Building Good Fedora Packages

RPM Packaging Workshop for Beginners

Pavel Raiskup · Jiří Kyjovský · Miroslav Suchý

Original materials were created by Tom "spot" Callaway

Things to do to prepare your environment

- Packages that you will need installed:

```
sudo dnf install fedora-packager rpm-build rpmdevtools rpmlint patch
```

- Get a copy of the files we will be working with:
  - `materials/` folder in this repository
- Use the `rpmdev-setuptree` command to create `~/rpmbuild` directory tree:

```
[jkyjovsk@fedora ~]$ rpmdev-setuptree
[jkyjovsk@fedora ~]$ tree ~/rpmbuild/
/home/jkyjovsk/rpmbuild/
├── BUILD
├── RPMS
├── SOURCES
├── SPECS
└── SRPMS

6 directories, 0 files
[jkyjovsk@fedora ~]$
```

Agenda

Intro to RPM, create a simple RPM package from scratch

- **Assumptions**
  - You know how to use a text editor (no matter which one)
  - You know how to manually build "normal" software for Linux
  - You know what is a patch and how to apply it
- **Limitations**
  - This workshop covers a simple piece of software
  - Most packages will be a bit more complicated
    - Some will be a LOT more complicated
- **Information**
  - Feel free to ask questions! Or for a break!

Objectives — RPM Package Management

- Why to use RPM packages?
- Can package be problematic?
- What is source and binary package?
- Package file name vs. package name
- Basic commands

Packaging as a standard (aka, why package at all?)

- Auditing software usage — What, where?
- Version control
- Reproducible installations (Kickstart integration via anaconda, Ansible)
- Minimizes risk
  - Security
  - Rogue Applications
  - Licensing
  - Trusted provider

> ⚠️ So you see why `sudo make install` is just not enough...

RPMs works well

- RPM itself is solid technology
- Package creation is easier than you think
- Easy to install and remove
- The quality depends on the packager

- You can write good software and bad software
- And you can write good packages and bad packages

✅ Our goal today: learn to write **good** packages!

RPM Usage

- RPM is archive (actually CPIO, try `/bin/mc` to step inside)
- Binary (built) package `pkg-workshop-1-0.x86_64.rpm`
  - ⚠️ File name is **different** from package name

Install packages with file name:

```
# i for install, v for verbose, h for process hash
rpm -ivh pkg-workshop-1-0.x86_64.rpm
```

Remove package with package name:

```
# e for erase
rpm -eh pkg-workshop
```

Query installed package with package name:

```
# q for query, l for list files
rpm -ql pkg-workshop
```

List all installed packages in RPM database:

```
# info about package
rpm -qa
rpm -qi <pkgname>
```

ℹ️ There's DNF5/DNF/YUM on top of RPM database, but that's not the topic for today.

Source RPM Overview

- Source Package `pkg-workshop-1-0.src.rpm`
  - SRPMs contain sources/patches/spec file used to generate binary RPM packages
- Install SRPM package with SRPM file name:

```
# i for install, v for verbose, h for process hash
rpm -ivh pkg-workshop-1-0.src.rpm
```

- Source packages just install source into defined source directory
  - Red Hat default: `~/rpmbuild/SOURCES`
- SRPMs do not go into the RPM database
- Remove installed SRPM with spec file name:

```
rpmbuild --rmsource --rmspec pkg-workshop.spec
```

More Source RPM Overview

- Making a binary rpm from SRPM:

```
rpmbuild --rebuild pkg-workshop-1-0.src.rpm
```

- Making a binary rpm from spec file:

```
# -b for build, -a for producing all (sub)packages, src.rpm and binary
rpmbuild -ba pkg-workshop.spec
```

Objectives — Spec file

- What is a SPEC file
- Generators
- Basic syntax:
  - RPM Macros
  - Comments
- Best practices

Cooking with Spec Files

- It's a text file
- Think of a spec file as a recipe
- Defines the contents of the RPMs
- Describes the process to build, install the sources
- Required to make packages
- Contains shell scripts

Sections/stages:

1. Preamble — metadata
2. Setup (`%prep`)
3. Build (`%build`)
4. Install (`%install`)
5. Clean (`%clean`)
6. Files (`%files`)
7. Changelog (`%changelog`)

Common mistakes new packagers make

❌ Spec file generators

- Remember, functional is not the same as good.

