# Workshop: Building Good Fedora Packages

Pavel Raiskup
<praiskup@redhat.com>

Miroslav Suchý
<msuchy@redhat.com>

Slides originally created by
Tom 'spot' Callaway

## Workshop Overview

- Topic
  - intro to RPM, create a simple RPM package from scratch
- Assumptions
  - You know how to manually build “normal” software for Linux
  - You know how to use a text editor (I don't care which one)
  - You have some familiarity with installing RPM packages
- Limitations
  - This workshop covers a simple piece of software
  - Most packages will be a little more complicated
    - Some will be a LOT more complicated
- Information
  - Feel free to ask question.  Break?

## Things to do to prepare your environment

- Packages that you will need installed
  - fedora-packager, rpm-build, dnf (or yum on RHEL <8), rpmdevtools, rpmlint, patch

- Get a copy of the tarball of source code we will be packaging:
- https://praiskup.fedorapeople.org/courses/packaging/

## The Importance of Packaging

- Why when we have kube, podman, docker, etc.?
- Standards compliance
  - Know what is present, rpm -qa
- Simplify environment
  - Know how to find it, rpm -qf
- Standardize deployments
  - Know that you installed it, dnf history
- Sanity retention
  - Know where it is installed, rpm -ql
- Reproducible
  - Know how it was built

## Fedora Packaging Guidelines

- Intended to document a set of “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/

## A Quick Primer on RPM

- Red Hat Enterprise Linux and Fedora use RPM (formerly Red Hat Package Manager)
- Database driven solution
- Dependency tracking
- Built-in package verification, rpm -V

## Myths about RPM

- Doesn't work well
- Hard to create packages
- Hard to install packages
- Hard to remove packages
- Dependency Nightmares
  - Dependency Hell

## Don't Slay The Dragon!

- RPM is misunderstood
- Works extremely well
- Package creation is easier than you think
- Easy to install...
- Easy to remove...
- ... with good packages!

## Dependency Resolution: dnf/yum

- RPM Pain Point: Dependency resolution
  - Dependencies make RPM useful but also complicated.
- RHEL/Fedora use yum/dnf to ease the pain
- Metadata is generated from tree of RPM packages
- Yum/DNF uses metadata to resolve dependencies
- Support for plugins, history, rollbacks
- Used to enable preupgrade, anaconda
- DNF use satsolver

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

- Auditing software usage
  - What, where, when?
- Version control, rpm -q <pkg>
- Kickstart integration (anaconda)
- Minimizes risk
  - Security
  - Rogue Applications
  - Licensing
  - Trusted provider

## Common mistakes new packagers make

- Spec file generators
  - Remember, functional is not the same as good.
- Packaging pre-built binaries, not building from source.
  - Not always possible, but you shouldn't start here if you can help it.
  - Not permitted in Fedora
- Disabling check for unpackaged files
  - This is a recipe for disaster.

## Crash course in RPM Usage

- Binary Package (pkg-workshop-1-0.x86_64.rpm)
  - File name is <u>different</u> from package name
- Install packages with <u>file name</u>
  - rpm -ivh pkg-workshop-1-0.x86_64.rpm
  - i for install, v for verbose, h for process hash
- Query installed package with <u>package name</u>
  - rpm -ql pkg-workshop
  - q for query, l for list files
- Remove package with <u>package name</u>
  - rpm -eh pkg-workshop
  - e for erase

