How to make good RPM Packages

Tom Callaway

Fedora Engineering Manager, Red Hat

About the Presenter

- Fedora Engineering Manager
- Primary Author of Fedora Packaging Guidelines
- Maintainer for 250+ Fedora packages
- Chair of Fedora Packaging Committee
- Fedora/SPARC Architecture Team Lead
- Wild and crazy guy

The Importance of Good Packaging

- Standardize deployments
  - Know that you installed it
- Simplify environment
  - Know how to find it
- Standards compliance
  - Know what is present
- Sanity retention
  - Know where it is

A Quick Primer on RPM

- Red Hat Enterprise Linux and Fedora use RPM
- RPM Format is Linux Standard (LSB)
- Database driven solution
- Dependency tracking
- Built-in package verification

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: yum

- RPM Pain Point: Dependency resolution
  - Dependencies make RPM useful but also complicated.
- RHEL 5/Fedora use yum to ease the pain
- Metadata is generated from tree of RPM packages
- Yum uses metadata to resolve dependencies
- Only install what you need, remove what you don't need anymore
- Red Hat Network uses yum (RHEL 5)

Packaging as a standard

- Auditing software usage
  - What, where, when?
- Version control
- Kickstart integration
- Minimizes risk
  - Security
  - Rogue Applications
  - Licensing

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.
- Disabling check for unpackaged files
  - This is a recipe for disaster.

Crash course in RPM Usage

- Binary Package (goldfish-1.0.0-1.i386.rpm)
  - File name is different from package name
- Install packages with file name
  - rpm -ivh goldfish-1.0.0-1.i386.rpm
  - i for install, v for verbose, h for process hash
- Query installed package with package name
  - rpm -ql goldfish
  - q for query, l for list files
- Remove package with package name
  - rpm -e goldfish
  - e for erase

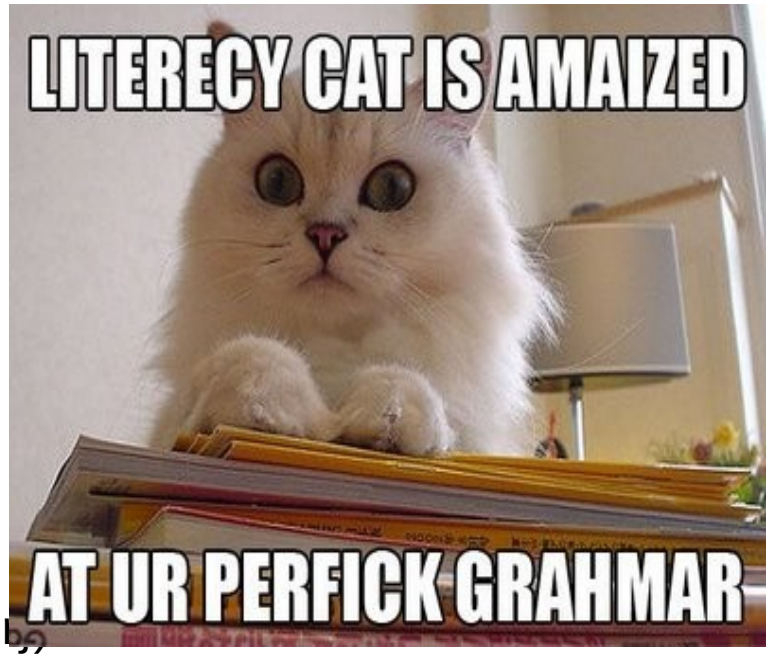
Source RPM Overview

- Source Package (goldfish-1.0.0-1.src.rpm)
  - SRPMs contain sources/components/spec file used to generate binary RPM packages
- Install SRPM package with SRPM file name
  - rpm -ivh goldfish-1.0.0-1.src.rpm
  - i for install, v for verbose, h for process hash
  - Source packages just install source into defined source directory
    - Red Hat default: /usr/src/redhat/SOURCES
- SRPMs do not go into the RPM database
- Remove installed SRPM with spec file name
  - rpmbuild --rmsource --rmspec goldfish.spec

More Source RPM Overview

- Making a binary rpm from SRPM:
  - rpmbuild --rebuild goldfish-1.0.0-1.src.rpm
- Making a binary rpm from spec file
  - rpmbuild -ba goldfish.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 patch only
- Patched source trees go into the builddir
  - Red Hat default is /usr/src/redhat/BUILD
- Building for a different architecture
  - rpmbuild –target sparc –rebuild goldfish-1.0.0-1.src.rpm
  - Be careful! Many software programs detect the system type in configure and ignore what rpm told it.

