Packaging software with RPM, Part 1: Building and distributing packages

Martin Streicher
Software Developer
Pixel, Byte, and Comma

12 January 2010

In this first article in a three-part series on the RPM Package Manager, learn how to use RPM not just to install software and attendant files but to package almost anything, from system scripts to source code to documentation. (This series replaces an earlier series on RPM written by Dan Poirier.)
Share your expertise: What are your favorite RPM tricks? Add your comments below.

View more content in this series

The principal benefit of open source software is, as its name implies, access to the inner workings of an application. Given the source, you can study how an application works; change, improve, and extend its operation; borrow and repurpose code (per the limits of the application's license); and port the application to novel and emergent platforms.

However, such liberal access is not always wanted. For instance, a user may not want the onus of building from source code. Instead, he or she may simply want to install the software much like a traditional "shrink-wrapped" application: insert media, run setup, answer a few prompts, and go. Indeed, for most computer users, such pre-built software is preferred. Pre-built code is less sensitive to system vagaries and thus more uniform and predictable.

In general, a pre-built, open source application is called a *package* and bundles all the binary, data, and configuration files required to run the application. A package also includes all the steps required to deploy the application on a system, typically in the form of a script. The script might generate data, start and stop system services, or manipulate files and directories. A script might also perform operations to upgrade existing software to a new version.

Because each operating system has its idiosyncrasies, a package is typically tailored to a specific system. Moreover, each operating system provides its own *package manager,* a special utility to add and remove packages from the system. For example, Debian Linux®-based systems use the Advanced Package Tool (APT), while Fedora Linux systems use the RPM Package Manager.

© Copyright IBM Corporation 2010

The package manager precludes partial and faulty installations and "uninstalls" by adding and removing the files in a package atomically. The package manager also maintains a manifest of all packages installed on the system and can validate the existence of prerequisites and co-requisites beforehand.

If you're a software developer or a systems administrator, providing your application as a package makes installations, upgrades, and maintenance much easier. Here, you learn how to use the popular RPM Package Manager to bundle a utility. For purposes of demonstration, you'll bundle the networking utility `wget`, which downloads files from the Internet. The `wget` utility is useful but isn't commonly found standard in distributions. (An analog, `curl`, is often included in distributions.) Be aware that you can use RPM to distribute most anything—scripts, documentation, and data—and perform nearly any maintenance task.

Building wget manually

The `wget` utility, like many other open source applications, can be built manually. Understanding that process is the starting point for bundling `wget` in a package. Per the general convention, building `wget` requires four steps:

1. Download and unpack the source.
2. Configure the build.
3. Build the code.
4. Install the software.

You can download the latest version of the `wget` source code from ftp.gnu.org (see Resources for a link; as of late September 2009, the current version of `wget` was 1.12). The rest of the steps require the command line, as shown in Listing 1.

Listing 1. Installing wget

```
$ tar xzf wget-latest.tar.gz
$ cd wget-1.12
$ ./configure
configure: configuring for GNU Wget 1.12
checking for a BSD-compatible install... /usr/bin/install -c
checking whether build environment is sane... yes
checking for a thread-safe mkdir -p... build-aux/install-sh -c -d
checking for gawk... no
checking for mawk... no
checking for nawk... no
...
$ make
$ sudo make install
```

`./configure` queries the system and sets compilation options suitable for the hardware and software detected. `make` compiles the code, and `sudo make install` installs the code in system directories. By default, the directories are rooted at /usr/local, although you can change the target root with the `--prefix=`*`/some/full/path/name`* option to `./configure`.

To convert this process to RPM, you place the source in a repository and write a configuration file to dictate where to find the source to be compiled and how to build and install the code. The configuration file, called a *spec file,* is the input to a utility called `rpmbuild`. The spec file and the

binaries are packaged by `rpmbuild` into an RPM. When another user downloads your RPM, the `rpm` utility reads the spec file and installs the package per your prewritten instructions.

Building your first RPM

Before you continue, one word of caution. In the past, packages were built by root, the superuser, because root was the only user able to access the system source code repository. However, this approach was potentially hazardous. Because root can alter any file on the system, it was easy to inadvertently alter a running system by adding extraneous files or removing important files during interim builds of an RPM. More recently, the RPM system changed to allow any user to build RPMs in a home directory. Building an RPM without the privileges of root prevents changes to core system files. Here, the more modern approach is shown.

To build an RPM, you must:

- Set up a directory hierarchy per the `rpmbuild` specifications.
- Place your source code and supplemental files in the proper locations in the hierarchy.
- Create your spec file.
- Build the RPM. You can optionally build a *source RPM* to share your source code with others.

