RPM

The *RPM Package Manager* (**RPM**) is an open packaging system that runs on Red Hat Enterprise Linux as well as other Linux and UNIX systems. Red Hat and the Fedora Project encourage other vendors to use **RPM** for their own products. **RPM** is distributed under the terms of the *GPL* (*GNU General Public License*).

The **RPM Package Manager** only works with packages built in the *RPM format*. **RPM** itself is provided as the pre-installed *rpm* package. For the end user, **RPM** makes system updates easy. Installing, uninstalling, and upgrading **RPM** packages can be accomplished with short commands. **RPM** maintains a database of installed packages and their files, so you can invoke powerful queries and verifications on your system. There are several applications, such as **Yum** or **PackageKit**, that can make working with packages in the **RPM** format even easier.

Warning

For most package-management tasks, the **Yum** package manager offers equal and often greater capabilities and utility than **RPM**. **Yum** also performs and tracks complicated system-dependency resolutions. **Yum** maintains the system integrity and forces a system integrity check if packages are installed or removed using another application, such as **RPM**, instead of **Yum**. For these reasons, it is highly recommended that you use **Yum** instead of **RPM** whenever possible to perform package-management tasks. See Chapter 7, *Yum*.

If you prefer a graphical interface, you can use the **PackageKit** GUI application, which uses **Yum** as its back end, to manage your system's packages.

During upgrades, **RPM** handles configuration files carefully, so that you never lose your customizations — something that you cannot accomplish with regular `.tar.gz` files.

For the developer, **RPM** enables software source code to be packaged into source and binary packages for end users. This process is quite simple and is driven from a single file and optional patches that you create. This clear delineation between pristine sources and your patches along with build instructions eases the maintenance of the package as new versions of the software are released.

Note

Because **RPM** can make changes to the system itself, performing operations like installing, upgrading, downgrading, and uninstalling binary packages system-wide requires `root` privileges in most cases.

A.1. RPM Design Goals

To understand how to use **RPM**, it is helpful to understand the design goals of **RPM**:

Upgradability

With **RPM**, you can upgrade individual components of your system without a complete reinstallation. When you get a new release of an operating system based on **RPM**, such as Red Hat Enterprise Linux, you do not need to reinstall a fresh copy of the operating system

on your machine (as you might need to with operating systems based on other packaging systems). **RPM** allows for intelligent, fully-automated, in-place upgrades of your system. In addition, configuration files in packages are preserved across upgrades, so you do not lose your customizations. There are no special upgrade files needed to upgrade a package because the same **RPM** file is used to both install and upgrade the package on the system.

Powerful Querying

RPM is designed to provide powerful querying options. You can perform searches on your copy of the database for packages or even just certain files. You can also easily find out what package a file belongs to and where the package came from. The files an **RPM** package contains are in a compressed archive, with a custom binary header containing useful information about the package and its contents, allowing you to query individual packages quickly and easily.

System Verification

Another powerful **RPM** feature is the ability to verify packages. It allows you to verify that the files installed on the system are the same as the ones supplied by a given package. If an inconsistency is detected, **RPM** notifies you, and you can reinstall the package if necessary. Any configuration files that you modified are preserved during reinstallation.

Pristine Sources

A crucial design goal was to allow the use of *pristine* software sources, as distributed by the original authors of the software. With **RPM**, you have the pristine sources along with any patches that were used, plus complete build instructions. This is an important advantage for several reasons. For example, if a new version of a program is released, you do not necessarily have to start from scratch to get it to compile. You can look at the patch to see what you *might* need to do. All the compiled-in defaults, and all of the changes that were made to get the software to build properly, are easily visible using this technique.

The goal of keeping sources pristine may seem important only for developers, but it results in higher quality software for end users.

A.2. Using RPM

RPM has five basic modes of operation (excluding package building): installing, uninstalling, upgrading, querying, and verifying. This section contains an overview of each mode. For complete details and options, try **`rpm --help`** or see rpm(8). Also, see Section A.5, “Additional Resources” for more information on **RPM**.

A.2.1. Installing and Upgrading Packages

RPM packages typically have file names in the following form:

```
package_name-version-release-operating_system-CPU_architecture.rpm
```

For example the **`tree-1.6.0-10.el7.x86_64.rpm`** file name includes the package name (**`tree`**), version (**`1.6.0`**), release (**`10`**), operating system major version (**`el7`**) and CPU architecture (**`x86_64`**).

Important

When installing a package, ensure it is compatible with your operating system and processor architecture. This can usually be determined by checking the package name. For example, the file name of an **RPM** package compiled for the AMD64/Intel 64 computer architectures ends with **x86_64.rpm**.

The **-U** (or **--upgrade**) option has two functions, it can be used to:

- upgrade an existing package on the system to a newer version, or
- install a package if an older version is not already installed.

