How to Build an RPM

Author: Chris Negus
Editor: Allison Pranger
10/08/2013

OVERVIEW

You have created some software that you want to install on Red Hat Enterprise Linux systems. Now that it is done, the question is, “How do you gather up the software to make it easy for others to install and manage?” The answer is to package it into an RPM.

Although it is possible to just drop your software (via a tarball or another type of archive file) into a Linux system, packaging your Linux software as an RPM lets you:

- Include metadata with the package that describes its components, version number, size, package group, project URL, and many other pieces of information.
- Add the package to a **`yum`** repository so clients can easily find your software.
- Have clients use common Linux tools (**`yum`**, **`rpm`**, and **PackageKit**) to install, remove, and manage your software.
- Easily update and deploy new versions of the software, using the same Linux installation tools.

You do not have to be a programmer to create RPMs: you only need to understand how to create a SPEC file and use commands to build that SPEC file and package contents into an RPM. These procedures are outlined in this document. Building an RPM is not only useful for managing your company's software, but it is also listed as a skill that could be tested for on a Red Hat Certified Engineer (RHCE) exam.

If you want more information on building RPMs after you complete this tutorial, refer to the following:

- **Red Hat Network Satellite Deployment Guide: RPM Building:** https://access.redhat.com/site/documentation/en-US/Red_Hat_Network_Satellite/5.3/html/Deployment_Guide/satops-rpm-building.html
- **Red Hat Network Satellite User Guide: RPMs:** https://access.redhat.com/site/documentation/en-US/Red_Hat_Network_Satellite/5.4/html/User_Guide/chap-User_Guide-RPMs.html
- **Red Hat Network Satellite Channel Management Guide: Building Custom Packages:** https://access.redhat.com/site/documentation/en-US/Red_Hat_Network_Satellite/5.1.1/html/Channel_Management_Guide/Channel_Management_Guide-Building_Custom_Packages.html

UNDERSTANDING THE PROCESS OF BUILDING RPMS

The process of building an RPM requires knowing how to use a text editor and how to run a few commands to build, sign, and distribute the RPM. With your software in hand, most of the work needed to build the RPM involves creating a SPEC file. Within the SPEC file, you can:

- Identify the commands, configuration files, documentation, and other items in your package.
- Define where components are ultimately installed on the target Linux system.
- Set permissions and ownership of each file.
- Note when your package is dependent on other components being available.

- Tag files as configuration or documentation files.
- Have extra commands executed on the target system when the package is installed or uninstalled (such as creating user accounts, making directories, or moving files around).
- Add **`changelog`** entries to identify what changes have gone into each version of your software.

Once you have mastered the most critical features for building an RPM (as covered in this document), you will find that there is a wealth of features in RPM packaging tools to provide more powerful and flexible ways to create RPMs. For example, you can add platform-specific tags to an SPEC file so you can use the same file to build RPMs for multiple computer architectures.

REBUILDING AN EXISTING SOURCE CODE PACKAGE INTO AN RPM

The best way to learn how to create an RPM package is to start with an existing source code RPM package and rebuild it. Going through the process will let you see the procedures and components that go into building an RPM. This section outlines steps for rebuilding the **`tree`** RPM package from an existing source code package.

> **NOTE:** Once you build this RPM, do not use it on a production system as the package will conflict with one already in your Red Hat Enterprise Linux software channels.

1. **Log In**: Log into a Red Hat Enterprise Linux system as a regular user (not root).
2. **Get a Source Code Package**: Download a working source code package. This example uses the **`tree`** source code package:

```
$ wget ftp://ftp.redhat.com/pub/redhat/linux/enterprise/6Workstation/en/os/SRPMS/tree-1.5.3-2.el6.src.rpm
```

3. **Install the Source Code**: Install the source code (which should be in your current directory) into a new **`rpmbuild`** directory:

```
$ rpm -ihv tree-1.5.3-2.el6.src.rpm
```

This creates an **`rpmbuild`** directory structure in your home directory similar to the following:

```
~/SPECS
~/SPECS/tree.spec
~/BUILDROOT
~/SOURCES
~/SOURCES/tree-1.5.3.tgz
~/SOURCES/tree-1.2-no-strip.patch
~/SOURCES/tree-no-color-by-default.patch
~/SOURCES/tree-1.2-carrot.patch
~/SOURCES/tree-preserve-timestamps.patch
```

Notice that the new **`rpmbuild`** directory in your home directory includes a **SPECS** directory (which includes the **`tree.spec`** file) and a **SOURCES** directory. The **SOURCES** directory includes the **`tree-1.5.3.tgz`** tarball of the code and four patch files.