To begin, build the hierarchy. In a directory in your home directory—say, $HOME/mywget—create five subdirectories:

- **BUILD.** BUILD is used as scratch space to actually compile the software.
- **RPMS.** RPMS contains the binary RPM that `rpmbuild` builds.
- **SOURCES.** SOURCES is for source code.
- **SPECS.** SPECS contains your spec file or files—one spec file per RPM you want to build.
- **SRPMS.** SRPMS contains the source RPM built during the process.

At a minimum, you need source code in SOURCES and a spec file in SPECS.

Copy your source, ideally bundled as a tarball, into the SOURCES directory, as shown in Listing 2. If necessary, rename the tarball to include the version number of the application to differentiate it from others. The naming convention is *package-version*.tar.gz. In the case of `wget`, you would use:

Listing 2. Copying your source

```
$ cd ~
$ mkdir mywget
$ cd mywget
$ mkdir BUILD RPMS SOURCES SPECS SRPMS
$ cd SOURCES
$ cp wget-latest.tar.gz .
$ mv wget-latest.tar.gz wget-1.12.tar.gz
$ cd ..
```

Next, create the spec file. A spec file is nothing more than a text file with a special syntax. Listing 3 shows an example of a spec file.

Listing 3. Sample spec file

```
# This is a sample spec file for wget
```

```
%define _topdir   /home/strike/mywget
%define name   wget
%define release  1
%define version  1.12
%define buildroot %{_topdir}/%{name}-%{version}-root

BuildRoot: %{buildroot}
Summary:   GNU wget
License:   GPL
Name:    %{name}
Version:   %{version}
Release:   %{release}
Source:   %{name}-%{version}.tar.gz
Prefix:   /usr
Group:    Development/Tools

%description
The GNU wget program downloads files from the Internet using the command-line.

%prep
%setup -q

%build
./configure
make

%install
make install prefix=$RPM_BUILD_ROOT/usr

%files
%defattr(-,root,root)
/usr/local/bin/wget

%doc %attr(0444,root,root) /usr/local/share/man/man1/wget.1
```

Let's walk through the spec file from top to bottom. Lines 1-5 define a set of convenience variables used throughout the rest of the file. Lines 7-15 set a number of required parameters using the form *parameter: value*. As you can see in line 7 and elsewhere, variables can be evaluated and combined to produce the value of a setting.

The parameter names are largely self-evident, but `BuildRoot` merits some explanation to differentiate it from the BUILD directory you already created. `BuildRoot` is a proxy for the final installation directory. In other words, if `wget` is ultimately installed in /usr/local/bin/wget and other subdirectories in /usr/local, such as /usr/local/man for documentation, `BuildRoot` stands in for /usr/local during the RPM build process. Once you set `BuildRoot`, you can access its value using the `RPM_BUILD_ROOT` environment variable. You should always set `BuildRoot` in your spec file and check the contents of that directory to verify what is going to be installed by the package.

Here are a few tips:

- Do not use `./configure --prefix=$RPM_BUILD_ROOT`. This command builds the entire package, assuming that the final location of the files is the build root. It is likely that this would cause any program that needs to locate its installed files at run time to fail, because when your RPM is finally installed on a user's system, the files aren't under the build root anymore —that's just a temporary directory on your build system.
- Do not include a path in the definition of Source.

- Version and Release are especially important. Each time you change your application's code or data and make a new RPM available, be sure to increment the values of Version and Release to reflect major and minor changes, respectively. You may find it helpful to bump the release number each time you build an RPM, even if for your own use, to keep attempts separate.

The next section starts with `%description`. You should provide a concise but clear description of the software here. This line is shown whenever a user runs `rpm -qi` to query the RPM database. You can explain what the package does, describe any warnings or additional configuration instructions, and more.

The `%prep`, `%build`, and `%install` sections are next, consecutively. Each section generates a shell script that is embedded into the RPM and run subsequently as part of the installation. `%prep` readies the source code, such as unpacking the tarball. Here, `%setup -q` is a `%prep` macro to automatically unpack the tarball named in Source.

The instructions in the `%build` section should look familiar. They are identical to the steps you used to configure and launch the build manually. The `%install` section is identical, too. However, while the target of the manual build was the actual /usr/local directory of your system, the target of the `%install` instruction is ~/mywget/BUILD.

`%files` lists the files that should be bundled into the RPM and optionally sets permissions and other information. Within `%files`, you can use the `%defattr` macro to define the default permissions, owner, and group of files in the RPM; in this example, `%defattr(-,root,root)` installs all the files owned by root, using whatever permissions found when RPM bundled them up from the build system.