❌ Packaging pre-built binaries, not building from source

- (Technically) possible, but you shouldn't start here if you can help it
- Not permitted in Fedora

❌ Disabling automatic checks

- This is a recipe for disaster, they exist for a reason
- For example "installed but unpacked" check (you'll see later..)

RPM Macros

- Similar to shell variables
  - They can act as integers or strings, but it's easier to always treat them as strings
  - Recursively expanded: `%{_bindir}` → `%{_exec_prefix}/bin` → `/usr/bin`
- Macro formats: `%{foo}` and `%foo`
  - Some macros have environment variable variant, e.g. `$RPM_BUILD_ROOT` vs. `%buildroot` — they hold the same value, but for your sanity (and guidelines), you should consistently use one type of macro in a spec file.
- Many common macros come predefined
  - `rpm --showrc` will show you all the defined macros
    - the numbers in output explanation
  - `rpm --eval %macroname` will show you what a specific macro evaluates to
  - Most system path macros begin with an _ (e.g. `%{_bindir}`)

RPM Comments

- To add a comment to your RPM spec file, simply **start a new line** with a `#` symbol.
  - ❌ Don't: `Version: 0.0.0.1 # beta, but cool!`
- Good example:

```
# I have to delete this file, or else it will not build properly
rm -f foo/bar/broken.c
```

- RPM ignores comment lines entirely
  - Well, to be fair, this isn't true — if you comment out a macro definition, it will evaluate it anyways. Use double `%`:

```
Before: # %configure          ← WRONG, still runs!
After:  # %%configure         ← CORRECT, escaped
```

⚠️ Note that macros are Turing complete! `# %(rm -rf stg)` — this WILL execute!

Custom rpm macros `~/.rpmmacros`

- Syntax: `%macroname value`
- Overrides for system-defined macros `/usr/lib/rpm/`
- Can be overwritten directly in spec file:
  - Syntax: `%global macroname value`
- Example: `%_smp_mflags` expands to `-j<N>` (number of CPU cores for parallel make)

⚠️ You can make your own macros here, be careful! (why?)
... so, **don't rely on them in spec file** — other builders won't have your `~/.rpmmacros`!

Fedora Packaging Guidelines

- Intended to document a set of "rules" and "best practices"
- Living document, constantly being amended and improved
- Exceptions are possible
  - Common exception cases are usually documented
  - If you can justify doing something differently, it is usually permissible, although, it may need to be approved by the Fedora Engineering Steering Committee (FESCo)
  - Use common sense, but when in doubt, defer to the guidelines

📖 **https://docs.fedoraproject.org/en-US/packaging-guidelines/**

Objectives — real-life spec — preamble

- What is a preamble?
- What tags can be defined in a preamble?
- Write simple preamble section.

Understanding the Spec File: Preamble

- Initial section — mostly metadata (typically no scripts)
  - Defines output of `rpm -qi <package>`
- Defines package characteristics
  - And custom macro definitions

```
[jkyjovsk@fedora]$ rpm -qi tar
Name        : tar
Epoch       : 2
Version     : 1.35
Release     : 6.fc43
Architecture: x86_64
Install Date: Mon Jan  5 06:47:02 2026
Group       : Unspecified
Size        : 3093037
License     : GPL-3.0-or-later
Signature   :
              RSA/SHA256, Mon Jul 28 01:00:50 2025, Key ID 829b60663164
Source RPM  : tar-1.35-6.fc43.src.rpm
Build Date  : Sat Jul 26 08:44:17 2025
Build Host  : buildhw-x86-02.rdu3.fedoraproject.org
Packager    : Fedora Project
Vendor      : Fedora Project
URL         : https://www.gnu.org/software/tar/
Bug URL     : https://bugz.fedoraproject.org/tar
Summary     : GNU file archiving program
Description :
The GNU tar program saves many files together in one archive and can
```

Creating a new spec file

- A spec file is a text file, grab a text editor
- RPM expects spec files to be in `~/rpmbuild/SPECS/`
    - Technically, it doesn't care, but for sanity, lets just keep them there.
- Go ahead and open a new text file called `~/rpmbuild/SPECS/enum.spec`

```
rpmdev-setuptree && rpmdev-newspec ~/rpmbuild/SPECS/enum.spec
```

Preamble Items

Here is a blank preamble section, this is where we will start our package!

```
Name:
Version:
Release:
Summary:
License:
URL:
Source0:

%description
```