4. **Edit the Spec File**: Review (and possibly change) the spec file. Using either the **vim** or **emacs** editor will add color to the file as you edit it: **`vim ~/rpmbuild/SPECS/tree.spec`**. An example of that file is shown below. Save and exit the file after you are done changing it.

```
Summary: File system tree viewer
Name: tree
Version: 1.5.3
Release: 2%{?dist}
Group: Applications/File
License: GPLv2+
Url: http://mama.indstate.edu/users/ice/tree/
Source: ftp://mama.indstate.edu/linux/tree/tree-%{version}.tgz
Patch1: tree-1.2-carrot.patch
Patch2: tree-1.2-no-strip.patch
Patch3: tree-preserve-timestamps.patch
Patch4: tree-no-color-by-default.patch

BuildRoot: %{_tmppath}/%{name}-%{version}-%{release}-root-%(%{__id_u} -n)

%description
The tree utility recursively displays the contents of directories in a tree-
like format. Tree is basically a UNIX port of the DOS tree utility.%prep
%setup -q
# Fixed spelling mistake in man page.
%patch1 -p1 -b .carrot
# Don't strip binary in the Makefile -- let rpmbuild do it.
%patch2 -p1 -b .no-strip
# Preserve timestamp on man page.
%patch3 -p1 -b .preserve-timestamps
# Disable color output by default.
%patch4 -p1 -b .no-color-by-default
%build
make CFLAGS="$RPM_OPT_FLAGS" "CPPFLAGS=$(getconf LFS_CFLAGS)" %{?_smp_mlags}

%install
rm -rf $RPM_BUILD_ROOT
mkdir -p $RPM_BUILD_ROOT%{_bindir}
make    BINDIR=$RPM_BUILD_ROOT%{_bindir} \
       MANDIR=$RPM_BUILD_ROOT%{_mandir}/man1 \
       install
chmod -x $RPM_BUILD_ROOT%{_mandir}/man1/tree.1

%clean
rm -rf $RPM_BUILD_ROOT

%files
%defattr(-,root,root)
%{_bindir}/tree
%{_mandir}/man1/tree.1*
%doc README LICENSE
%changelog
  ...
```

After you install the rpm-build package, you can read about features of SPEC files in the **`/usr/share/doc/rpm-build*/spec`** file. The **Name** is the base name of the package. The **Summary** is a one-line description of the package. **Version** is the upstream version number on the package, while **Release** is the number you add as the packager to reflect multiple builds of the same upstream version (such as for bug fixes).

The **URL** points to the project site that produced the source code, and **Source** points to where the original source code used to make the package came from. **BuildRoot** identifies the location of the temporary directory where the RPM will be built. Other lines prepare the build environment, add patches, compile and build the software, identify the files and permissions in the package, and allow you to keep a log to the changes over time.

At the end of this document, you can find some **`rpm -qp`** options that you can use to check the content of the package you build.

5. **Build the RPM:** Use the **`rpmbuild`** command to turn your spec file and content into the RPM package for distribution. You can also package the source code into a separate source RPM (**`src.rpm`**). Install the **`rpm-build`** package (as root) and run **`rpmbuild`** (from your regular user account):

```
# yum install rpm-build                          Run as root
$ rpmbuild -ba ~/rpmbuild/SPECS/tree.spec    Run as regular user account
```

This results in a binary RPM and a source RPM in the **RPMS** and **SRPMS** subdirectories, respectively.

6. **Sign the RPM:** Signing an RPM requires that you create a public and private key pair, use the private key to sign your RPM, and then distribute the public key to clients so they can use that key to check your signed package.

```
$ gpg --gen-key      Generate public/private keys
```

When you generate your public/private keys, you can use most of the defaults. The end of the output will be similar to the following:

```
pub   2048R/99A9CF07 2011-09-16
Key fingerprint = 90BF B5DC 628E C9E0 88D0  E5D1 E828 4641 99A9 CF07
uid  Chris Negus (My own build of the tree package.) <cnegus@redhat.com>
sub   2048R/48E60E56 2011-09-16
```

Use the key ID generated (in this case, **`99A9CF07`**) to export your private key to a public key:

```
$ gpg -a -o RPM-GPG-KEY-ABC –-export 99A9CF07     Export public key
```

To make sure the key ID is used to sign your package, add a **`_gpg_`**name line to the **`.rpmmacros`** file in your home directory:

```
$ vi ~/.rpmmacros                  Add _gpg_name keyID to your .rpmmacros file
%_gpg_name 99A9CF07
```

Now you are ready to sign the package:

```
$ rpm –-resign  ~/rpmbuild/RPMS/x86_64/tree-1.5.3-2.el6.x86_64.rpm   Sign pkg
```

7. **Publish the RPM in a `yum` Repository:** One way to make your RPM accessible is to create a **`yum`** repository that is accessible from your web server. Assuming a web server is running on the system on which you build your RPM, these steps publish the RPM and make a **`yum`** repository:

```
# mkdir /var/www/html/abc
# cp ~/RPM-GPG-KEY-ABC /var/www/html/abc/        Make the public key available
# cp ~/rpmbuild/RPMS/x86_84/tree-1.5.3-2.el6.x86_64.rpm /var/www/html/abc/
# createrepo /var/www/html/abc                   Create the repository
```