You can include multiple files per line in `%files`. You can tag files by adding `%doc` or `%config` to the line. `%doc` tells RPM that the file is a documentation file, so that if a user installs the package using `--excludedocs`, the file is not be installed. `%config` tells RPM that this is a configuration file. During upgrades, RPM will attempt to avoid overwriting a user's carefully modified configuration with an RPM-packaged default configuration file.

Be aware that if you list a directory name under `%files`, RPM includes every file under that directory.

Revving the RPM

Now that your files are in place and your spec file is defined, you are ready to build the actual RPM file. To build it, use the aptly named `rpmbuild` utility:

```
$ rpmbuild -v -bb --clean SPECS/wget.spec
```

This command uses the named spec file to build a binary package (`-bb` for "build binary") with verbose output (`-v`). The build utility removes the build tree after the packages are made (`--clean`). If you also wanted to build the source RPM, specify `-ba` ("build all") instead of `-bb`. (See the `rpmbuild` man page for a complete list of options.)

`rpmbuild` performs these steps:

- Reads and parses the wget.spec file.
- Runs the `%prep` section to unpack the source code into a temporary directory. Here, the temporary directory is BUILD.
- Runs the `%build` section to compile the code.
- Runs the `%install` section to install the code into directories on the build machine.
- Reads the list of files from the `%files` section, gathers them up, and creates a binary RPM (and source RPM files, if you elect).

If you examine your $HOME/mywget directory, you should find a new directory named *wget-1.12-root.* This directory is the proxy for the target destination. You should also find a new directory named RPMS/i386, which should in turn contain your RPM, named *wget-1.12-1.i386.rpm.* The name of the RPM reflects that this is `wget` version 1.12 for the i386 processor.

To verify that the RPM contains the proper files, you can use the `rpm` command, as shown in Listing 4.

Listing 4. Verifying the RPM contents

```
$ rpm -Vp RPMS/i386/wget-1.12-1.i386.rpm
missing     /usr/local/bin/wget
.M....G.    /usr/local/etc
missing   c /usr/local/etc/wgetrc
.M....G.    /usr/local/share
missing     /usr/local/share/info
missing   d /usr/local/share/info/wget.info
missing     /usr/local/share/locale
missing     /usr/local/share/locale/be
missing     /usr/local/share/locale/be/LC_MESSAGES
missing   d /usr/local/share/locale/be/LC_MESSAGES/wget.mo
.
.
.
```

The command `rpm -Vp RPMS/i386/wget-1.12-1.i386.rpm` verifies the package against the files on the system. Although there are seemingly lots of errors, each is a clue that the contents of the RPM file are correct. If you are expecting a file to be installed and it does not appear in the output, it was not included in the package. In that event, review the spec file and make sure the file is enumerated in the `%files` section.

After you've verified the RPM, you can distribute the file to coworkers. Once your colleagues receive the file, they should run `rpm` to install `wget` on their own systems:

```
$ sudo rpm -i wget-1.12-1.i386.rpm
```

Other uses for RPM

This brief introduction merely scratches the surface of what's possible with RPM. Although it is most often used to install software and attendant files, you can package most anything, from system scripts to source code to documentation. And as you'll see in the second installment of

this series, you can also use RPM to patch source code as well as rebuild and reinstall software. The RPM distribution format is found on many Linux systems and is the preferred method to install binary software on Red Hat and Fedora systems, among others.

If you build it and package it with RPM, they will come.

Resources

Learn

- Read all three articles in this series (developerWorks, January 2010) on the RPM Package Manager.
- Read Wikipedia's overview of the RPM Package Manager.
- Visit the official RPM site for more information about package management in Linux.
- Learn about RPM in Red Hat Linux.
- Check out Fedora's RPM Guidelines.
- Get the official RPM software guide from Fedora.
- Read all of Martin's articles on developerWorks
- In the developerWorks Linux zone, find more resources for Linux developers, and scan our most popular articles and tutorials.
- See all Linux tips and Linux tutorials on developerWorks.
- Stay current with developerWorks technical events and Webcasts.

Get products and technologies

- Download the latest version of the `wget` source code from ftp.gnu.org.
- Learn more and download `rpmlint`.
- Learn more and download Easy RPM Builder.
- With IBM trial software, available for download directly from developerWorks, build your next development project on Linux.

Discuss

- 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

Martin Streicher

Martin Streicher is a freelance Ruby on Rails developer and the former Editor-in-Chief of *Linux Magazine*. Martin holds a Master of Science degree in computer science from Purdue University, and has programmed UNIX-like systems since 1986. He collects art and toys.

© Copyright IBM Corporation 2010
(www.ibm.com/legal/copytrade.shtml)
Trademarks
(www.ibm.com/developerworks/ibm/trademarks/)