Name

First, lets fill in the name of our package, which is "enum".

```
Name:       enum
Version:
Release:
Summary:
License:
URL:
Source0:

%description
```

- Note: Either use spaces or tabs to separate your fields. Doesn't matter which one as long as you are **consistent**.
- https://docs.fedoraproject.org/en-US/packaging-guidelines/Naming/

Version

```
Name:       enum
Version:    1.1
Release:
Summary:
License:
URL:
Source0:

%description
```

- Hopefully, it should be obvious where I got the version from. :)
- Use `/bin/rpmdev-vercmp` if you are in doubt (e.g. beta versions)!
- If your package is a pre or a post release, there are special rules for handling Version and Release, see:
  - https://docs.fedoraproject.org/en-US/packaging-guidelines/Versioning/

Release

```
Name:       enum
Version:    1.1
Release:    1%{?dist}
Summary:
License:
URL:
Source0:

%description
```

- The Release field is where you track your package builds, starting from 1 (guidelines), and incrementing by 1 each time you make a change.

Wait, what is that `%{?dist}` thing?

- It is a macro!
- It is there to add an identifier (or dist) tag to the end of the release field. For humans!
- The `?` means "if defined, use it, if not defined, evaluate to nothing"
- So, if your release field is:

```
Release: 1%{?dist}
```

- Then, it evaluates to `1.fc42` on Fedora 42, and `1.el10` on RHEL 10.

ℹ️ More info: https://docs.fedoraproject.org/en-US/packaging-guidelines/DistTag/

Summary

```
Name:       enum
Version:    1.1
Release:    1%{?dist}
Summary:    Seq- and jot-like enumerator
License:
URL:
Source0:

%description
```

- Summary is a single sentence describing what the package does. It does not end in a period, and is no longer than 80 characters.

ℹ️ We'll fill in a longer description of the package in `%description`.

License

```
Name:       enum
Version:    1.1
Release:    1%{?dist}
Summary:    Seq- and jot-like enumerator
License:    TODO
URL:
Source0:

%description
```

- The License tag is where we put the short license identifier (or identifiers) that reflect the license(s) of files that are built and included in this package.
- It is easiest to determine the correct license once we have an unpacked source tree, so we'll put "TODO" in this field, and fix it later.

URL

```
Name:       enum
Version:    1.1
Release:    1%{?dist}
Summary:    Seq- and jot-like enumerator
License:    TODO
URL:        https://fedorahosted.org/enum
Source0:

%description
```

- The URL tag is a link to the software homepage.

Source0

```
Name:       enum
Version:    1.1
Release:    1%{?dist}
Summary:    Seq- and jot-like enumerator
License:    TODO
URL:        https://fedorahosted.org/enum
Source0:    https://fedorahosted.org/releases/e/n/enum/%{name}-%{version}.tar.bz2

%description
```

- The Source0 tag tells the rpm what source file to use. You can have multiple `SourceN` entries, if you need them. Put the full upstream URL where you downloaded the file.

ℹ️ `rpmbuild` takes the basename. Full URL is mostly for humans.

Wait, why did you use macros there?

- You should have noticed that I used `%{name}` and `%{version}` in the Source0 URL.
- `%{name}` evaluates to whatever we have set as Name.
- `%{version}` evaluates to whatever we have set as Version.
- By doing this, it means that we should not have to change the Source0 line as new versions release (or if upstream changes the name).
- In fact, all of the fields in the preamble are defined as macros, in the exact same way!
- In your spec files, you should try to use these macros whenever possible.

Copy sources to `~/rpmbuild`

ℹ️ This is a great time to copy (not unpack) the source tarball and patch into `~/rpmbuild/SOURCES/`

```
cp materials/enum-1.1.tar.bz2 ~/rpmbuild/SOURCES/
cp materials/enum-1.1-use-putchar.patch ~/rpmbuild/SOURCES/
```

%description

- `%description` indicates to RPM that you are entering a block of text which describes the package.
- This can be multiple lines, but should be concise and describe the functionality of the package.
- No line in the `%description` can be longer than 80 characters and it must end with a period.
- Try not to simply repeat the summary.

```
%description
Utility enum enumerates values (numbers) between two values, possibly
further adjusted by a step and/or a count, all given on the command line.
Before printing, values are passed through a formatter. Very fine control
over input interpretation and output is possible.
```

Complete Preamble

```
Name:       enum
Version:    1.1
Release:    1%{?dist}
Summary:    Seq- and jot-like enumerator
License:    TODO
URL:        https://fedorahosted.org/enum
Source0:    https://fedorahosted.org/releases/e/n/enum/%{name}-%{version}.tar.bz2

%description
Utility enum enumerates values (numbers) between two values, possibly
further adjusted by a step and/or a count, all given on the command line.
Before printing, values are passed through a formatter. Very fine control
over input interpretation and output is possible.
```

Preamble: Explicit Requires

`Requires` – This lists any packages which we know are necessary to be present on the system to **run** the software in our package, once it is installed.

