results may not always look exactly like the listings and figures shown here. In particular, much of the output we show is highly dependent on the packages that are already installed on our systems. Your own output may be quite different, although you should be able to recognize the important commonalities.

Introducing package management

In the past, many Linux programs were distributed as source code, which a user would build into the required program or set of programs, along with the required man pages, configuration files, and so on. Nowadays, most Linux distributors use prebuilt programs or sets of programs called *packages*, which ship ready for installation on that distribution. In this article, you will learn about *package management tools* that help you install, update, and remove packages. This article focuses on the **Red Hat Package Manager (RPM)**, which was developed by Red Hat, as well as the **Yellowdog Updater Modified (YUM)**, which was originally developed to manage Red Hat Linux systems at Duke University's Physics department. Another article in this series, "Learn Linux 101: Debian package management," covers the package management tools used on Debian systems.

From a user perspective, the basic package management function is provided by commands. As Linux developers have striven to make Linux easier to use, the basic tools have been supplemented by other tools, including GUI tools, which hide some of the complexities of the basic tools from the end user. In this article and in the article on Debian package management, we focus on the basic tools, although we mention some of the other tools so you can pursue them further.

RPM, YUM, and APT (for Debian systems) have many similarities. All can install and remove packages. Information about installed packages is kept in a database. All have basic command-line functionality, while additional tools can provide more user-friendly interfaces. All can retrieve packages from the Internet.

When you install a Linux system, you typically install a large selection of packages. The set may be customized to the intended use of the system, such as a server, desktop, or developer workstation. And at some time you will probably need to install new packages for added

functionality, update the packages you have, or even remove packages that you no longer need or that have been made obsolete by newer packages. Let's look at how you do these tasks, and at some of the related challenges such as finding which package might contain a particular command.

RPM

Red Hat introduced RPM in 1995. RPM is now the package management system used for packaging in the Linux Standard Base (LSB). The `rpm` command options are grouped into three subgroups for:

- Querying and verifying packages
- Installing, upgrading, and removing packages
- Performing miscellaneous functions

We will focus on the first two sets of command options in this article. You will find information about the miscellaneous functions in the man pages for RPM.

We should also note that `rpm` is the command name for the main command used with RPM, while *.rpm* is the extension used for RPM files. So "an rpm" or "the xxx rpm" will generally refer to an RPM file, while `rpm` will usually refer to the command.

YUM

YUM adds automatic updates and package management, including dependency management, to RPM systems. In addition to understanding the installed packages on a system, YUM, like the Debian Advanced Packaging Tool (APT), works with *repositories*, which are collections of packages, typically accessible over a network connection.

Installing RPM packages

Suppose you want to learn Lisp, and a colleague tells you to use the `gcl` command. You might try `gcl --help`, or you might try `which gcl`, or `type gcl`. But if your system can't find `gcl`, you might see output similar to that shown in Listing 1.

Listing 1. Missing gcl command

```
[ian@echidna ~]$ gcl --help
bash: gcl: command not found

[ian@echidna ~]$ which gcl
/usr/bin/which: no gcl in (/usr/lib64/qt-3.3/bin:/usr/kerberos/sbin:/usr/kerber
os/bin:/usr/lib64/ccache:/usr/local/bin:/usr/bin:/bin:/usr/local/sbin:/usr/sbin:/
sbin:/home/ian/bin)

[ian@echidna ~]$ type gcl
bash: type: gcl: not found
```

You might check back with your colleague to find out which package to install, or you might just guess that the `gcl` command is in the gcl package. This is often a good guess, but not always the right one. We'll see later how to find the right package. In this case, you do need the gcl package.

Assuming you have downloaded or otherwise acquired a copy of the package, you might try installing it using the `rpm` command with the `-i` (for install) option, as shown in Listing 2.

Listing 2. Installing gcl with rpm - take 1

```
[root@echidna ~]# rpm -i gcl-2.6.8-0.6.20090701cvs.fc12.x86_64.rpm
error: Failed dependencies:
gcl-selinux is needed by gcl-2.6.8-0.6.20090701cvs.fc12.x86_64
```

The `rpm` command knows that the package has a dependency, but unfortunately, it won't help you resolve that dependency. You will need to get the dependent package or packages, try again, and see if there are additional dependencies—and keep doing this until all dependencies are satisfied. One good thing is that you can give the `rpm` command a list of packages to install and it will install them all in the right order if all dependencies are satisfied. So you at least don't have to manually install each piece in the right order.

If you've used Debian's APT, by this time you're probably wishing you had something like the `apt-get` command, which would simply go and find what you need, including dependencies, and just install it. For RPM-based systems, YUM (or Yellowdog Updater Modified) provides just such a function. Listing 3 shows how to install gcl and the required gcl-selinux prerequisite using the `yum` command with the `install` option.

Listing 3. Installing gcl using yum

```
[root@echidna ~]# yum install gcl
Loaded plugins: presto, refresh-packagekit
Setting up Install Process
Resolving Dependencies
--> Running transaction check
---> Package gcl.x86_64 0:2.6.8-0.7.20100201cvs.fc12 set to be updated
--> Processing Dependency: gcl-selinux for package: gcl-2.6.8-0.7.20100201cvs.fc12.x86_64
--> Running transaction check
---> Package gcl-selinux.x86_64 0:2.6.8-0.7.20100201cvs.fc12 set to be updated
--> Finished Dependency Resolution

Dependencies Resolved

================================================================================
 Package           Arch         Version                          Repository     Size
================================================================================
Installing:
 gcl               x86_64       2.6.8-0.7.20100201cvs.fc12       updates       6.3 M
Installing for dependencies:
 gcl-selinux       x86_64       2.6.8-0.7.20100201cvs.fc12       updates        17 k

Transaction Summary
================================================================================
Install       2 Package(s)
Upgrade       0 Package(s)

Total download size: 6.4 M
Installed size: 40 M
Is this ok [y/N]: y
Downloading Packages:
Setting up and reading Presto delta metadata
updates/prestodelta                                      | 964 kB     00:01
Processing delta metadata
Package(s) data still to download: 6.4 M
(1/2): gcl-2.6.8-0.7.20100201cvs.fc12.x86_64.rpm         | 6.3 MB     00:12
```

```
(2/2): gcl-selinux-2.6.8-0.7.20100201cvs.fc12.x86_64.rpm      |  17 kB     00:00
--------------------------------------------------------------------------------
Total                                           398 kB/s | 6.4 MB     00:16
Running rpm_check_debug
Running Transaction Test
Transaction Test Succeeded
Running Transaction
  Installing     : gcl-selinux-2.6.8-0.7.20100201cvs.fc12.x86_64                1/2
  Installing     : gcl-2.6.8-0.7.20100201cvs.fc12.x86_64                        2/2

Installed:
  gcl.x86_64 0:2.6.8-0.7.20100201cvs.fc12

Dependency Installed:
  gcl-selinux.x86_64 0:2.6.8-0.7.20100201cvs.fc12

Complete!
```

The output in Listing 3 shows that YUM has found the gcl.x86_64 0:2.6.8-0.7.20100201cvs.fc12 and gcl-selinux.x86_64 0:2.6.8-0.7.20100201cvs.fc12 in a repository called "updates" (more on that shortly), and determined the total download size. After you respond "y" to agree to the transaction, it downloaded both packages, and then installed the dependency, followed by gcl. You will learn more about dependencies later in this article.