fedora

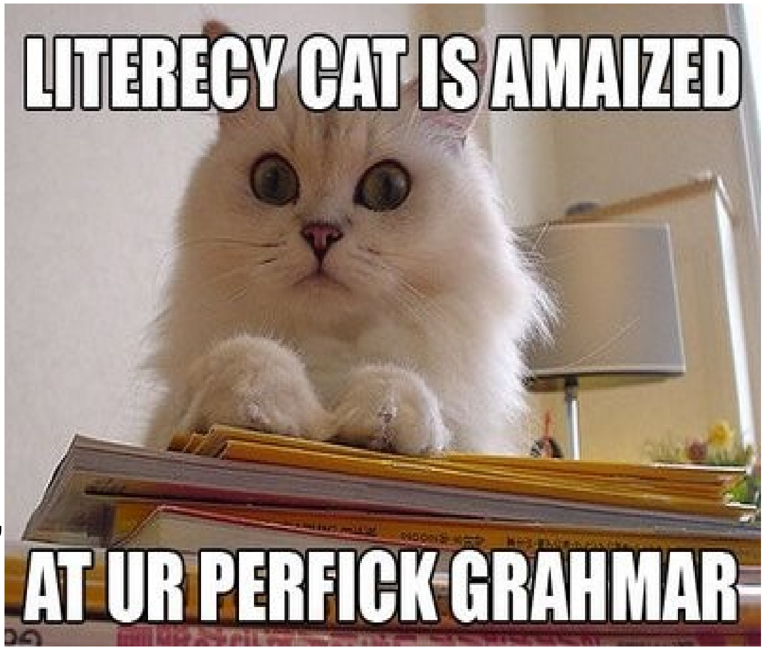
## Source RPM Overview

- Source Package (pkg-workshop-1-0.src.rpm)
  - SRPMs contain sources/components/spec file used to generate binary RPM packages
- Install SRPM package with <u>SRPM file name</u>
  - rpm -ivh pkg-workshop-1-0.src.rpm
  - i for install, v for verbose, h for process hash
  - 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 <u>spec file name</u>
  - 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
  - rpmbuild -ba pkg-workshop.spec
  - -b for build, -a for all packages, src and bin
- Making a patched source tree from spec file
  - rpmbuild -bp goldfish.spec
  - -b for build, -p for %prep only
- Patched source trees go into the builddir
  - Red Hat default is ~/rpmbuild/BUILD

## RPM Macros



- Just like variables in shell scripting
  - They can act as integers or strings, but its easier to always treat them as strings.
- Many common macros come predefined
  - **rpm --showrc** will show you all the defined macros, number explanation
  - **rpm --eval %macroname** will show you what a specific macro evaluates to
  - Most system macros begin with an _ (e.g. %{_bindir})
- Macro formats
  - %{foo} and %foo
  - some macros have shell 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.

## RPM Comments

- To add a comment to your RPM spec file, simply **start** a new line with a # symbol. Feel free to do this as we go, to take notes for yourself. It never hurts to explain in a comment why you did something, and it may save a bit of your sanity later on.
- For 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**, sometimes if you # comment out a macro definition, it will see it and evaluate it anyways. To comment out a macro definition, use two %% instead of just one:

    Before: # %configure
    After: # %%configure

## ~/.rpmmacros : Use it or else

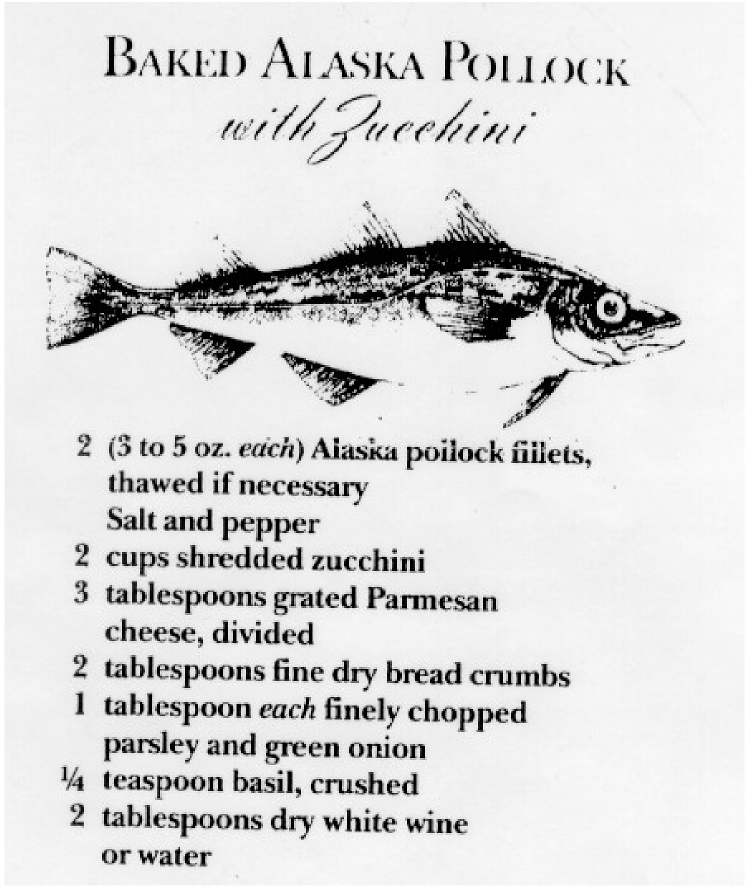
- Do it now. You'll thank me later. So will the kittens.
- Having an ~/.rpmmacros file enables custom macros for your UID.
  - **Do NOT EVER build RPMS as root.
    Let me repeat, do NOT EVER build RPMS as root.**
- Make a rpmbuild tree in your home directory:
  mkdir -p ~/rpmbuild{BUILD,BUILDROOT,RPMS,SOURCES,SPECS,SRPMS}
  mkdir -p ~/rpmbuild/RPMS/{noarch,i386,i686}
  - On Fedora, you can use the “rpmdev-setuptree” command from the “rpmdevtools” package to accomplish the above steps.
- This is a great time to copy (not unpack) the enum-1.1.tar.bz2 source into ~/rpmbuild/SOURCES/
- You can make your own macros here, but be careful!  Why?
  - Accordingly, don't use them in %pre/%post.