- RPM usually does a very good job of autodetecting dependencies and adding them for you, especially when the software is in C, C++, Perl, etc.
- Think twice if you need to be explicit.
- Requires can be versioned:

```
Requires: bar >= 2.0
Requires: bar < 10.0.0
```

Preamble: BuildRequires

- `BuildRequires` – This lists the packages which need to be present to build the software.
- Most packages have at least one BuildRequires:

```
BuildRequires: make
```

- We will see later if our package needs some BuildRequires.
- You can list as many packages as BuildRequires as you need, although, you should try to avoid redundant items. Also, BuildRequires can be versioned:

```
BuildRequires: bar >= 2.0
```

Preamble: `Patch<N>`

`Patch0` – If you need to apply a patch to the software being packaged, you can add a numbered patch entry here:

```
Patch0:     enum-1.1-use-putchar.patch
```

- Patch can be a full URL as well (like `Source<N>`), if it makes sense.

`%if` condition

How to handle different OSes with one spec? Use macros and conditions:

```
## BUGs!
%if 0%{?rhel} <= 6 || 0%{?fedora} < 17
Requires: ruby(abi) = 1.8
%else
Requires: ruby(release)
%endif
```

Preamble: Explicit Provides

- `Provides:` tells rpm that this package provides something else.
  - E.g. Httpd `Provides: webserver`, nginx too, and some package `Requires: webserver`.

```
Provides: webserver
```

- Note: RPM 4.13+ boolean dependencies:

```
Requires: (pkgA or pkgB or pkgC)
```

Objectives — real spec — continued

In this chapter you will:

- Learn about main sections of SPEC file: prep, build, install, files and changelog
- Learn about licenses
- Learn about scriptlets
- How to build the binary package from SPEC

Understanding the `%prep` section (setup)

- Source tree is generated in `~/rpmbuild/BUILD`
- Sources unpacked here
- Patches applied
- Any pre-build actions

Example of a `%setup` stage:

```
%prep
%setup -q
%patch 0 -p1
```

✅ Modern alternative: `%autosetup -p1`

Prep & Setup

- First, we need to add a `%prep` line to tell rpm that we're in the %prep phase.
- `%setup` is a very powerful (and complicated) "macro" (builtin) that is included with RPM. It is used to unpack `Source#` files, into `~/rpmbuild/BUILD/`

So, below our `%description` text, we'll add:

```
%prep
%setup -q
```

- The `-q` option tells `%setup` to unpack the Source file quietly. If you want to see what is happening here, you can omit it, but it usually is a good thing to keep the build logs small and easy to read.
- By default, `%setup` unpacks `Source0` only. It is possible to use `%setup` to unpack multiple `Source#` files at once, for more details on complicated use cases, see Maximum RPM docs.

Prep: Other items

- `%patch 0` – let's specify `Patch0` entry in our spec, and apply it here with the matching `%patch N` macro. Some common options that are good to know:
  - `-p#` — the patch level (how many directories deep does this patch apply)
  - `-b .foo` — a patch suffix, appended to the original files before patching. This is very useful when you need to update or change a patch.
- So, for example, a spec with:

```
Patch0: enum-1.1-use-putchar.patch
```

would also need:

```
%patch 0 -p0 -b .use-putchar
```

⚠️ Note that `%patch` uses `--fuzz 0`!

fedora

%prep

Here's what our spec looks like now (try `rpmbuild -bp enum.spec`):

```
Name:       enum
Version:    1.1
Release:    1%{?dist}
Summary:    Seq- and jot-like enumerator
License:    TODO
URL:        https://fedorahosted.org/enum
Source0:    https://fedorahosted.org/releases/e/n/enum/%{name}-%{version}.tar.bz2
Patch0:     enum-1.1-use-putchar.patch

%description
Utility enum enumerates values (numbers) between two values, possibly
further adjusted by a step and/or a count, all given on the command line.
Before printing, values are passed through a formatter. Very fine control
over input interpretation and output is possible.

%prep
%autosetup -p1
```

Understanding the Spec: Build

- Binary components created ("compiled") within `~/rpmbuild/BUILD`
- Use the included `%configure` macro for good defaults
- If your package uses "scons", "cmake", alter accordingly (we'll show later how to find out).