Note: In Listing 3, YUM found a later version of the gcl package than the one we attempted to install in Listing 2. You will usually want the latest version of a package, but you can provide additional qualification if you need an earlier version. See the section on specifying package names in the man pages for the yum command.

Package locations

In the previous section, you learned how to install an RPM package. But where do the packages come from? How does `yum` know where to download packages from? The starting point is the /etc/yum.repos.d/ directory, which usually contains several *repo* files. This is the default location for repos, but other locations may be specified in the YUM configuration file, normally /etc/yum.conf. Listing 4 shows the fedora-updates.repo corresponding to the location from which we installed gcl on our Fedora 12 system.

A typical repo file is divided into three sections, one for normal packages, one for debug packages, and the last for source packages. Usually there will be several copies of a distribution's packages available from different locations, or *mirrors*. So the repo file tells `yum` where to find the latest list of mirrors for each section. Note that the distribution release level and machine architecture are parameterized, so `yum` would download the list for my x86_64 Fedora 12 system from https://mirrors.fedoraproject.org/metalink?repo=updates-released-f12&arch=x86_64.

In addition to the repository location, the repo file tells whether a particular repository is enabled and whether GPG signatures should be used to check the downloaded packages.

Listing 4. /etc/yum.repos.d/*.repo

```
[ian@echidna ~]$ cat /etc/yum.repos.d/fedora-updates.repo
[updates]
name=Fedora $releasever - $basearch - Updates
```

```
failovermethod=priority
#baseurl=http://download.fedoraproject.org/pub/fedora/linux/updates/$releasever
/$basearch/
mirrorlist=https://mirrors.fedoraproject.org/metalink?repo=updates-released-f$r
eleasever&arch=$basearch
enabled=1
gpgcheck=1
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-fedora-$basearch

[updates-debuginfo]
name=Fedora $releasever - $basearch - Updates - Debug
failovermethod=priority
#baseurl=http://download.fedoraproject.org/pub/fedora/linux/updates/$releasever
/$basearch/debug/
mirrorlist=https://mirrors.fedoraproject.org/metalink?repo=updates-released-deb
ug-f$releasever&arch=$basearch
enabled=0
gpgcheck=1
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-fedora-$basearch

[updates-source]
name=Fedora $releasever - Updates Source
failovermethod=priority
#baseurl=http://download.fedoraproject.org/pub/fedora/linux/updates/$releasever
/SRPMS/
mirrorlist=https://mirrors.fedoraproject.org/metalink?repo=updates-released-sou
rce-f$releasever&arch=$basearch
enabled=0
gpgcheck=1
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-fedora-$basearch
```

YUM and RPM use a local database to determine what packages are installed. The metadata about packages that is stored in the local database is retrieved from the enabled repositories. Although you will seldom need to worry about the local database, you use the command `yum clean` to clean out various parts of the locally stored information and `yum makecache` to create the information in your local database for the enabled repos. You might do this if you change your repo configuration, for example.

Removing RPM packages

If you want to remove a package, you can use the `remove` option of `yum`, or the `-e` option of `rpm`. A test run to remove gcl using `rpm -e` is shown in Listing 5. If the package can be removed, there is no output.

Listing 5. Test removal of gcl

```
[root@echidna ~]# rpm -e --test gcl
```

Unlike the simulated removal of Debian packages using `apt-get`, the RPM system does not maintain information on packages that were automatically added, so there is no trivial way to find out which dependencies might also be removed. However, if you specify multiple packages for removal on a single command, then packages without dependencies will be removed before packages that have dependencies.

When you remove packages using `rpm`, there is no prompt before the packages are removed, unlike when you install packages. However, if you attempt to remove a package that is required for

some other package, the operation is not performed and you get an error message as shown in Listing 6.

Listing 6. Removing a dependent package with rpm

```
[root@echidna ~]# rpm -e gcl-selinux
error: Failed dependencies:
 gcl-selinux is needed by (installed) gcl-2.6.8-0.7.20100201cvs.fc12.x86_64
```

If you use `yum remove` instead, then you will be prompted after the transaction tests are performed. If the package you are trying to remove is a dependent package for some other installed packages, then YUM will offer to remove those as well as the dependent package, as shown in Listing 7.

Listing 7. Removing a dependent package with yum

```
[root@echidna ~]# yum remove gcl-selinux
Loaded plugins: presto, refresh-packagekit
Setting up Remove Process
Resolving Dependencies
--> Running transaction check
---> Package gcl-selinux.x86_64 0:2.6.8-0.7.20100201cvs.fc12 set to be erased
--> Processing Dependency: gcl-selinux for package: gcl-2.6.8-0.7.20100201cvs.fc12.x86_64
--> Running transaction check
---> Package gcl.x86_64 0:2.6.8-0.7.20100201cvs.fc12 set to be erased
--> Finished Dependency Resolution

Dependencies Resolved

================================================================================
 Package          Arch        Version                          Repository     Size
================================================================================
Removing:
 gcl-selinux      x86_64      2.6.8-0.7.20100201cvs.fc12       @updates       90 k
Removing for dependencies:
 gcl              x86_64      2.6.8-0.7.20100201cvs.fc12       @updates       40 M

Transaction Summary
================================================================================
Remove        2 Package(s)
Reinstall     0 Package(s)
Downgrade     0 Package(s)

Is this ok [y/N]: n
Exiting on user Command
Complete!
```

Upgrading RPM packages

Now that you know how to install and remove an RPM, let's look at upgrading RPM packages to a newer level. You can use `yum update` to update your entire system, or you can specify a single package or a wildcard specification. Listing 8 shows how to update all the packages whose names start with "gr". Note the use of apostrophes to prevent shell expansion of the "*".