## Useful items for your ~/.rpmmacros

- Now would be a great time to open your text editor and add the following two lines to ~/.rpmmacros
- %_smp_mflags    -j3
  - If your package SPEC has “make %{?_smp_mflags}, then this will tell it to try to build across three CPUs.
  - Why three? Three is a nice odd number that isn't too harsh for uniprocessor systems but enough to expose code that doesn't build well in SMP environments.
- %__arch_install_post /usr/lib/rpm/check-rpaths && /usr/lib/rpm/check-buildroot
  - Fedora has an rpmdevtools package full of, well, rpm development tools.
  - check-rpaths will make sure that your package doesn't have any hardcoded rpaths (a bad thing, see guidelines)
  - check-buildroot will make sure none of the packaged files have the buildroot hardcoded (also a bad thing)
  - other BRP scripts (BuildRoot Policy)

## Cooking with Spec Files

- Think of a spec file as a recipe
- Lists the contents of the RPMS
- Describes the process to build, install the sources
- Required to make packages
- Very similar to shell script
- Sections/stages:
  - Preamble
  - Setup
  - Build
  - Install
  - Clean
  - Files
  - Changelog



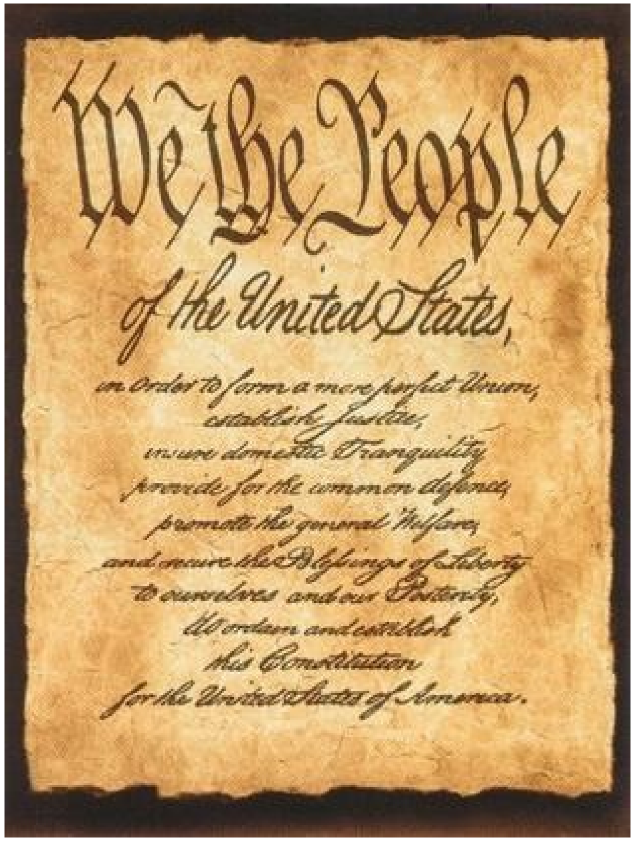
BAKED ALASKA POLLOCK

*with Zucchini*

- 2 (3 to 5 oz. *each*) Alaska pollock fillets, thawed if necessary
- Salt and pepper
- 2 cups shredded zucchini
- 3 tablespoons grated Parmesan cheese, divided
- 2 tablespoons fine dry bread crumbs
- 1 tablespoon *each* finely chopped parsley and green onion
- ¼ teaspoon basil, crushed
- 2 tablespoons dry white wine or water