Example of a `%build` section:

```
%build
%configure
%make_build
```

Build

Here's what our spec looks like now.

```
...
%prep
%autosetup -p1

%build
%configure
%make_build
```

✅ Command `rpmbuild -bc enum.spec` should work now, fix bugs!

Understanding the Spec: Install

- Creates buildroot, `~/rpmbuild/BUILDROOT` — a directory where all build artifacts are installed on the final path. This can include all non-compiled files which should be in final package: documentation, examples, man pages, data files.
- Lays out filesystem structure:

```
mkdir -p %{buildroot}/var
mkdir -p %{buildroot}/%{_bindir}
```

- Puts built files in buildroot:

```
cp -a result-of-make/my-application %{buildroot}/%{_bindir}/my-application
```

- Cleans up unwanted installed files (if needed):

```
%make_install          # installs too much stuff
rm -rf %{buildroot}/%{_datadir}/non-free/
```

Fixing our Install Section

- Our build section is done now. Now, we need to fix up our `%install` section. We know that we need to use "`make install`" to install... try it now.
- ... but RPM doesn't want to install the files into their positions on `/`. We need to install the files into our BuildRoot, so that RPM can collect them and package them up in the binary rpm file.
- Here's how you do this. Add this line below `%install`:

```
make DESTDIR=%{buildroot} install    # old variant
```

- `%make_install` — modern macro equivalent
- That's all! The `DESTDIR` variable tells make to install into our `%{buildroot}`.
- Some sloppy software Makefiles may not support DESTDIR. If you come across one of these, you should try to add support to the Makefile. Feel free to ask on `#fedora-devel` or `devel@lists.fedoraproject.org` for help with this.

Install

```
BuildRequires: asciidoc
...

%build
%configure --disable-doc-rebuild
%make_build

%install
%make_install
```

ℹ️ Old variants (don't use anymore):

`make %{?_smp_mflags}` → use `%make_build`

`rm -rf $RPM_BUILD_ROOT` → not needed nowadays

`make install DESTDIR=%{buildroot}` → use `%make_install`

Understanding the Spec: `%files`

- `%files`: List of package contents
  - If it is not in `%files`, it is not in the package.
  - RPM WILL complain about unpackaged files.
- We'll come back to this section at the end, when we know what to put in it. For now, lets just define the section, by adding the `%files` line below our `%install` section:

```
%files
```

Understanding the Spec: Changelog

- Used to track package changes
- Not intended to replace source code Changelog
- Provides explanation for package users, audit trail
- Update on every change

Example of Changelog section:

```
%changelog
* Mon Jun 2 2008 Tom "spot" Callaway <tcallawa@redhat.com> - 1.1-3
- minor example changes

* Mon Apr 16 2007 Tom "spot" Callaway <tcallawa@redhat.com> - 1.1-2
- update example package

* Sun May 14 2006 Tom "spot" Callaway <tcallawa@redhat.com> - 1.1-1
- initial package
```

Changelog

- Below our `%files` section, as the last item in the spec, add a line for changelog:

```
%changelog
```

- Then, lets add our first changelog entry, in the proper layout:

```
* Mon Feb 23 2026 Your Name Here <youremail@here> - 1.1-1
- New package for Fedora
```

✅ You may use `rpmdev-bumpspec enum.spec` to do the most hard work for you.

Spec File Status

```
Name:       enum
Version:    1.1
Release:    1%{?dist}
Summary:    Seq- and jot-like enumerator
License:    TODO
URL:        https://fedorahosted.org/enum
Source0:    https://fedorahosted.org/releases/e/n/enum/%{name}-%{version}.tar.bz2

%description
Utility enum enumerates values (numbers) between two values, possibly
....

%prep
%autosetup -p1

%build
%configure
%make_build

%install
%make_install
```

Building our source tree

- Now, we can use that source tree to help us fill in the blanks we left earlier — `License: TODO`
- At this point, we can use our spec to build our source tree and take a look around it.
- Save your spec file, and run (as a normal user):

```
rpmbuild -bp ~/rpmbuild/SPECS/enum.spec
```

- The `-bp` option tells rpmbuild to run through the `%prep` stage, then stop.
- It should complete without errors, and you should see a new directory in `~/rpmbuild/BUILD/enum-1.1/`

Licensing

As a responsible Fedora packager, it is important that you do the licensing correct for your package.