Listing 8. Updating using yum update

```
[root@echidna ~]# yum update 'gr*'
Loaded plugins: presto, refresh-packagekit
Setting up Update Process
Resolving Dependencies
```

```
--> Running transaction check
---> Package grep.x86_64 0:2.6.3-1.fc12 set to be updated
---> Package groff.x86_64 0:1.18.1.4-20.fc12 set to be updated
--> Finished Dependency Resolution

Dependencies Resolved

================================================================================
 Package         Arch             Version                    Repository         Size
================================================================================
Updating:
 grep            x86_64           2.6.3-1.fc12               updates           228 k
 groff           x86_64           1.18.1.4-20.fc12           updates           1.5 M

Transaction Summary
================================================================================
Install       0 Package(s)
Upgrade       2 Package(s)

Total download size: 1.7 M
Is this ok [y/N]: y
Downloading Packages:
Setting up and reading Presto delta metadata
Processing delta metadata
Download delta size: 854 k
http://fedora.fastsoft.net/pub/linux/fedora/linux/updates/12/x86_64/drpms/grep-2.5.3-
6.fc12_2.6.3-1.fc12.x86_64.drpm: [Errno 14] HTTP Error 404 : http://fedora.fastsoft.n
et/pub/linux/fedora/linux/updates/12/x86_64/drpms/grep-2.5.3-6.fc12_2.6.3-1.fc12.x86_
64.drpm
Trying other mirror.
(1/2): grep-2.5.3-6.fc12_2.6.3-1.fc12.x86_64.drpm             | 214 kB     00:00
(2/2): groff-1.18.1.4-18.fc12_1.18.1.4-20.fc12.x86_64.drpm    | 640 kB     00:00
Finishing rebuild of rpms, from deltarpms
<delta rebuild>                                               | 1.7 MB     00:02
Presto reduced the update size by 52% (from 1.7 M to 854 k).
Running rpm_check_debug
Running Transaction Test
Transaction Test Succeeded
Running Transaction
  Updating       : grep-2.6.3-1.fc12.x86_64                                  1/4
  Updating       : groff-1.18.1.4-20.fc12.x86_64                             2/4
  Cleanup        : grep-2.5.3-6.fc12.x86_64                                  3/4
  Cleanup        : groff-1.18.1.4-18.fc12.x86_64                             4/4

Updated:
  grep.x86_64 0:2.6.3-1.fc12            groff.x86_64 0:1.18.1.4-20.fc12

Complete!
```

If you know where the RPM files are located, or have downloaded them, you can also update using the `rpm` command. This is similar to installing, except that you use the `-U` or the `-F` option instead of the `-i` option. The difference between these two options is that the `-U` option will upgrade an existing package **or** install the package if it is not already installed, while the `-F` option will only upgrade or *freshen* a package that is already installed. Because of this, the `-U` option is frequently used, particularly when the command line contains a list of RPMs. This way, uninstalled packages are installed, while installed packages are upgraded. Two other options, `-v` (verbose) and `-h` (hash marks), are often used to give progress indication. Listing 9 shows how to update the vim-common, vim-enhanced, and vim-minimal packages using the `rpm` command. We have the vim-common and vim-enhanced packages already downloaded in root's home directory, while we retrieve the vim-minimal package from one of the update mirrors.

Listing 9. Updating packages with rpm

```
[root@echidna ~]# ls *.rpm
vim-common-7.2.411-1.fc12.x86_64.rpm  vim-enhanced-7.2.411-1.fc12.x86_64.rpm
[root@echidna ~]# rpm -Uvh *.rpm http://mirrors.usc.edu/pub/linux/distributions\
> /fedora/linux/updates/12/x86_64/vim-minimal-7.2.411-1.fc12.x86_64.rpm
Retrieving http://mirrors.usc.edu/pub/linux/distributions/fedora/linux/updates/12/x86
_64/vim-minimal-7.2.411-1.fc12.x86_64.rpm
Preparing...                ########################################### [100%]
   1:vim-common             ########################################### [ 33%]
   2:vim-enhanced           ########################################### [ 67%]
   3:vim-minimal            ########################################### [100%]
```

Querying RPM packages

In our examples you saw that installing an rpm with the `rpm` command requires the full name of the package file (or URL), such as gcl-2.6.8-0.6.20090701cvs.fc12.x86_64.rpm. On the other hand, installing with `yum`, or removing an rpm with either command requires only the package name, such as gcl. As with APT, RPM maintains an internal database of your installed packages, allowing you to manipulate installed packages using the package name. In this section, we look at some of the information that is available to you from this database using the `-q` (for *query*) option of the `rpm` command, or the associated `yum` queries.

The basic query simply asks if a package is installed, and, if so, what version. Add the `-i` option and you get information about the package. Note that you need to have root authority to install, upgrade, or remove packages, but non-root users can perform queries against the rpm database.

Listing 10. Displaying information about gcl

```
[ian@echidna ~]$ yum list gcl
Loaded plugins: presto, refresh-packagekit
Installed Packages
gcl.x86_64                  2.6.8-0.7.20100201cvs.fc12                 @updates
[ian@echidna ~]$ rpm -q gcl
gcl-2.6.8-0.7.20100201cvs.fc12.x86_64

[ian@echidna ~]$ yum info gcl
Loaded plugins: presto, refresh-packagekit
Installed Packages
Name       : gcl
Arch       : x86_64
Version    : 2.6.8
Release    : 0.7.20100201cvs.fc12
Size       : 40 M
Repo       : installed
From repo  : updates
Summary    : GNU Common Lisp
URL        : http://www.gnu.org/software/gcl/
License    : GPL+ and LGPLv2+
Description: GCL is a Common Lisp currently compliant with the ANSI standard.
           : Lisp compilation produces native code through the intermediary of
           : the system's C compiler, from which GCL derives efficient
           : performance and facile portability. Currently uses TCL/Tk as GUI.

[ian@echidna ~]$ rpm -qi gcl
Name        : gcl                           Relocations: (not relocatable)
Version     : 2.6.8                              Vendor: Fedora Project
Release     : 0.7.20100201cvs.fc12           Build Date: Tue 23 Mar 2010 03:20:36 PM EDT
Install Date: Wed 05 May 2010 01:01:34 PM EDT      Build Host: x86-02.phx2.fedoraproject.
org
Group       : Development/Languages         Source RPM: gcl-2.6.8-0.7.20100201cvs.fc12.sr
```

```
c.rpm
Size        : 41667750                         License: GPL+ and LGPLv2+
Signature   : RSA/8, Tue 23 Mar 2010 04:14:06 PM EDT, Key ID 9d1cc34857bbccba
Packager    : Fedora Project
URL         : http://www.gnu.org/software/gcl/
Summary     : GNU Common Lisp
Description :
GCL is a Common Lisp currently compliant with the ANSI standard.  Lisp
compilation produces native code through the intermediary of the
system's C compiler, from which GCL derives efficient performance and
facile portability. Currently uses TCL/Tk as GUI.
```

The more extensive listings show you some of the *tags* that can be associated with an RPM package. You will notice that `rpm` and `yum` show slightly different information in slightly different formats. For this article, we will stick to the basic output provided by standard command options. See the man page if you would like to use the `rpm --queryformat` option to build custom query output. Try running `rpm --querytags` if you want to know all the tags supported by your version of `rpm`.