## Understanding the Spec File: Preamble

- Initial section
- mostly metadata (no scripts)
- try `rpm -qi tar`
- Defines package characteristics
  - Name/Version/Group/License
  - Release tracks build changes
  - Sources/Patches
  - Requirements
    - Build & Install
  - Summary/Description
  - Custom macro definitions



We the People
of the United States,
in Order to form a more perfect Union,
establish Justice,
insure domestic Tranquility,
provide for the common defence,
promote the general Welfare,
and secure the Blessings of Liberty
to ourselves and our Posterity,
do ordain and establish
this Constitution
for the United States of America.

## Notes on modern RPM : Preamble

- In the recent past, there was a mandatory field in the Preamble section called “BuildRoot”
- This field defined the folder where the installed files would go, before being placed into the final RPM
- It looked like this (in Fedora):
  BuildRoot: %{_tmppath}/%{name}-%{version}-%{release}-root-%(%{__id_u} -n)
- As of the RPM in Fedora 12 (and RHEL 6), the BuildRoot field is no longer necessary. %buildroot is predefined by RPM.
  - Older versions of RPM, most notably, the one in RHEL 5, still require this field.
- RPM now auto-sets the BuildRoot for each package to:
- ~/rpmbuild/BUILDROOT/%{name}-%{version}-%{release}.%{_build_arch}
- %_sourcedir => %_builddir => %buildroot => *.rpm

## Workshop: Creating a new spec file

- A spec file is simply a text file
- 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
- Some editors (notably, vim, before rhbz#1724126, now it’s in /usr/share/vim/vimfiles/template.spec) will generate a spec template when you open a new spec file.
- rpmdev-newspec, (rhbz#1724126)

## Workshop : Preamble Items

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

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

%description
```

## Workshop : 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.

## Workshop : Version

Next, lets fill in the version of our package, which is “1.1”.

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

%description
```

- Hopefully, it should be obvious where I got the version from. :) Sane upstreams will put the version in the source tarball name. Other code may require digging.

## Workshop : Release

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

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

%description
```

- 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/
- Use /bin/rpmdev-vercmp if you are in doubt!

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

- It is a macro!
- This macro exists in Fedora and RHEL 5+.
- It is there to add an identifier (or dist) tag to the end of the release field.
- 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.fc13” on Fedora 13, and “1.el5” on RHEL 5.
- You can see more information on the Dist Tag here:

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

## Workshop : Summary

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

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

%description
```

- NOTE: We'll fill in a longer description of the package in %description
- rpm -qa --qf "%{SUMMARY}\n" | grep \\.$

## Workshop : License

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.

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

%description
```

## Workshop : URL

The URL tag is a link to the software homepage.

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

%description
```

## Workshop : Source0

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

- 
```
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
```

- RPM is smart enough to know that a URL in the source path is different from the file we downloaded and put in ~/rpmbuild/SOURCES/. It will look for the filename, minus the URL.

## Wait, why did you use macros there?

- You should have noticed that I used %{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.

## Workshop : %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.

## Workshop : Complete Preamble

```
Name: enum
Version: 1.1
Release: 2%{?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: Other items

- 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
  ```

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

  ```
  BuildRequires:  gcc  (previously in minimal build chroot)
  ```

  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: 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++, or Perl.
  - Be careful about adding explicit Requires here, as most packages will not need any.
  - You can add versioned Requires as needed, in exactly the same way as versioned BuildRequires are done.

## %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: tell rpm that this package provides something else. E.g. Httpd provides webserver
- Note: boolean dependencies

## Understanding the Spec: Setup

- Source tree is generated
- Sources unpacked here
- Patches applied
- Any pre-build actions
- Example of a %setup stage:

```
%prep
%setup -q
%patch0 -p1
```

- modern alternative: %autosetup -p1

## Workshop : 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” 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

- %patch0 – If we had a Patch0 entry in our spec, we would apply it here with the matching %patch# 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: foo-1.2.3-fixbugs.patch
  would also need:
  %patch0 -p0 -b .fixbugs
- Note that patch patch use fuzzy 0!

