How to create a GNU Hello RPM package

In other languages: Deutsch (de) (https://fedoraproject.org/wiki/How_to_create_a_GNU_Hello_RPM_package/de) | **English (en)** | Italiano (it) (https://fedoraproject.org/wiki/How_to_create_a_GNU_Hello_RPM_package/it) | 中文(中国大陆) (zh-cn) (https://fedoraproject.org/wiki/How_to_create_a_GNU_Hello_RPM_package/zh-cn) | 中文(台灣) (zh-tw) (https://fedoraproject.org/wiki/How_to_create_a_GNU_Hello_RPM_package/zh-tw) [edit (https://fedoraproject.org/w/index.php?title=Template:Lang/How_to_create_a_GNU_Hello_RPM_package&action=edit)]

This is a short hands-on tutorial on writing RPM files, showing how to quickly step up to create simple source and binary software packages. It assumes some familiarity with using pre-made RPM packages, and with the FOSS software building process.

For a comprehensive information on how to create RPM files, including more detailed tips, refer to How to create an RPM package. If you plan to create an RPM package for the Fedora repository, follow the process for How to join the Fedora Package Collection Maintainers, including following the various Fedora guidance.

This tutorial demonstrates packaging of the GNU "Hello World" project. While the C program printing 'Hello World" to standard output is trivial, the GNU version contains most of the usual peripheral components associated with a typical FOSS software project, including the configuration/build/install environment, documentation, internationalization, etc. The GNU version, however, traditionally consists of a tar file containing the source code and configure/make scripts, but it does not include the packaging information. Therefore, it's a reasonable vehicle to practice building RPMs on.

Development environment

To build RPMs we need a set of development tools. This is a one-time-only setup, installed by running those commands from a system administration (`root`) account:

```
# yum install @development-tools
# yum install fedora-packager
```

If you want to test the build procedure in a clean chroot, you need to configure your non-privileged account to be a member of the 'mock' group:

```
# usermod -a -G mock <your username>
```

Those are the only commands requiring `root` privileges. All the remaining work should be done from your regular, non-privileged account, or even from a separate account created just for development work. Modern RPM-based systems, including Fedora, are set up to build and test RPM packages purely from within a non-privileged account. The command

```
$ rpmdev-setuptree
```

sets up a RPM build area in your `~/rpmbuild`. This directory will contain several subdirectories, for the project source code, RPM configuration files and for the resulting source and binary packages.

Building a "Hello World" RPM

We need the source code of the project we are packaging, often referred to as the 'upstream' source. We will download it from the project's website into the `~/rpmbuild/SOURCE` directory. We are getting the compressed tarball archive, which happens to be a preferred distribution form for most FOSS projects.

```
$ cd ~/rpmbuild/SOURCES
$ wget http://ftp.gnu.org/gnu/hello/hello-2.8.tar.gz
```

The RPM package is configured by `.spec` files. We will create a template file `hello.spec` in the appropriate directory:

```
$ cd ~/rpmbuild/SPECS
$ rpmdev-newspec hello
```

Recent versions of `Emacs` and `vi` have .spec file editing modes which will also bring up a similar template upon creating a new file. So you can just use the following command for example to use the template automatically.

```
vi hello.spec
```

Inside a `.spec` file

The fields in our `.spec` file need slight editing. Please follow the Fedora rules for these fields. In our case, the file might start as follows:

```
Name:     hello
Version:  2.8
Release:  1
Summary:  The "Hello World" program from GNU
License:  GPLv3+
URL:      http://ftp.gnu.org/gnu/hello
Source0:  http://ftp.gnu.org/gnu/hello/hello-2.8.tar.gz

%description
The "Hello World" program, done with all bells and whistles of a proper FOSS
project, including configuration, build, internationalization, help files, etc.

%changelog
* Thu Jul 07 2011 The Coon of Ty <Ty@coon.org> - 2.8-1
- Initial version of the package
```

The `Version` should mirror upstream while `Release` numbers our work within Fedora.

The first letter of the `Summary` should be uppercase to avoid rpmlint complaints.

It is your responsibility to check the `License` status of the software, by inspecting the source files and/or their LICENSE files, and/or by talking to the authors.

The `Group` tag was historically used to classify the package in accordance to the list in `/usr/share/doc/rpm-<version>/GROUPS`. It is being phased out so you will not see it added by default. However, it doesn't hurt to add it anyway.

The `%changelog` should document the work on preparing the RPM, especially if there are security and bug patches included on top of the base upstream source. Changelog data can be displayed by `rpm --changelog -q <packagename>`, which is very useful for instance to find out if specific bug and security patches were included in the installed software, thanks to diligent Fedora packagers who include this info with the relevant CVE (http://cve.mitre.org/) numbers.