As shown in Listing 10, you can use `yum` to list installed packages. You can also use it to list packages that have updates available, packages that are available for installation, and packages with other characteristics, such as obsolete, or recently added to a repository. You can even use `yum` to search for packages. In Listing 11, you see that the texmacs package is not installed, but is available from the fedora repository. If you search for "texmacs" you see four packages that mention it. You can easily see why the TeXmacs* packages were found. Use `yum info pydot` to find out why the pydot package is also mentioned.

Listing 11. Displaying information about gcl

```
[ian@echidna ~]$ yum list texmacs
Loaded plugins: presto, refresh-packagekit
Available Packages
TeXmacs.x86_64                          1.0.7.2-2.fc12                          fedora
[ian@echidna ~]$ yum search texmacs
Loaded plugins: presto, refresh-packagekit
================================ Matched: texmacs ================================
TeXmacs-devel.i686 : Development files for TeXmacs
TeXmacs-devel.x86_64 : Development files for TeXmacs
TeXmacs.x86_64 : Structured wysiwyg scientific text editor
pydot.noarch : Python interface to Graphviz's Dot language
```

For the remaining query examples, we will mostly use `rpm`, as it has a more extensive set of options. Many of the examples can also be done with `yum`, and `yum` has some capabilities that are not in the basic `rpm` options. See the man pages to learn more.

RPM packages and files in them

You will often want to know what is in a package or what package a particular file came from. To list the files in the gcl package, use the `-ql` option as shown in Listing 12. There are many files in this package, so we've only shown part of the output.

Listing 12. Displaying files in the gcl package

```
[ian@echidna ~]$ rpm -ql gcl
/usr/bin/gcl
```

```
/usr/lib/gcl-2.6.8
/usr/lib/gcl-2.6.8/clcs
/usr/lib/gcl-2.6.8/clcs/sys-proclaim.lisp
/usr/lib/gcl-2.6.8/cmpnew
/usr/lib/gcl-2.6.8/cmpnew/gcl_cmpmain.lsp
/usr/lib/gcl-2.6.8/cmpnew/gcl_cmpopt.lsp
/usr/lib/gcl-2.6.8/cmpnew/gcl_collectfn.lsp
.
.
.
/usr/share/info/gcl-tk.info.gz
/usr/share/info/gcl.info-1.gz
/usr/share/info/gcl.info-2.gz
/usr/share/info/gcl.info-3.gz
/usr/share/info/gcl.info-4.gz
/usr/share/info/gcl.info-5.gz
/usr/share/info/gcl.info-6.gz
/usr/share/info/gcl.info-7.gz
/usr/share/info/gcl.info-8.gz
/usr/share/info/gcl.info-9.gz
/usr/share/info/gcl.info.gz
/usr/share/man/man1/gcl.1.gz
```

You can restrict the files listed to just configuration files by adding the `-c` option to your query. Similarly, the `-d` option limits the display to just documentation files.

Querying package files

The above package query commands query the RPM database for installed packages. If you've just downloaded a package and want the same kind of information, you can get this using the `-p` option (for *package file*) on your query along with specifying the package *file* name (as used for installing the package). Listing 13 shows this for the two vim packages that we downloaded earlier. We run it as root only because the files were in root's home directory. You can add other query options, such as `-l` to list files or `-i` to list information.

Listing 13. Displaying package file information for two vim packages

```
[root@echidna ~]# rpm -qp *.rpm
vim-common-7.2.411-1.fc12.x86_64
vim-enhanced-7.2.411-1.fc12.x86_64
```

Querying all installed packages

The `-a` option applies your query to all installed packages. This can generate a lot of output, so you will usually use it in conjunction with one or more filters, such as `sort` to sort the listing, `more` or `less` to page through it, `wc` to obtain package or file counts, or `grep` to search for packages if you aren't sure of the name. Listing 14 shows the following queries:

1. A sorted list of all packages on the system
2. A count of all packages on the system
3. A count of all files in all packages on the system
4. A count of all documentation files installed with RPMs
5. A search for all packages with "gcl" (case-insensitive) as part of their name

Listing 14. Queries against all packages

```
[ian@echidna ~]$ rpm -qa | sort | more
```

```
aalib-libs-1.4.0-0.18.rc5.fc12.x86_64
abrt-1.0.8-2.fc12.x86_64
abrt-addon-ccpp-1.0.8-2.fc12.x86_64
abrt-addon-kerneloops-1.0.8-2.fc12.x86_64
abrt-addon-python-1.0.8-2.fc12.x86_64
abrt-desktop-1.0.8-2.fc12.x86_64
abrt-gui-1.0.8-2.fc12.x86_64
abrt-libs-1.0.8-2.fc12.x86_64
abrt-plugin-bugzilla-1.0.8-2.fc12.x86_64
abrt-plugin-logger-1.0.8-2.fc12.x86_64
abrt-plugin-runapp-1.0.8-2.fc12.x86_64
abyssinica-fonts-1.0-5.fc12.noarch
acl-2.2.49-2.fc12.x86_64
...
[ian@echidna ~]$ rpm -qa | wc -l
1792
[ian@echidna ~]$ rpm -qal | wc -l
281052
[ian@echidna ~]$ rpm -qad | wc -l
45686
[ian@echidna ~]$ rpm -qa | grep -i gcl
gcl-selinux-2.6.8-0.7.20100201cvs.fc12.x86_64
gcl-2.6.8-0.7.20100201cvs.fc12.x86_64
```

Using `rpm -qa` can ease the administration of multiple systems. If you redirect the sorted output to a file on one machine, and then do the same on the other machine, you can use the `diff` program to find differences.

Which package owns a file?

Given that you can list all packages and all files in a package, you now have all the information you need to find which package owns a file. However, the `rpm` command provides a `-f` (or `--file`) option to help you locate the package that owns a file. Suppose you want to know which of the vim packages we saw earlier actually provides the `vim` command. You will need to provide the full path to the file. Listing 15 shows how to use the `which` command to get the full path to the `vim` command, and a handy tip for using this output as input to the `rpm -qf` command. Note that the tick marks surrounding `` `which vim` `` are back-ticks. Another way of using this in the Bash shell is to use `$(which vim)`.