8. **Create a Repository (`.repo`) File:** Create a **`.repo`** file that identifies the URL to the repository. Clients that want to install the package will be able to simply copy the **`abc.repo`** file to their own RHEL system's **`/etc/yum.repos.d`** directory to enable it. Replace *`whatever.example.com`* with the FQDN of your own web server:

```
$ vim abc.repo
      [abc-repo]
     name=My ABC yum repository
     baseurl=http://whatever.example.com/abc
     gpgkey= http://whatever.example.com/RPM-GPG-KEY-ABC

$ cp abc.repo /var/www/html/abc
```

9. **Prepare Clients to Install the RPM:** To install your RPM, clients can simply copy your **`.repo`** file to their systems, then use the **`yum`** command to install any package from your repository:

```
# wget http://whatever.example.com/abc/abc.repo -O /etc/yum.repos.d/abc.repo
# yum install tree
```

To update your RPM in the future, you can simply rebuild the RPM, copy the latest version to your **`yum`** repository directory, and rerun the **`createrepo`** command. Clients will get the new RPM the next time they install or update the package.

CHECKING YOUR RPM PACKAGE

Once you have finished building your RPM, you can use the **`rpm`** command to check its contents and make sure the signature worked properly. You can do this on any Red Hat Enterprise Linux system, as long as you can get the package and the public key. Start by importing the key used to sign the package and checking the signature:

```
# rpm --import ~/RPM-GPG-KEY-ABC                                  Import key file
$ rpm -K ~/rpmbuild/RPMS/x86_64/tree-1.5.3-2.el6.x86_64.rpm  Check signature
~/rpmbuild/RPMS/x86_64/tree-1.5.3-2.el6.x86_64.rpm: sha1 md5 OK
```

Next, check the contents of the file. This can be accomplished in various way. The following options query (**`q`**) a package (**`p`**), as opposed to the RPM database, and show information (**`i`**):

```
$ rpm -qpi ~/rpmbuild/RPMS/x86_64/tree-1.5.3-2.el6.x86_64.rpm
Name        : tree                         Relocations: (not relocatable)
Version     : 1.5.3                             Vendor: (none)
Release     : 2.el6                      Build Date: Thu 15 Sep 2011 11:53:37 PM EDT
Install Date: (not installed)            Build Host: cnegus.linuxtoys.net
Group       : Applications/File          Source RPM: tree-1.5.3-2.el6.src.rpm
Size        : 73868                         License: GPLv2+
Signature   : (none)
URL         : http://mama.indstate.edu/users/ice/tree/
Summary     : File system tree viewer
Description :
```

```
The tree utility recursively displays the contents of directories in a
tree-like format. Tree is basically a UNIX port of the DOS tree
utility.
```

With the **-l** (lowercase L) option, you can list the contents of a package:

```
$ rpm -qpl ~/rpmbuild/RPMS/x86_64/tree-1.5.3-2.el6.x86_64.rpm
/usr/bin/tree
/usr/share/doc/tree-1.5.3
/usr/share/doc/tree-1.5.3/LICENSE
/usr/share/doc/tree-1.5.3/README
/usr/share/man/man1/tree.1.gz
```

There are many other options to the **rpm** command for checking and working with RPM packages. Refer to the **rpm** man page for details (type **man rpm**).

BUILDING AN RPM FROM SCRATCH

To create your own RPM, you need to create your own spec file (and put it in the **SPECS** directory) and gather into a tarball the executables, scripts, user documentation files, and configuration files you want included in the RPM. You can create your spec file by simply copying an existing spec file and modifying it. As an alternative, you can use the **vim** or **emacs** command to open any new file that ends in **.spec**. The editor will automatically create a template within the new file for writing an RPM spec file. You can then follow the rest of the procedure described earlier in this document.

To look at an example of a tarball of content for an RPM, try untarring the **tree** tarball included in the **tree** source code package:

```
$ tar xvf ~/rpmbuild/SOURCES/tree-1.5.3.tgz
tree-1.5.3/CHANGES
tree-1.5.3/INSTALL
tree-1.5.3/LICENSE
tree-1.5.3/Makefile
tree-1.5.3/README
tree-1.5.3/tree.c
tree-1.5.3/strverscmp.c
tree-1.5.3/man/tree.1
tree-1.5.3/man/tree.1.fr
```

To create your own tarball, you can simply put your content in a directory (such as **~/abc-1.0**) and gather it into a tarball that is placed into the **SOURCES** directory:

```
$ tar -cvzf ~/rpmbuild/SOURCES/abc-1.0-1.tar.gz ~/abc-1.0/
```

GETTING MORE HELP WITH RPMS

There is more to building RPMs than is described in this document. Most of the complexity comes in building the spec files. You can identify dependencies on other components, set the types and permissions of files in the RPM, and run scripts when the package is installed or uninstalled. Below are some other places you can look for help.

How to Create an RPM Package (Fedora Project)

Thousands of RPM packages have been created for the Fedora Project. The FedoraProject.org site has an excellent document on creating RPMs that are similar to those you would use for Red Hat Enterprise Linux: http://fedoraproject.org/wiki/How_to_create_an_RPM_package.

Packaging Software with RPM (IBM)

More information to help you get started building your first RPM can also be found at the following site: http://www.ibm.com/developerworks/library/l-rpm1/.