The changelog entry should include the version string to avoid rpmlint complaints.

Multi-line sections like `%changelog` or `%description` start on a line under the directive, and end with an empty line.

Lines which aren't needed (e.g. `BuildRequires` and `Requires`) can be commented out with a hash ('#') for now.

Many lines in the template don't need to be changed at all in many cases, at least for the initial attempt.

Building the package

We are ready for the first run to build source, binary and debugging packages:

```
$ rpmbuild -ba hello.spec
```

It will complain and list the unpackaged files, i.e. the files that would be installed in the system that weren't declared as belonging to the package. We need to declare them in the `%files` section. Do not hardcode names like `/usr/bin/`, but use macros, like `%{_bindir}/hello` instead. The manual pages should be declared in the `%doc` subsection: `%doc %{_mandir}/man1/hello.1.gz`.

This is an iterative process: after editing the `.spec` file, rerun `rpmbuild`.

Since our program uses translations and internationalization, we are seeing a lot of undeclared i18 files. The recommended method to declare them is:

- find the filenames in the `%install` step: `%find_lang %{name}`
- add the required build dependencies: `BuildRequires: gettext`

- use the found filenames `%files -f %{name}.lang`

If the program uses GNU info files, you need to make sure the installation and uninstallation of the package does not interfere with other software on the system, by using this boilerplate:

- delete the 'dir' file in %install: `rm -f %{buildroot}/%{_infodir}/dir`
- `Requires(post): info` and `Requires(preun): info`
- add those steps:

```
%post
/sbin/install-info %{_infodir}/%{name}.info %{_infodir}/dir || :

%preun
if [ $1 = 0 ] ; then
/sbin/install-info --delete %{_infodir}/%{name}.info %{_infodir}/dir || :
fi
```

A complete `hello.spec` file

Here's the initial version of `hello.spec`:

```
Name:           hello
Version:        2.8
Release:        1%{?dist}
Summary:        The "Hello World" program from GNU

License:        GPLv3+
URL:            http://ftp.gnu.org/gnu/%{name}
Source0:        http://ftp.gnu.org/gnu/%{name}/%{name}-%{version}.tar.gz

BuildRequires: gettext

Requires(post): info
Requires(preun): info

%description
The "Hello World" program, done with all bells and whistles of a proper FOSS
project, including configuration, build, internationalization, help files, etc.

%prep
%setup -q

%build
%configure
make %{?_smp_mflags}

%install
%make_install
%find_lang %{name}
rm -f %{buildroot}/%{_infodir}/dir

%post
/sbin/install-info %{_infodir}/%{name}.info %{_infodir}/dir || :

%preun
if [ $1 = 0 ] ; then
/sbin/install-info --delete %{_infodir}/%{name}.info %{_infodir}/dir || :
fi

%files -f %{name}.lang
%doc AUTHORS ChangeLog COPYING NEWS README THANKS TODO
%{_mandir}/man1/hello.1.gz
%{_infodir}/%{name}.info.gz
%{_bindir}/hello

%changelog
* Tue Sep 06 2011 The Coon of Ty <Ty@coon.org> 2.8-1
- Initial version of the package
```

With this spec file, you should be able to successfully complete the build process, and create the source and binary RPM packages.

Next you should check them for conformance with RPM design rules, by running `rpmlint` on the spec file and all RPMs:

```
$ rpmlint hello.spec ../SRPMS/hello* ../RPMS/*/hello*
```

If there are no warnings or errors, we've succeeded. Otherwise, use `rpmlint -i` or `rpmlint -I <error_code>` to see a more verbose description of the `rpmlint` diagnostics.

The `mock` builds

To check that the package build will succeed in the Fedora restricted build environment, check it with mock. The default mock configuration builds the package against Rawhide - the Fedora development branch.

```
$ mock --verbose ../SRPMS/hello-2.7-1.fc19.src.rpm
```

References

- How to create an RPM package
- Building RPM packages (20090405)
- Using Mock to test package builds
- Using the Koji build system

History

Przemek Klosowski wrote this tutorial when he worked through Christoph Wickert's IRC session on building RPMs using Rahul Sundaram suggestion of GNU "Hello World" as a test case. After he wrote up his experience, he found out about the excellent and extensive How to create an RPM package page on this wiki, as well as the Christian Lyder Jacobsen's website (http://www.absolutepanic.org/blog/2009/07/building-a-gnu-hello-world-rpm) . However, Christian isn't planning to update his site, and it seemed that a 5-minute 'fast food' alternative to the more extensive article might suit some people. More in-depth information on using and building RPM packages is available from other sources.

Categories: Package Maintainers | How to