Listing 15. Which package supplies the vim executable

```
[ian@echidna ~]$ which vim
/usr/bin/vim
[ian@echidna ~]$ rpm -qf `which vim`
vim-enhanced-7.2.411-1.fc12.x86_64
[ian@echidna ~]$ rpm -qf $(which vim)
vim-enhanced-7.2.411-1.fc12.x86_64
```

RPM dependencies

You saw earlier that our attempt to erase the gcl-selinux package failed because of *dependencies*. In addition to files, an RPM package may contain arbitrary *capabilities* that other packages may depend on.

As you have seen, this usually works out fine. If you need to install several packages at once, some of which may depend on others, simply use `yum`, or give the whole list to your `rpm -Uvh` command, and it will analyze the dependencies and perform the installs in the right order.

Besides trying to install or erase a package and getting an error message, there are ways to find out what files or capabilities a package requires or depends on.

The `rpm` command provides an option to interrogate installed packages or package files to find out what capabilities they depend on or *require*.This is the `--requires` option, which may be abbreviated to `-R`. Listing 16 shows the capabilities required by gcl. Add the `-p` option and use the full RPM file name if you want to query the package file instead of the RPM database.

Listing 16. What does gcl require

```
[ian@echidna ~]$ rpm -qR gcl
/bin/sh
/bin/sh
/bin/sh
/sbin/install-info
/sbin/install-info
gcl-selinux
libX11.so.6()(64bit)
libc.so.6()(64bit)
libc.so.6(GLIBC_2.11)(64bit)
libc.so.6(GLIBC_2.2.5)(64bit)
libc.so.6(GLIBC_2.3)(64bit)
libc.so.6(GLIBC_2.3.4)(64bit)
libc.so.6(GLIBC_2.4)(64bit)
libc.so.6(GLIBC_2.7)(64bit)
libc.so.6(GLIBC_2.8)(64bit)
libdl.so.2()(64bit)
libgmp.so.3()(64bit)
libm.so.6()(64bit)
libm.so.6(GLIBC_2.2.5)(64bit)
libreadline.so.6()(64bit)
libtcl8.5.so()(64bit)
libtk8.5.so()(64bit)
libz.so.1()(64bit)
rpmlib(CompressedFileNames) <= 3.0.4-1
rpmlib(FileDigests) <= 4.6.0-1
rpmlib(PayloadFilesHavePrefix) <= 4.0-1
rtld(GNU_HASH)
rpmlib(PayloadIsXz) <= 5.2-1
```

It can be somewhat tricky to match capabilities to the packages that provide them. The `yum` command with the `deplist` option can help here. If you just give a package name that is not qualified by version, you may get a listing for other known versions. Listing 17 shows how to get the dependency list for just the version of gcl that is installed.

Listing 17. Using yum deplist to find what gcl requires

```
[ian@echidna ~]$ yum deplist $(rpm -q gcl)
Loaded plugins: presto, refresh-packagekit
Finding dependencies:
package: gcl.x86_64 2.6.8-0.7.20100201cvs.fc12
  dependency: libc.so.6(GLIBC_2.3.4)(64bit)
   provider: glibc.x86_64 2.11-2
   provider: glibc.x86_64 2.11.1-6
  dependency: gcl-selinux
   provider: gcl-selinux.x86_64 2.6.8-0.6.20090701cvs.fc12
   provider: gcl-selinux.x86_64 2.6.8-0.7.20100201cvs.fc12
  dependency: libgmp.so.3()(64bit)
   provider: gmp.x86_64 4.3.1-5.fc12
  dependency: libc.so.6(GLIBC_2.8)(64bit)
   provider: glibc.x86_64 2.11-2
```

```
  provider: glibc.x86_64 2.11.1-6
 dependency: libc.so.6(GLIBC_2.4)(64bit)
  provider: glibc.x86_64 2.11-2
  provider: glibc.x86_64 2.11.1-6
 dependency: libc.so.6()(64bit)
  provider: glibc.x86_64 2.11-2
  provider: glibc.x86_64 2.11.1-6
 dependency: /sbin/install-info
  provider: info.x86_64 4.13a-7.fc12
  provider: info.x86_64 4.13a-9.fc12
 dependency: libX11.so.6()(64bit)
  provider: libX11.x86_64 1.3-1.fc12
 dependency: libc.so.6(GLIBC_2.7)(64bit)
  provider: glibc.x86_64 2.11-2
  provider: glibc.x86_64 2.11.1-6
 dependency: libtcl8.5.so()(64bit)
  provider: tcl.x86_64 1:8.5.7-4.fc12
  provider: tcl.x86_64 1:8.5.7-5.fc12
 dependency: libc.so.6(GLIBC_2.11)(64bit)
  provider: glibc.x86_64 2.11-2
  provider: glibc.x86_64 2.11.1-6
 dependency: libtk8.5.so()(64bit)
  provider: tk.x86_64 1:8.5.7-2.fc12
  provider: tk.x86_64 1:8.5.7-3.fc12
 dependency: libc.so.6(GLIBC_2.3)(64bit)
  provider: glibc.x86_64 2.11-2
  provider: glibc.x86_64 2.11.1-6
 dependency: libm.so.6()(64bit)
  provider: glibc.x86_64 2.11-2
  provider: glibc.x86_64 2.11.1-6
 dependency: libz.so.1()(64bit)
  provider: zlib.x86_64 1.2.3-23.fc12
 dependency: rtld(GNU_HASH)
  provider: glibc.i686 2.11-2
  provider: glibc.x86_64 2.11-2
  provider: glibc.x86_64 2.11.1-6
  provider: glibc.i686 2.11.1-6
 dependency: libdl.so.2()(64bit)
  provider: glibc.x86_64 2.11-2
  provider: glibc.x86_64 2.11.1-6
 dependency: libreadline.so.6()(64bit)
  provider: readline.x86_64 6.0-3.fc12
 dependency: /bin/sh
  provider: bash.x86_64 4.0.33-1.fc12
  provider: bash.x86_64 4.0.35-3.fc12
 dependency: libc.so.6(GLIBC_2.2.5)(64bit)
  provider: glibc.x86_64 2.11-2
  provider: glibc.x86_64 2.11.1-6
 dependency: libm.so.6(GLIBC_2.2.5)(64bit)
  provider: glibc.x86_64 2.11-2
  provider: glibc.x86_64 2.11.1-6
```

This list also shows possible providers for each capability. You can see that most dependencies could be provided by more than one alternative level of a package. For example, /bin/sh could come from either of two levels of bash. With a little creative filtering, you can reduce this output to a list of package names as shown in Listing 18.