Here are some general steps to follow:

1. Is there a `COPYING` or `LICENSE` file? If so, read it, and remember its file name, because we'll want to include it in our package.
2. Is there a `README` file? Read it, and look for any mention of "Copyright" or "License".
3. Look at the actual source files in your text editor. Some code projects will describe their license in the header comments of each source file.
4. Note all licenses that you see, then look them up in the Fedora Licensing chart.

> ℹ️ Fedora allowed licenses

What license do you find for enum?

Licensing Part Two

- `enum` is licensed under the BSD 3-Clause Licence. In Fedora, the short name identifier for this license is `BSD-3-Clause`.
- Licensing can be very complicated! When in doubt, feel free to email `fedora-legal@lists.fedoraproject.org` or `legal@fedoraproject.org` to get help.
- Now, we need to fix our spec. Open it up in a text editor again, and change the License tag from `TODO` to `BSD-3-Clause`
- Also, did you see that file `COPYING`? We need to make sure they are installed in our package, so we'll add them to our `%files` list, using a macro called `%license` (formerly `%doc` was used).
- Audit tools: `licensecheck` helper, `askalono-cli` package
- License changes must be announced on developer list
- Multiple Licensing Scenarios: `LGPL-2.0-or-later AND MIT`
- Watch out for code copied from other projects (different license may apply to that portion)

Understanding `%doc`

- `%doc` is a special macro that is used to:
  - Mark files as documentation inside an RPM (`rpm -qd tar`)
  - Copy them directly from the source tree in `~/rpmbuild/BUILD/enum-1.1/` into the "docdir" for your package, ensuring that your package always has them.
- `%license` is similar — license-related "documentation" (`rpm -qL tar`)
- So, lets add a line to `%files` for our license texts, so that it now looks like this:

```
%files
%license COPYING
%doc ChangeLog
```

- Once you're done, save your spec file again.

How Does Enum Build and Install?

- Now, we need to look in the `~/rpmbuild/BUILD/enum-1.1/` source tree to figure out how to build and install the code.
- Here's a big hint: Many Linux software projects use these commands to build:

```
./configure
make
```

And these commands to install:

```
make install
```

- Some packages will not be so clean. Look for `INSTALL` or `README` to describe it, or the project website. When in doubt, ask!

History Lessons

- Before Fedora 12 (and in RHEL 5 or older), it used to be necessary to manually remove the `%{buildroot}` as the very first step in the `%install` stage, like this:

```
%install
rm -rf %{buildroot}
```

With Fedora 12 (or newer) RPM, the `%install` stage automatically deletes the `%{buildroot}` for you as the very first step, so this is no longer necessary.

- Also, it used to be necessary to define a `%clean` section to clean up the `%{buildroot}` at the end of the package build process, it looked like this:

```
%clean
rm -rf %{buildroot}
```

With Fedora 12+ RPM, this `%clean` section is handled internally and does not need to be explicitly defined.

History Lessons #2

- In the recent past, there was a mandatory field in the Preamble statement called `BuildRoot` — it is now defined automatically (EL6+). You can drop such lines:

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

ℹ️ Don't add `BuildRoot`, `%clean`, or `rm -rf %{buildroot}` to new specs — they're obsolete.

Status

- Okay, history lesson over. Your spec should have a fleshed out `%build` and `%install` by now.
- Save your spec file out. There is one thing we're missing from our spec, the list of files to go into `%files`! I'm about to share a clever trick on how to make it easier.

Build our package

Now, as a normal user run:

```
rpmbuild -ba ~/rpmbuild/SPECS/enum.spec
```

- The `-ba` options tell rpmbuild to build "all", source and binary packages.
- This is going to run through `%build`, then `%install` ... and then **fail**, because we don't have a complete `%files` list specified.

Build our package (output)

But, when it fails, it does us a favor, check out the output!

```
RPM build errors:
    Installed (but unpackaged) file(s) found:
    /usr/bin/enum
    /usr/lib/debug/usr/bin/enum-1.1-1.fc42.x86_64.debug
    /usr/share/man/man1/enum.1.gz
```

✅ RPM has just told us what files are missing from the `%files` list!

