Packaging software with RPM, Part 2: Upgrading and uninstalling software

Martin Streicher
Software Developer
Pixel, Byte, and Comma

12 January 2010

In this second article in a three-part series on the RPM Package Manager, learn how how to use RPM to upgrade and uninstall software on your Linux® system. (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

In the telecommunications industry, the "last mile" describes the infrastructure, construction, costs, and complications inherent in delivering a service to a great number of consumers. A cable provider may find it a snap to link one end of the country to another via satellite; however, it's quite a different and onerous prospect to gain the rights of way, dig trenches, and lay cable to wire each and every home and business. For some ideas, the *last mile*—a notion not necessarily an exact distance—might as well be a billion miles.

Other industries face the equivalent of telecomm's last mile. Grocers repeatedly fail online, largely because the last mile is so expensive. The United States Postal Service continues to suffer steep losses delivering letters (quite literally) the last mile. Software developers face the last mile, too. Once your software is built, it's just a collection of so many ones and zeros until it's deployed. Conceptually, installation is easy; but as with the delivery of wires, bananas, and envelopes, the devil is in the details.

The first article in this series introduced you to RPM, a popular software-delivery system. You can find RPM on many Linux distributions, and as such, it's widely used to deploy commercial and open source software—for example, variants of Red Hat Linux and Fedora Core Linux. The first article also showed you how to build, package, and install an application from source onto a clean system.

Continuing the investigation into RPM and its many uses, this article dives into upgrading and uninstalling existing software. The third and final article will look at more unusual scenarios, such as patching and distributing source code.

The lowdown on upgrades

Installing software for the first time is the simplest case of deployment. The target system lacks your package's daemons, binaries, configuration files, and/or data files, so a first installation need not stop work in progress or back up, restore, and possibly merge files.

However, if the target system has a prior or current version of your software, or if your software is intertwined with other code and services, your RPM must take special care to maintain a workable configuration. You may need to halt processes, swap or preserve files, and restart applications after your new code has been copied to the system. Of course, it isn't always possible to maintain backward compatibility. In those instances, your RPM should make the fewest changes possible to keep the system functional and secure. In some cases, that may mean elaborate scripts to interpret old settings and transform them into new ones.

RPM provides four hooks for injecting commands into the installation and uninstallation sequences: two for installation and two for uninstallation. All hooks run on the target system and are generally sufficient for most housekeeping chores. These four hooks are:

- All commands listed in the `%pre` hook run before your package is installed.
- Commands in the `%post` hook run after your package has been installed.
- The `%preun` hook runs before your package is removed from the system.
- Commands in the `%postun` hook run after your package is removed from the system.

Hence, the order of operations during an upgrade is:

1. Run the `%pre` section of the RPM being installed.
2. Install the files that the RPM provides.
3. Run the `%post` section of the RPM.
4. Run the `%preun` of the old package.
5. Delete any old files not overwritten by the newer version. (This step deletes files that the new package does not require.)
6. Run the `%postun` hook of the old package.

Steps 4 and 6 may seem a bit suspect, and for good reason: If you are upgrading a package, running the older version's uninstallation hooks could undo portions or all of steps 1 through 3. In fact, without conditions, the uninstallation hooks of the older version could destroy the newer version. To prevent unintentional clobbering, RPM passes each hook one argument, a flag. The value of the flag indicates which operation is being performed:

- If the first argument to `%pre` is `1`, the RPM operation is an initial installation. If the argument to `%pre` is `2`, the operation is an upgrade from an existing version to a new one.
- Similarly, the arguments to a `%post` are `1` and `2` for a new installation and upgrade, respectively. (Again, `%pre` and `%post` aren't executed during an uninstallation.)
- If the first argument to `%preun` and `%postun` is `1`, the action is an upgrade.
- If the first argument to `%preun` and `%postun` is `0`, the action is uninstallation.