Listing 18. Reducing the yum deplist output to just list package names

```
[ian@echidna ~]$ yum deplist $(rpm -q gcl) | awk '/provider:/ { print $2 }'|sort|uniq
bash.x86_64
gcl-selinux.x86_64
glibc.i686
glibc.x86_64
gmp.x86_64
info.x86_64
libX11.x86_64
readline.x86_64
tcl.x86_64
tk.x86_64
zlib.x86_64
```

If you just need to know what packages need to be installed, you can always run `yum install` and see the list before you are prompted to accept the installation proposal.

In addition to finding out what capabilities a package requires, you may need to find what package provides some capability. You saw above how to find which package owns a file. Listing 19 shows how to use `rpm` or `yum` to find what package provides the gcl-selinux(x86-64) capability. In addition to information about installed packages providing the capability, YUM also shows the packages or versions available in repositories. These are the original 2.6.8-0.6 version from the fedora repository and the updated 2.6.8-0.7 version available from the updates repository.

Listing 19. What packages provide gcl-selinux(x86-64) capability

```
[ian@echidna ~]$ rpm -q --whatprovides 'gcl-selinux(x86-64)'
gcl-selinux-2.6.8-0.7.20100201cvs.fc12.x86_64
[ian@echidna ~]$ yum whatprovides 'gcl-selinux(x86-64)'
Loaded plugins: presto, refresh-packagekit
gcl-selinux-2.6.8-0.6.20090701cvs.fc12.x86_64 : SELinux policy for GCL images
Repo        : fedora
Matched from:
Other       : gcl-selinux(x86-64)



gcl-selinux-2.6.8-0.7.20100201cvs.fc12.x86_64 : SELinux policy for GCL images
Repo        : updates
Matched from:
Other       : gcl-selinux(x86-64)



gcl-selinux-2.6.8-0.7.20100201cvs.fc12.x86_64 : SELinux policy for GCL images
Repo        : installed
Matched from:
Other       : Provides-match: gcl-selinux(x86-64)
```

RPM package file integrity

To ensure their integrity, RPM packages include a digest, such as MD5 or SHA1, and are usually digitally signed. Packages that are digitally signed need a public key for verification. To check the integrity of an RPM package file, use the `--checksig` (abbreviated to `-K`) option of `rpm`. You will usually find it useful to add the `-v` option for more verbose output. Listing 20 shows an example for the vim-enhanced RPM.

Listing 20. Checking the integrity of the vim-enhanced package file

```
[root@echidna ~]# rpm -vK vim-enhanced-7.2.411-1.fc12.x86_64.rpm
vim-enhanced-7.2.411-1.fc12.x86_64.rpm:
    Header V3 RSA/SHA256 signature: OK, key ID 57bbccba
    Header SHA1 digest: OK (f9a199545a515f7ff0716729768b41eb68fe29a8)
    V3 RSA/SHA256 signature: OK, key ID 57bbccba
    MD5 digest: OK (d4045f1f72d48073e3f401ee9d1f71cf)
```

You may get an output line like:

```
V3 DSA signature: NOKEY, key ID 16a61572
```

This means that you have a signed package, but you do not have the needed public key in your RPM database. Note that earlier versions of RPM may present the verification differently.

If a package is signed and you want to verify it against a signature, then you will need to locate the appropriate signature file and import it into your RPM database. You should first download the key and then check its fingerprint before importing it using the `rpm --import` command. For more information, see the RPM man pages. You will also find more information on signed binaries at the RPM home page (see Resources for a link).

Verifying an installed package

Like checking the integrity of an rpm, you can also check the integrity of your installed files using `rpm -V`. This step makes sure that the files haven't been modified since they were installed from the rpm. As shown in Listing 21, there is no output from this command if the package is still good, but you can add the `-v` option to get much more detailed output.

Listing 21. Verifying the installed vim-common package

```
[ian@echidna ~]$ rpm -V vim-common
```

Let's become root and corrupt our vim-common installation by deleting /usr/bin/xxd and replacing /usr/share/vim/vim72/syntax/bindzone.vim with /bin/bash. Let's try the verification again. The results are shown in Listing 22.

Listing 22. Tampering with the vim-common package

```
[root@echidna ~]# rpm -qf /usr/bin/xxd /usr/share/vim/vim72/syntax/bindzone.vim
vim-common-7.2.411-1.fc12.x86_64
vim-common-7.2.411-1.fc12.x86_64
[root@echidna ~]# rm /usr/bin/xxd
rm: remove regular file `/usr/bin/xxd'? y
[root@echidna ~]# cp /bin/bash /usr/share/vim/vim72/syntax/bindzone.vim
cp: overwrite `/usr/share/vim/vim72/syntax/bindzone.vim'? y
[root@echidna ~]# rpm -V vim-common
missing     /usr/bin/xxd
S.5....T.    /usr/share/vim/vim72/syntax/bindzone.vim
```

This output shows us that the /usr/share/vim/vim72/syntax/bindzone.vim file fails MD5 sum, file size, and mtime tests. One way to solve the problem would be to remove the package and then

reinstall it, but there are other packages that depend on vim-common and that are installed and still OK. The solution is to forcibly reinstall it using the `--force` option of `rpm`, or the `reinstall` function of `yum`. Listing 23 shows how to reinstall with `yum`, and then verify that the package is now OK and the deleted file has been restored.

Listing 23. Reinstalling the vim-common package

```
[root@echidna ~]# yum reinstall vim-common
Loaded plugins: presto, refresh-packagekit
Setting up Reinstall Process
Resolving Dependencies
--> Running transaction check
---> Package vim-common.x86_64 2:7.2.411-1.fc12 set to be updated
--> Finished Dependency Resolution

Dependencies Resolved

================================================================================
 Package            Arch           Version                     Repository       Size
================================================================================
Reinstalling:
 vim-common         x86_64         2:7.2.411-1.fc12            updates         6.0 M

Transaction Summary
================================================================================
Remove        0 Package(s)
Reinstall     1 Package(s)
Downgrade     0 Package(s)

Total download size: 6.0 M
Installed size: 17 M
Is this ok [y/N]: y
Downloading Packages:
Setting up and reading Presto delta metadata
updates/prestodelta                                      | 969 kB     00:00
Processing delta metadata
Package(s) data still to download: 6.0 M
vim-common-7.2.411-1.fc12.x86_64.rpm                     | 6.0 MB     00:01
Running rpm_check_debug
Running Transaction Test
Transaction Test Succeeded
Running Transaction
Warning: RPMDB altered outside of yum.
  Installing     : 2:vim-common-7.2.411-1.fc12.x86_64                       1/1

Installed:
  vim-common.x86_64 2:7.2.411-1.fc12

Complete!
[root@echidna ~]# rpm -V vim-common
[root@echidna ~]# ls /usr/bin/xxd
/usr/bin/xxd
```

If you need more force

Usually the package management system keeps your packages in order. However, if you manage to delete some file that is an important part of a package—and reinstalling the package without removing does not fix the problem—then you may need to remove the package before reinstalling. For such a case, you probably want to delete the existing copy and reinstall it, without needing to uninstall and reinstall all the packages that depend on it. For this, you can use the `rpm` command's `--nodeps` option to bypass dependency checking when you remove a package. Listing 24 shows

how this might work if you accidentally removed the /usr/bin/xxd file, which is part of the vim-common package, as we did earlier.