## Workshop : %prep

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

```
Name: enum
Version: 1.1
Release: 2%{?dist}
Summary: Seq- and jot-like enumerator
Group:   Applications/System
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.

%prep
%setup -q
```

## Understanding the Spec: Build

- Binary components created
- Use the included %configure macro for good defaults
- Just build binary bits in sourcedir (no binary rpms):
  rpmbuild -bc helloworld.spec
  - -b for build, -c for compile and stop
- Example of a %build section

  %build
  %configure
  ~~make %{?_smp_mflags}~~

  %make_build
- If your package uses scons, cmake, alter accordingly.

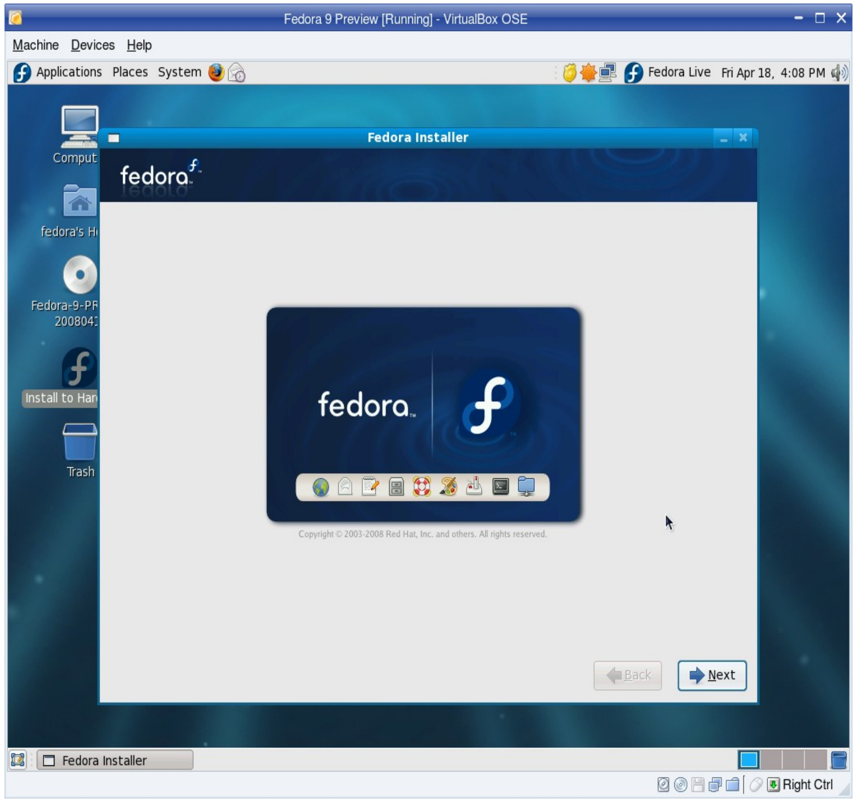
## Workshop : Build

- Here's what our spec looks like now.
- 
- ...
- %prep
- %setup -q
- 
- 
- %build
- %configure
- ~~make %%{?_smp_mflags}~~
- %make_build

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

## Understanding the Spec: Install

- Creates buildroot (see %buildroot)
- Lays out filesystem structure
- Puts built files in buildroot
- Cleans up unnecessary installed files


Fedora 9 Preview [Running] - VirtualBox OSE
Machine Devices Help
Applications Places System
Fedora Live Fri Apr 18, 4:08 PM
Fedora Installer
fedora
Back
Next
Fedora Installer
Right Ctrl

## Workshop – 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~~

  %make_install

- 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.

## Workshop : Install

BuildRequires: asciidoc

- ...
- %build
- %configure ~~--disable-doc-rebuild~~
- ~~make %{?_smp_mflags}~~
- %make_build
- 
- %install
- ~~rm -rf $RPM_BUILD_ROOT~~
- ~~make install DESTDIR=%{buildroot}~~
- %make_install

## Understanding the Spec: Files

- Files: List of package contents
  - If it is not in %files, it is not in the package.
  - RPM <u>WILL</u> complain about unpackaged files.
    - Please, please, please. Don't ever hack around this and generate files in %post.