Wrap each of your hooks with logic to test the value of the argument and execute the correct code. By default, each hook is interpreted as a Bourne shell (typically, /bin/sh) script, unless you name another script interpreter, such as Perl. Here is a conditional `%pre` hook written for the Bourne shell:

```
%pre
if [ "$1" = "1" ]; then
  # Perform tasks to prepare for the initial installation
elif [ "$1" = "2" ]; then
  # Perform whatever maintenance must occur before the upgrade begins
fi
```

And here is the same hook, albeit written for Perl. To use another interpreter for your hook scripts, specify the `-p` option, and name the fully qualified path to the interpreter.

```
%pre -p /usr/bin/perl
if ( $ARGV[0] == 1 ) {
  print "Preparing for initial install...\n"
}
elsif ( $ARGV[0] == 2 ) {
  print "Preparing to upgrade software...\n"
}
```

If you specify an interpreter that does not exist on the target machine, the RPM utility generates an error that looks something like this:

```
$ sudo rpm -i RPMS/i386/wget-1.12-1.i386.rpm
error: Failed dependencies:
 /usr/bin/perl is needed by wget-1.12-1.i386
```

If the system administrator of the target system sees such a message, he or she should install the RPM for the dependency and retry the installation.

Stay alert with triggers

In general, an RPM installs a single package that needs little from other packages. Typically, an RPM has prerequisites or co-requisites but is otherwise isolated. That said, there are those packages that affect others.

For example, some text editors are extensible: Add a complementary package for Ruby, say, and the editor might highlight the syntax of that programming language. If the editor is installed first followed by its complement, the new feature surfaces as you would expect. But what if the complement is installed first—say, because the extension works for a number of editors—and is followed by the installation of the editor? Ideally, the feature would be added just the same. That's the notion of an RPM *trigger.*

An RPM can attach a trigger to another package to perform one or more tasks if the named packaged is installed or uninstalled *after* your package is installed. Each trigger is just a script, just like those you write for `%pre` or `%post`. You can even name an alternate interpreter, if you prefer.

- A `%triggerin` script runs when the named package is installed.
- `%triggerun` runs when the named package is uninstalled.

- A `%triggerpostun` script runs *after* the named package is uninstalled.

For example, if you want to run a script if Ruby is installed after your package has already been placed on a system, you might write:

```
%triggerin -p /usr/bin/perl -- ruby
# React to the addition of Ruby
```

The `-p` option is optional. You must type `--` (two hyphens), and then name the package you want to monitor.

With copies, hooks, and triggers in mind, here's the order execution of those actions during the installation of a new package, *N.* (The previous version of the package is named *n.*)

1. Run *N*'s `%pre` script.
2. Copy *N*'s new files to the file system.
3. Run *N*'s `%post` script.
4. Run all installation triggers (those marked `%triggerin` in other packages) set off by the installation of *N.*
5. Execute all of *N*'s installation triggers.
6. Run all of *n*'s uninstallation (`%triggerun`) triggers.
7. Run all the uninstallation triggers (those found in other packages) set off by the uninstallation of *n.*
8. Execute the `%preun` hook of *n.*
9. Remove any files not overwritten by the install of *N.*
10. Execute all the uninstallation triggers (`%triggerpostun`) found in *n.*

Installing software can be complex, yet hooks and triggers can accommodate nearly any scenario. One caveat: Do not attempt to interact with a user during any step of the process. RPM is designed to allow batch installations, during which no user is necessarily present. If an RPM package pauses during an installation to ask a question and no one sees the question, the installation will seemingly hang.

RPM variables

RPM files can become lengthy and complex. As you use variables as shorthand in applications, you can use variables as placeholders in an RPM spec file.

For example, you can define a variable near the top of your spec file and refer to it using `%{variable_name}` throughout—and even in your scripts for `%pre`:

```
%define foo_dir /usr/lib/foo

%install
cp install.time.message $RPM_BUILD_ROOT/%{foo_dir}

%files
%{foo_dir}/install.time.message

%post
/bin/cat %{foo_dir}/install.time.message
```

More RPMs to come

If you develop software for UNIX® and Linux machines, writing the installer can be a chore. Happily, you need not write installation technology from scratch. RPM is a capable and widely supported format for software distribution. It makes the "last mile" a walk in the park.

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

- 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.