Listing 24. Updating packages with rpm

```
[root@echidna ~]# rm /usr/bin/xxd
rm: remove regular file `/usr/bin/xxd'? y
[root@echidna ~]# # Oops! we needed that file
[root@echidna ~]# rpm -Fvh vim-common-7.2.411-1.fc12.x86_64.rpm
[root@echidna ~]# ls /usr/bin/xxd
ls: cannot access /usr/bin/xxd: No such file or directory
[root@echidna ~]# # Oh! Freshening the package didn't replace the missing file
[root@echidna ~]# rpm -e vim-common
error: Failed dependencies:
 vim-common = 2:7.2.411-1.fc12 is needed by (installed) vim-enhanced-2:7.2.411-1.f
c12.x86_64
[root@echidna ~]# # Can't remove vim-common because vim-enhanced needs it
[root@echidna ~]# rpm -e --nodeps vim-common
[root@echidna ~]# # Bypassing the dependency check allowed removal
[root@echidna ~]# rpm -Uvh vim-common-7.2.411-1.fc12.x86_64.rpm
Preparing...                ########################################### [100%]
   1:vim-common             ########################################### [100%]
[root@echidna ~]# # Update (or install) vim-common again
[root@echidna ~]# ls /usr/bin/xxd
/usr/bin/xxd
[root@echidna ~]# # And /usr/bin/xxd is back
```

So now you have some approaches to updating or repairing if accidents happen and the ordinary update process fails. Note that you can also bypass dependency checking when installing an RPM, but this not usually a good idea.

Downloading RPMs from repositories

Although `yum` will automatically retrieve packages from repositories, you may want to download RPMs and save them, perhaps to install them on a non-networked system, or to examine their contents, or for some other reason. You can use the `yumdownloader` command as shown in Listing 25. In our case, the package is already installed, so there are no additional packages to download. If there were such packages, the `--resolve` option would cause them to be downloaded too.

Listing 25. Downloading the gcl package

```
[ian@echidna ~]$ yumdownloader --resolve gcl
Loaded plugins: presto, refresh-packagekit
adobe-linux-i386                                                   17/17
--> Running transaction check
---> Package gcl.x86_64 0:2.6.8-0.7.20100201cvs.fc12 set to be updated
--> Finished Dependency Resolution
gcl-2.6.8-0.7.20100201cvs.fc12.x86_64.rpm                    | 6.3 MB     00:01
```

Using rpm2cpio

If you download an RPM and need to examine its contents, rather than install it, you can use the `rpm2cpio`command to convert the contents to a `cpio` archive and then filter that through the `cpio` command to extract individual files or all the files in the package. Listing 26 shows how to do this for the gcl-selinux package and then shows what files (and directories) were unpacked. See the man pages for `rpm2cpio` and `cpio` for additional details on these commands.

Listing 26. Unpacking the gcl-selinux package with rpm2cpio

```
[ian@echidna ~]$ yumdownloader gcl-selinux
Loaded plugins: presto, refresh-packagekit
gcl-selinux-2.6.8-0.7.20100201cvs.fc12.x86_64.rpm        |  17 kB     00:00
[ian@echidna ~]$ mkdir gcl-selinux
[ian@echidna ~]$ cd gcl-selinux
[ian@echidna gcl-selinux]$ rpm2cpio ../gcl-selinux*.rpm | cpio -idv
./usr/share/selinux/packages/gcl
./usr/share/selinux/packages/gcl/gcl.pp
182 blocks
[ian@echidna gcl-selinux]$ find .
.
./usr
./usr/share
./usr/share/selinux
./usr/share/selinux/packages
./usr/share/selinux/packages/gcl
./usr/share/selinux/packages/gcl/gcl.pp
```

Finding RPMs

We saw earlier that YUM offers a search capability, which searches descriptions as well as package names. If you need to find what package contains a program that you do not have installed, there are a few other ways:

- You can guess what package might contain it and download the package without installing. Once you have the package, you can interrogate it.
- You can search the Internet.
- You may be able to use the command-not-found capability described below.

If you can't find a particular RPM through your system tools, a good Internet resource for locating RPMs is the Rpmfind.Net server (see Resources for a link).

Command not found

When the Bash shell searches for a command and does not find it, then the shell searches for a shell function named `command_not_found_handle`. If the `command_not_found_handle` function exists, it is invoked with the original command and original arguments as its arguments, and the function's exit status becomes the exit status of the shell. If the function is not defined, the shell prints an error message and returns an exit status of 127. The function is usually set in the system /etc/bash.bashrc file. Listing 27 shows how we searched for the command-not-found capability and then installed it.

Listing 27. Locating and installing the command-not-found capability

```
[root@echidna ~]# yum search command-not-found
Loaded plugins: presto, refresh-packagekit
========================== Matched: command-not-found ==========================
PackageKit-command-not-found.x86_64 : Ask the user to install command line
                                    : programs automatically
You have new mail in /var/spool/mail/root
[root@echidna ~]# yum install PackageKit-command-not-found.x86_64
Loaded plugins: presto, refresh-packagekit
```

```
Setting up Install Process
Resolving Dependencies
--> Running transaction check
---> Package PackageKit-command-not-found.x86_64 0:0.5.7-2.fc12 set to be updated
--> Finished Dependency Resolution

Dependencies Resolved

================================================================================
 Package                        Arch       Version          Repository   Size
================================================================================
Installing:
 PackageKit-command-not-found     x86_64     0.5.7-2.fc12     updates     102 k

Transaction Summary
================================================================================
Install       1 Package(s)
Upgrade       0 Package(s)

Total download size: 102 k
Installed size: 262 k
Is this ok [y/N]: y
Downloading Packages:
Setting up and reading Presto delta metadata
Processing delta metadata
Package(s) data still to download: 102 k
PackageKit-command-not-found-0.5.7-2.fc12.x86_64.rpm     | 102 kB     00:00
Running rpm_check_debug
Running Transaction Test
Transaction Test Succeeded
Running Transaction
  Installing     : PackageKit-command-not-found-0.5.7-2.fc12.x86_64         1/1

Installed:
  PackageKit-command-not-found.x86_64 0:0.5.7-2.fc12

Complete!
```

Listing 28 shows how the function handle is defined after installing PackageKit-command-not-found. If the function cannot perform the search, then it mimics the standard system behavior and returns 127.