The **rpm -U *package.rpm*** command is therefore able to either *upgrade* or *install*, depending on the presence of an older version of *package.rpm* on the system.

Assuming the **tree-1.6.0-10.el7.x86_64.rpm** package is in the current directory, log in as **root** and type the following command at a shell prompt to either upgrade or install the *tree* package:

```
~]# rpm -Uvh tree-1.6.0-10.el7.x86_64.rpm
```

The **-v** and **-h** options (which are combined with **-U**) cause **rpm** to print a more verbose output and display a progress meter using hash signs. If the upgrade or installation is successful, the following output is displayed:

```
Preparing...                ########################################### [100%]
Updating / installing...
   1:tree-1.6.0-10.el7      ########################################### [100%]
```

Warning

rpm provides two different options for installing packages: the aforementioned **-U** option (which historically stands for *upgrade*), and the **-i** option (which historically stands for *install*). Because the **-U** option includes both install and upgrade functions, the use of **rpm -Uvh** with all packages, **except** *kernel* packages, is recommended.

You should always use the **-i** option to *install* a new kernel package instead of upgrading it. This is because using the **-U** option to upgrade a kernel package removes the previous (older) kernel package, which could render the system unable to boot if there is a problem with the new kernel. Therefore, use the **rpm -i *kernel_package*** command to install a new kernel *without replacing any older kernel packages*. For more information on installing *kernel* packages, see Chapter 25, *Manually Upgrading the Kernel*.

The signature of a package is checked automatically when installing or upgrading a package. The signature confirms that the package was signed by an authorized party. If the verification of the signature fails, an error message is displayed.

If you do not have the appropriate key installed to verify the signature, the message contains the word **NOKEY**:

```
warning: tree-1.6.0-10.el7.x86_64.rpm: Header V3 RSA/SHA256 Signature,
key ID 431d51: NOKEY
```

See Section A.3.2, “Checking Package Signatures” for more information on checking package signatures.

A.2.1.1. Replacing Already-Installed Packages

If a package of the same name and version is already installed, the following output is displayed:

```
Preparing...                ########################################### 
[100%]
        package tree-1.6.0-10.el7.x86_64 is already installed
```

To install the package anyway, use the **--replacepkgs** option, which tells **RPM** to ignore the error:

```
~]# rpm -Uvh --replacepkgs tree-1.6.0-10.el7.x86_64.rpm
```

This option is helpful if files installed from the package were deleted or if you want the original configuration files to be installed.

If you attempt an upgrade to an *older* version of a package (that is, if a newer version of the package is already installed), **RPM** informs you that a newer version is already installed. To force **RPM** to perform the downgrade, use the **--oldpackage** option:

```
rpm -Uvh --oldpackage older_package.rpm
```

A.2.1.2. Resolving File Conflicts

If you attempt to install a package that contains a file that has already been installed by another package, a conflict message is displayed. To make **RPM** ignore this error, use the **--replacefiles** option:

```
rpm -Uvh --replacefiles package.rpm
```

A.2.1.3. Satisfying Unresolved Dependencies

RPM packages sometimes depend on other packages, which means that they require other packages to be installed to run properly. If you try to install a package that has an unresolved dependency, a message about a failed dependency is displayed.

Find the suggested package(s) on the Red Hat Enterprise Linux installation media or on one of the active Red Hat Enterprise Linux mirrors and add it to the installation command. To determine which package contains the required file, use the **--whatprovides** option:

```
rpm -q --whatprovides "required_file"
```

If the package that contains *required_file* is in the **RPM** database, the name of the package is displayed.

Warning

Although you can *force* **rpm** to install a package that has an unresolved dependency (using the **--nodeps** option), this is *not* recommended and will usually result in the installed software failing to run. Installing packages with **--nodeps** can cause applications to misbehave or terminate unexpectedly. It can also cause serious package-management problems or system failure. For these reasons, heed the warnings about missing dependencies. The **Yum** package manager performs automatic dependency resolution and fetches dependencies from on-line repositories.

A.2.1.4. Preserving Changes in Configuration Files

Because **RPM** performs intelligent upgrading of packages with configuration files, you may see the following message:

```
saving /etc/configuration_file.conf as
/etc/configuration_file.conf.rpmsave
```

This message means that the changes you made to the configuration file may not be *forward-compatible* with the new configuration file in the package, so **RPM** saved your original file and installed a new one. You should investigate the differences between the two configuration files and resolve them as soon as possible to ensure that your system continues to function properly.

Alternatively, **RPM** may save the package's *new* configuration file as, for example, ***configuration_file.conf*.rpmnew** and leave the configuration file you modified untouched. You should still resolve any conflicts between your modified configuration file and the new one, usually by merging changes from the old one to the new one, for example using the **diff** program.

