Building Perl RPMs

Daniel S. Sterling
Devops at Plus Three
@eqhmcow

LIVE DEMO

oh goodness, how
did this get here

halp I am not good wit
computer

What is RPM?

The default package manager on Red Hat and Red Hat-derived systems.

A tool to install, inventory, manage and remove files.

A way to organize, distribute, install, upgrade and downgrade packages on multiple systems.

Why use a package manager?

Automate building things from source.

Only build once per target environment per package.

Automates dependency management. You can't install a package that won't work because another package is missing.

Why use a package manager? (cont)

Easily install, upgrade, downgrade and remove packages.

Identify configuration files and compare current configuration to original.

Easily inventory and verify installed packages and files. Accidentally delete /bin ? You can recover! Where did this file come from? You can find out!

How do I build a package?

You write a spec file and then run rpmbuild.

A spec file is a bunch of metadata and shell script that tells rpmbuild what to do.

RPM has a lot of macros, and a lot of associated magic automatically invoked by the default set of macros.

Macros, you say?

Yes, a lot of them. And they're not well-documented.

It's kind of a crisis.

Some links:

http://www.rpm.org/max-rpm-snapshot/s1-rpm-inside-macros.html
http://rpm.org/api/4.4.2.2/config_macros.html

halp

OK, it's not actually so bad.

Macros generally just are shorthand for common package building tasks. Mostly, they stay behind the scenes and just work.

The problem is, they're often invoked by default. That means there is literally no line in your code (spec file) that caused this weird behaviour that's now causing you a problem...

halp (cont)

OK, maybe it is a crisis after all.

If a macro is acting up, you have to figure out which one it is, sometimes without being able to see what is actually causing that macro to execute, or even what the macro name is.

Once you find the offending macro, you can redefine it to fix whatever broken behaviour you're seeing.

halp (cont)

But, macros mostly just work, so you mostly don't have to care about them.

Umm, let's move on.

RPM has macros that do things that usually work.

OK.

You said spec files have shell script?

Yes! Shell is actually a big part of spec files.

Spec files are divided up into sections.

Each section consists of either metadata or a shell script.

See http://fedoraproject.org/wiki/How_to_create_an_RPM_package#Creating_a_SPEC_file

spec file sections

The major spec file sections are:

%description
%prep
%build
%install
%check
%clean
%files

spec file sections (cont)

rpmbuild parses the spec file and then processes and executes each section in a predefined order.

Some sections contain metadata, like a block of text with a description of the package.

Other sections contain shell script, which rpmbuild simply executes.

So rpmbuild is just a way to run convoluted shell scripts?

Yes! But, unfortunately, rather quite convoluted. Remember those macros.

But yes, when you get down to it, building an RPM is simply just a matter of writing a shell script that puts some files in a directory, which rpmbuild then packages up into an .rpm file.

Taking advantage of this fact can lead to some rather interesting spec files...

So, literally, just shell script?

Yes!

spec files are quite literally just shell scripts, with some extra metadata and a weird way to run them.

Most spec files just use the default macros to do a standard build, but you can write whatever shell code you want to in a spec file, and rpmbuild will run it.

default build

The usual spec file operates on one source package and 0 or more patch files.

The source package is untar'd, patches are applied, and a standard build is executed (configure, make) in the build directory.

Then, "make install" is executed to install the package files into a "build root" directory.

DO NOT RUN RPMBUILD AS ROOT

Let me take a quick moment to note that you should

NEVER RUN RPMBUILD AS ROOT.

spec files (or their macros) often do a lot of rm'ing of old build directories. And other crazy stuff.

You don't want to be running those rm commands as root. Or any of the other build commands. ALWAYS build packages as a regular user.

rpmbuilding a package

During a normal build,

the %prep section extracts the source package,

the %build section builds it in the build directory,

and the %install section stages the final package files into the "build root" directory.

rpmbuilding a package (cont)

Again, all those three sections are literally just shell scripts, into which any shell code of any sort can be put.

During a normal build, macros are used as shorthand for the usual build and install commands. A spec file author could just as easily forgo macros and explicitly specify the build commands (but they usually wouldn't have any reason to do so).

rpmbuilding a package (cont)

The result of the %install stage will be a bunch of files in the "build root" directory. Each and every file and subdirectory in the "build root" directory must be listed in the %files section; otherwise rpmbuild will throw an error, as it assumes you didn't mean to package the unlisted files and your build isn't working correctly.

(Wildcard matching is allowed in the %files section).

rpmbuilding a package (cont)

Finally, all the files listed in the %files section are packaged up into an .rpm package that can be installed.

And, that's it; that's how you build an rpm package.

cpanspec

cpanspec is a tool maintained by Fedora that creates an initial spec file from a CPAN distribution.

The spec file it creates will often "just work" -- but you should review and update it (at least check the docs and update the changelog).

cpanspec (cont)

There's no need to re-run cpanspec if you already have a spec file for a Perl module.

You can also steal and re-use spec files out of the Red Hat SRPM (source RPM) repository; there's usually never a need to create a spec file entirely from scratch.

See the live demo for examples using cpanspec.

Conclusion

This talk discussed:
* what RPM and package management is
* some benefits of using a package manager

* what spec files are

* how to use rpmbuild to build a spec file

* cpanspec

Thank you!

Daniel S. Sterling
Devops at Plus Three
@eqhmcow