Listing 28. The command_not_found_handle

```
[ian@echidna ~]$ type command_not_found_handle
command_not_found_handle is a function
command_not_found_handle ()
{
    runcnf=1;
    retval=127;
    [ ! -S /var/run/dbus/system_bus_socket ] && runcnf=0;
    [ ! -x /usr/sbin/packagekitd ] && runcnf=0;
    if [ $runcnf -eq 1 ]; then
        /usr/libexec/pk-command-not-found $1;
        retval=$?;
    else
        echo "bash: $1: command not found";
    fi;
    return $retval
}
```

If this had been installed before we running gcl as we did back in Listing 1, you might have seen something like Listing 29.

Listing 29. Attempting gcl with a command_not_found_handle

```
[ian@echidna ~]$ gcl
Command not found. Install package 'gcl' to provide command 'gcl'? [N/y]
```

Other tools

In addition to `yum` and `rpm`, your distributor may provide other tools for installing packages from the repository or updating your entire system. These tools may be graphical or command line or both. Some examples include:

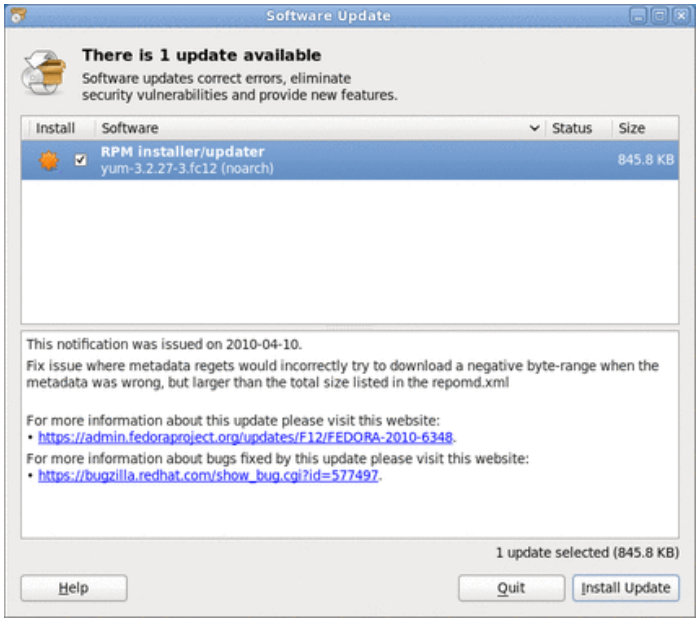
- YaST (SUSE)
- up2date (Red Hat)
- Mandrake Software Management (Mandriva)

Usually these tools will handle multiple package updates in an automatic or semi-automatic fashion. They may also provide capabilities to display contents of repositories or search for packages. Consult the documentation for your distribution for more details.

PackageKit

No discussion of package installation would be complete without mentioning PackageKit, which is a system designed to make installing and updating software easier. The intent is to unify all the software graphical tools used in different distributions. PackageKit uses a system activated daemon, which means that the daemon is activated only when needed. PackageKit has version for Gnome (gnome-packagekit) and KDE KPackageKit). The command-not-found handle described above is also part of PackageKit. It includes the commands `pkcon` to perform package management functions from the console, and `pkmon` to monitor package kit activity. It also includes graphical tools for adding software packages, or for updating your system. Figure 1 shows an example of the Software Update graphical interface.

Figure 1. Software Update graphical interface on Fedora 12 (Gnome)



There is a lot more to the RPM and YUM package management systems than covered here. See Resources for additional links.

Resources

Learn

- Use the developerWorks roadmap for LPIC-1 to find the developerWorks articles to help you study for LPIC-1 certification based on the April 2009 objectives.
- At the LPIC Program site, find detailed objectives, task lists, and sample questions for the three levels of the Linux Professional Institute's Linux system administration certification. In particular, see their April 2009 objectives for LPI exam 101 and LPI exam 102. Always refer to the LPIC Program site for the latest objectives.
- Review the entire LPI exam prep series on developerWorks to learn Linux fundamentals and prepare for system administrator certification based on earlier LPI exam objectives prior to April 2009.
- At the RPM home page find current information on the RPM software packaging tool and pointers to more information on RPM.
- The book *Maximum RPM* offers a comprehensive and systematic treatment of all aspects of RPM. It's available in both hard and soft copy formats.
- At the LSB Home, learn about the Linux Standard Base (LSB), a Free Standards Group (FSG) project to develop a standard binary operating environment.
- See the PackageKit home page to learn more about PackageKit.
- The Linux Documentation Project has a variety of useful documents, especially its HOWTOs.
- In the developerWorks Linux zone, find hundreds of how-to articles and tutorials, as well as downloads, discussion forums, and a wealth other resources for Linux developers and administrators.
- Stay current with developerWorks technical events and webcasts focused on a variety of IBM products and IT industry topics.
- Attend a free developerWorks Live! briefing to get up-to-speed quickly on IBM products and tools as well as IT industry trends.
- Watch developerWorks on-demand demos ranging from product installation and setup demos for beginners, to advanced functionality for experienced developers.
- Follow developerWorks on Twitter, or subscribe to a feed of Linux tweets on developerWorks.

Get products and technologies

- Find packages on one of your distribution's mirrors, such as Fedora/12 Public Active Mirrors.
- Search for RPMs for your distribution at Rpmfind.Net and RPM Search.
- Evaluate IBM products in the way that suits you best: Download a product trial, try a product online, use a product in a cloud environment, or spend a few hours in the SOA Sandbox learning how to implement Service Oriented Architecture efficiently.

Discuss

- Participate in the discussion forum for this content.
- Get involved in the My developerWorks community. Connect with other developerWorks users while exploring the developer-driven blogs, forums, groups, and wikis.

About the author

Ian Shields

Ian Shields works on a multitude of Linux projects for the developerWorks Linux zone. He is a Senior Programmer at IBM at the Research Triangle Park, NC. He joined IBM in Canberra, Australia, as a Systems Engineer in 1973, and has since worked on communications systems and pervasive computing in Montreal, Canada, and RTP, NC. He has several patents and has published several papers. His undergraduate degree is in pure mathematics and philosophy from the Australian National University. He has an M.S. and Ph.D. in computer science from North Carolina State University. Learn more about Ian in Ian's profile on developerWorks Community.