Adding missing `%files`

```
%files
%license COPYING
%doc ChangeLog
%{_mandir}/man1/enum.1*
%{_bindir}/enum
```

We skipped the debug file, because once we added the corresponding executable then rpmbuild picks the debug file automatically.

Files

- You may also add a `%defattr` line. The `%defattr` macro tells RPM what to set the default attributes to for any and all files in the `%files` section. The Fedora default is `(-,root,root,-)`. You do not need to define it again.

```
%attr(<file mode>, <user>, <group>, <dir mode>)
```

ℹ️ http://ftp.rpm.org/max-rpm/s1-rpm-specref-files-list-directives.html

Almost done

- At this point, make sure you have all useful documentation listed in `%files` as `%doc`, not just the license text. Look for `README` and `ChangeLog`.
- `INSTALL` is usually not very helpful, do not install it.
- That should be it! Look over your spec and make sure you're happy with it, then save it, and run:

```
rpmbuild -ba ~/rpmbuild/SPECS/enum.spec
```

Status Check

- At this point, you should now have a `enum-1.1-1.src.rpm` and one arch rpm: `enum-1.1-1.fc*.*.rpm` (the Fedora versions and architectures will vary, depending on your system).
- If so, congratulations! You're on your way to really understanding how to make good Fedora packages!
- If it failed, no worries. Take a look at the last few lines that RPM output, and it will probably give you an idea of what to fix. Feel free to ask for help.
- The next step is to check the package for minor issues, and you use a tool called `rpmlint` for this. Run:

```
rpmlint ~/rpmbuild/SRPMS/enum-1.1-1.src.rpm ~/rpmbuild/RPMS/*/enum*.rpm
```

- If you find any errors, try to correct them in the spec and rebuild. If you do make changes to your spec file, increment your Release and add a new changelog entry at the top of your `%changelog` section!

Bonus — RPM Scriptlets

- Shell code sections executed at certain times during package installation: `%pre`, `%post`, `%preun`, `%postun`
- Scriptlets are non-interactive, no arguments, macros are expanded at package build time and baked-into the RPM file
- Remember, RPM DB doesn't track what is done here
- The best intentions can easily go evil (it is hard to get scriptlets right), but sometimes are necessary (but distribution development is trying to convert them to a declarative format, like with system-users creation)
- More info: RPM packaging guide and Fedora guidelines.

Best Practices

- **K.I.S.S.** — Keep It Simple
- Use patches, not rpm hacks
- Avoid scriptlets, minimize wherever possible
  - Leverage triggers instead (fedora docs)
- Use `%changelog`
- Look at and learn from other Fedora packages
  - Huge tarball with all spec files
- Use macros sanely
  - Be consistent
  - Utilize system macros

Do not build a package as root

⚠️ Do **NOT EVER** build RPMS as root.
Let me repeat, do **NOT EVER** build RPMS as root.

- Ideally run under a separate build-specific user (use mock ideally)

Why? Look at this:

```
%install
%make_install
# remove patent protected files
rm -rf $RPM_TYPO_BUILDROO/var/
```

⚠️ **THIS WILL REMOVE `/var` on your system!** Why? Because the typo in the variable name makes it evaluate to empty string, so it becomes `rm -rf /var/`

Better than Best Practices

- Use `rpmlint`, fix warnings/errors
- Include configs/scripts as `Source` files
- Comment!
  - ...but keep it legible
  - Think of the guy who will have to fix your package when you leave.
- Don't ever call `/bin/rpm*` from inside a spec file.
  - RPM is NOT re-entrant

Good Packages Put You In Control

Practice

Practice makes perfect

Integration

Integration with the Fedora tools makes it easier for users to get and use that software!

Simplify

Simplify, standardize, save time and sanity.
Build once, install many.

Useful Links

- 📖 **Fedora Packaging Guidelines**
- 📋 **Review Guidelines**
- 📚 **RPM spec file reference**
- 📘 **RPM Packaging Guide**
- 💻 **Fedora dist-git** — contains lots of example specs
- ✉️ **Fedora packaging mailing list**
- 🔍 **rpmlint on GitHub**
- 📄 **RPM.org documentation**

Q&A

❓

Happy packaging! 📦

Packaging Guidelines • dist-git • RPM Packaging Guide

#fedora-devel (Matrix) · devel@lists.fedoraproject.org