A.2.2. Uninstalling Packages

Uninstalling a package is just as simple as installing one. Type the following command at a shell prompt as **root**:

```
rpm -e package
```

Note

Note that the command expects only the package *name*, not the name of the original package *file*. If you attempt to uninstall a package using the **rpm -e** command and provide the original full file name, you receive a package-name error.

You can encounter dependency errors when uninstalling a package if another installed package depends on the one you are trying to remove. For example:

```
~]# rpm -e ghostscript
error: Failed dependencies:
        ghostscript is needed by (installed) ghostscript-cups-9.07-
16.el7.x86_64
        ghostscript is needed by (installed) foomatic-4.0.9-6.el7.x86_64
        libgs.so.9()(64bit) is needed by (installed) libspectre-0.2.7-
```

```
4.el7.x86_64
        libijs-0.35.so()(64bit) is needed by (installed) gutenprint-
5.2.9-15.el7.x86_64
        libijs-0.35.so()(64bit) is needed by (installed) cups-filters-
1.0.35-15.el7.x86_64
```

Warning

Although you can *force* **rpm** to uninstall a package that has unresolved dependencies (using the **--nodeps** option), this is *not* recommended. Removing packages with **--nodeps** can cause applications from the packages whose dependencies are removed to misbehave or terminate unexpectedly. It can also cause serious package-management problems or system failure. For these reasons, heed the warnings about failed dependencies.

A.2.3. Freshening Packages

Freshening is similar to upgrading, except that only installed packages are upgraded. Type the following command at a shell prompt as **root**:

```
rpm -Fvh package.rpm
```

The **-F** (or **--freshen**) option compares the versions of the packages specified on the command line with the versions of packages that are already installed on the system. When a newer version of an already-installed package is processed by the **--freshen** option, it is upgraded to the newer version. However, the **--freshen** option does not install a package if no previously-installed package of the same name exists. This differs from regular upgrading, as an upgrade installs all specified packages regardless of whether or not older versions of the packages are already installed.

Freshening works for single packages or package groups. For example, freshening can help if you download a large number of different packages, and you only want to upgrade those packages that are already installed on the system. In this case, issue the following command with the ***.rpm** global expression:

```
~]# rpm -Fvh *.rpm
```

RPM then automatically upgrades only those packages that are already installed.

A.2.4. Querying Packages

The **RPM** database stores information about all **RPM** packages installed on the system. It is stored in the **/var/lib/rpm/** directory and is used for many things, including querying what packages are installed, what version each package is, and for calculating changes to files in packages since their installation. To query this database, use the **rpm** command with the **-q** (or **--query**) option:

```
rpm -q package_name
```

This command displays the package name, version, and release number of the installed package *package_name*. For example:

```
~]$ rpm -q tree
tree-1.6.0-10.el7.x86_64
```

See the **Package Selection Options** subheading in the rpm(8) manual page for a list of options that can be used to further refine or qualify your query. Use options listed below the **Package Query Options** subheading to specify what information to display about the queried packages.

A.2.5. Verifying Packages

Verifying a package is comparing information about files on the system installed from a package with the same information from the original package. Among other parameters, verifying compares the file size, MD5 sum, permissions, type, owner, and the group of each file.

Use the **rpm** command with the **-V** (or **--verify**) option to verify packages. For example:

```
~]$ rpm -V tree
```

See the **Package Selection Options** subheading in the rpm(8) manual page for a list of options that can be used to further refine or qualify your query. Use options listed below the **Verify Options** subheading to specify what characteristics to verify in the queried packages.

If everything verifies properly, there is no output. If there are any discrepancies, they are displayed. The output consists of lines similar to these:

```
~]# rpm -V abrt
S.5....T.  c /etc/abrt/abrt.conf
.M.......    /var/spool/abrt-upload
```

The format of the output is a string of nine characters followed by an optional attribute marker and the name of the processed file.

The first nine characters are the results of tests performed on the file. Each test is the comparison of one attribute of the file to the value of that attribute as recorded in the **RPM** database. A single period (**.**) means the test passed, and the question-mark character (**?**) signifies that the test could not be performed. The following table lists symbols that denote specific discrepancies:

Table A.1. RPM Verification Symbols

Symbol	Description
S	file size differs
M	mode differs (includes permissions and file type)
5	digest (formerly MD5 sum) differs
D	device major/minor number mismatch
L	readLink(2) path mismatch
U	user ownership differs
G	group ownership differs
T	mtime differs
P	capabilities differ

The attribute marker, if present, describes the purpose of the given file. The following table lists the available attribute markers:

Table A.2. RPM Verification Symbols

Marker	Description
c	configuration file
d	documentation file
l	license file
r	readme file

If you see any output, use your best judgment to determine if you should remove the package, reinstall it, or fix the problem in another way.