## Workshop : Files

- We'll come back to this section at the end, when we know what to put in it. I'll show you a trick to make it easier. For now, lets just define the section, by adding the %files line below our %install section:

  %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 will almost never need to change this default, so you do not need to use it.

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

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

## 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
```

## Workshop : 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:

  ```
  * Fri Sep 21 2012 Your Name Here <youremail@here> - 1.1-1
  - New package for Fedora
  ```

## Workshop – Spec File Status

```
Name: enum
Version: 1.1
Release: 2%{?dist}
Summary: Seq- and jot-like enumerator
Group:   Applications/System
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
%setup -q

%build
%configure
%make_build
```

## Workshop – Spec File Status

```
%install
%make_install

%files

%changelog
* Fri Sep 21 2012 Your Name Here <youremail@here> - 1.1-1
- New package for Fedora
```

## Workshop – Building our source tree

- 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/
- Now, we can use that source tree to help us fill in the blanks we left earlier.

## Workshop - Licensing

- As a responsible Fedora packager, it is important that you get the licensing correct for your package. Here are some general steps to follow:
  - 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.
  - Is there a README file? Read it, and look for any mention of “Copyright” or “License”.
  - Look at the actual source files in your text editor. Good code projects will describe their license in the header comments of each source file.
  - Note all licenses that you see, then look them up in the Fedora Licensing chart, found here: https://fedoraproject.org/wiki/Licensing#Software_License_List
  - What license do you find for enum?

## Workshop – Licensing Part Two

- enum is licensed under the BSD Licence. In Fedora, the short name identifier for this license is BSD.
- 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
- 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 “%doc” (in Fedora newly %license).

## Workshop – Understanding %doc

- %doc is a special macro that is used to:
  - Mark files as documentation inside an RPM
  - 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.
- So, lets add a line to %files for our license texts, so that it now looks like this:

  %files
  ~~%doc COPYING~~
  %license COPYING
  %doc ChangeLog
- Once you're done, save your spec file again.

## Workshop – 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: Most 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.

## Workshop - Status

- Okay, history lesson over. Your spec should have a fleshed out %build and %install, which look like this:

  see konsole
-
- 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.

## Workshop – Build our package

- Now, as a normal user (remember, no package building as root, EVER), 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.
-

## Workshop – Build our package

- 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/share/man/man1/enum.1.gz
- 
- RPM has just told us what files are missing from the %files list!

## Workshop – Adding missing %files

- %files
- %license COPYING
- %doc ChangeLog
- %_mandir/man1/enum.1*
- %_bindir/enum

## Workshop – Almost done

- At this point, make sure you have all useful documentation listed in %files as %doc, not just 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
```

## Workshop – 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 me 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 Version and add a new changelog entry at the top of your %changelog section!

## Best Practices

- K.I.S.S.
- Use patches, not rpm hacks
- Avoid scriptlets, minimize pre/post wherever p
  - Leverage triggers instead (fedora docs)
- Use %changelog
- Look at other Fedora packages
  - Huge tarball with all spec files
- Use macros sanely
  - Be consistent
  - Utilize system macros

## 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.
  - Remember Ghostbusters? Crossing the streams? Bad.

## Good Packages Put You In Control

- Practice makes perfect
- Integration with the Fedora tools makes it easier for users to get and use that software!
- Simplify, standardize, save time and sanity
  - Build once, install many.

## Useful Links

- Fedora Packaging Guidelines:
  https://docs.fedoraproject.org/en-US/packaging-guidelines/
  https://fedoraproject.org/wiki/Packaging:ReviewGuidelines
- Maximum RPM:
  http://www.rpm.org/max-rpm-snapshot/
- RPM Packaging Guide:
  https://rpm-packaging-guide.github.io/
- Fedora GIT Tree (contains lots of example specs)
  https://src.fedoraproject.org/
- Fedora packaging mailing list
  https://admin.fedoraproject.org/mailman/listinfo/packaging
- Rpmlint website:
  http://rpmlint.zarb.org/cgi-bin/trac.cgi
- Czech only, Rukověť baliče RPM:
  http://www.abclinuxu.cz/clanky/navody/rukovet-balice-rpm-i-uvod