A.3. Finding and Verifying RPM Packages

Before using any **RPM** packages, you must know where to find them and be able to verify if you can trust them.

A.3.1. Finding RPM Packages

Although there are many **RPM** repositories on the Internet, for security and compatibility reasons, you should consider installing only official Red Hat-provided RPM packages. The following is a list of sources for **RPM** packages:

- Official Red Hat Enterprise Linux installation media.
- Official **RPM** repositories provided with the **Yum** package manager. See Chapter 7, *Yum* for details on how to use the official Red Hat Enterprise Linux package repositories.
- The Red Hat Errata Page, available on the Customer Portal at https://rhn.redhat.com/rhn/errata/RelevantErrata.do.
- Extra Packages for Enterprise Linux (EPEL) is a community effort to provide a repository with high-quality add-on packages for Red Hat Enterprise Linux. See http://fedoraproject.org/wiki/EPEL for details on EPEL **RPM** packages.
- Unofficial, third-party repositories not affiliated with Red Hat also provide RPM packages.

Important

When considering third-party repositories for use with your Red Hat Enterprise Linux system, pay close attention to the repository's web site with regard to package compatibility before adding the repository as a package source. Alternate package repositories may offer different, incompatible versions of the same software, including packages already included in the Red Hat Enterprise Linux repositories.

A.3.2. Checking Package Signatures

RPM packages can be signed using **GNU Privacy Guard** (or **GPG**), which helps you make certain that downloaded packages are trustworthy. **GPG** is a tool for secure communication. With **GPG**, you can authenticate the validity of documents and encrypt or decrypt data.

To verify that a package has not been corrupted or tampered with, check its **GPG** signature by using the **rpmkeys** command with the **-K** (or **--checksig**) option:

```
rpmkeys -K package.rpm
```

Note that the **Yum** package manager performs automatic checking of **GPG** signatures during installations and upgrades.

GPG is installed by default, as well as a set of Red Hat keys for verifying packages. To import additional keys for use with **RPM**, see Section A.3.2.1, “Importing GPG Keys”.

A.3.2.1. Importing GPG Keys

To verify Red Hat packages, a Red Hat **GPG** key needs to be installed. A set of basic keys is installed by default. To view a list of installed keys, execute the following command at a shell prompt:

```
~]$ rpm -qa gpg-pubkey*
```

To display details about a specific key, use **rpm -qi** followed by the output from the previous command. For example:

```
~]$ rpm -qi gpg-pubkey-fd431d51-4ae0493b
```

Use the **rpmkeys** command with the **--import** option to install a new key for use with **RPM**. The default location for storing **RPM** GPG keys is the **/etc/pki/rpm-gpg/** directory. To import new keys, use a command like the following as **root**:

```
~]# rpmkeys --import /etc/pki/rpm-gpg/RPM-GPG-KEY-redhat-release
```

See the Product Signing (GPG) Keys article on the Red Hat Customer Portal for additional information about Red Hat package-signing practices.

A.4. Common Examples of RPM Usage

RPM is a useful tool for both managing your system and diagnosing and fixing problems. See the following examples for an overview of some of the most-used options.

- To verify your entire system and see what files are missing, issue the following command as **root**:

  ```
  rpm -Va
  ```

  If some files are missing or appear corrupted, consider reinstalling relevant packages.

- To determine which package owns a file, enter:

  ```
  rpm -qf file
  ```

- To verify the package that owns a particular file, enter as **root**:

  ```
  rpm -Vf file
  ```

- To locate documentation files that are a part of a package to which a file belongs, enter:

  ```
  rpm -qdf file
  ```

- To find information about a (non-installed) package file, use the following command:

```
rpm -qip package.rpm
```

- To list files contained in a package, use:

```
rpm -qlp package.rpm
```

See the rpm(8) manual page for more options.

A.5. Additional Resources

RPM is a complex utility with many options and methods for querying, installing, upgrading, and removing packages. See the following resources to learn more about **RPM**.

Installed Documentation

- **rpm --help** — This command displays a quick reference of **RPM** parameters.
- rpm(8) — The **RPM** manual page offers an overview of all available **RPM** parameters.

Online Documentation

- Red Hat Enterprise Linux 7 Security Guide — The *Security Guide* for Red Hat Enterprise Linux 7 documents how to keep your system up-to-date using the **Yum** package manager and how to verify and install downloaded packages.
- The **RPM** website — http://www.rpm.org/
- The **RPM** mailing list — http://lists.rpm.org/mailman/listinfo/rpm-list

See Also

- Chapter 7, *Yum* describes how to use the **Yum** package manager to search, install, update, and uninstall packages on the command line.