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
  - rpm --eval %{macroname} will show you what a specific macro evaluates to
  - Most system macros begin with an _ (e.g. %{_bindir})
- Two macro formats
  - %{foo} vs $foo
  - They are the same macro, but for your sanity, you should consistently use one type of macro in a spec file.
  - I recommend using %{foo} style macros
    - %{?foo} (if foo is defined, use it, otherwise, it does not get used).

~/.rpmmacros : Use it or else

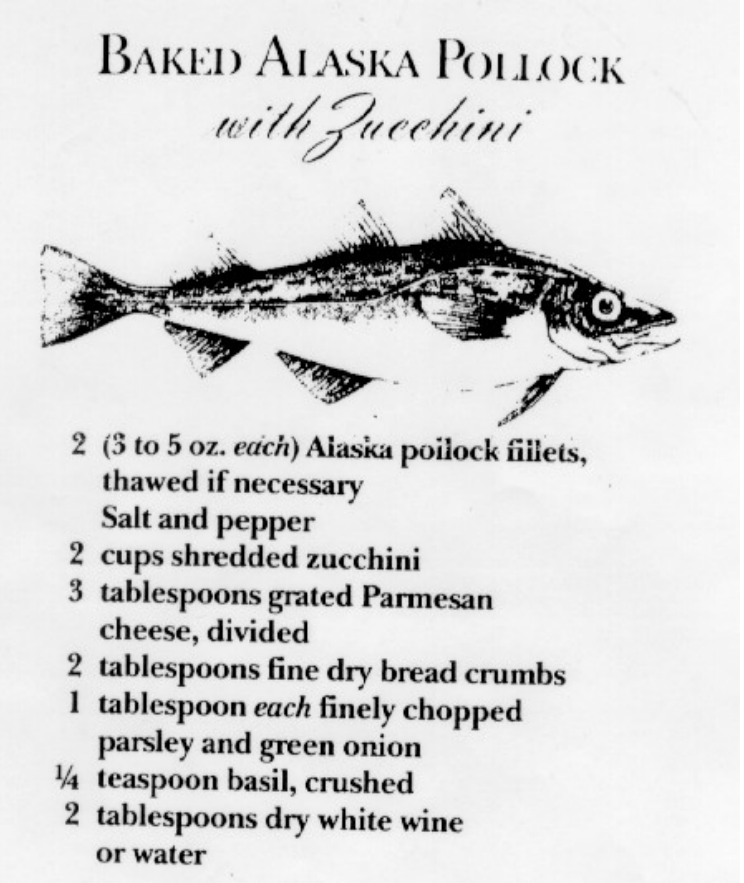
- Do it now. You'll thank me later. So will the kittens.
- Having an ~/.rpmmacros file enables custom macros for your user.
  - **Do NOT EVER build RPMS as root.**
    Let me repeat, do NOT EVER build RPMS as root.
- Make a rpmbuild tree in your home directory:
  mkdir -p ~/rpmbuild/{BUILD,RPMS,SOURCES,SPECS,SRPMS}
  mkdir -p ~/rpmbuild/RPMS/{noarch,i386,i686}
- Make an ~/.rpmmacros file:
  %_topdir %(echo $HOME)/rpmbuild
- You can make your own macros here, but be careful!
  - Custom macros will almost certainly not be on the system where you install your built packages.
  - Accordingly, don't use them in %pre/%post.

Other useful items for your ~/.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)
  - check-buildroot will make sure none of the packaged files have the buildroot hardcoded (also a bad thing)

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
- Stages:
  - Preamble
  - Setup
  - Build
  - Install
  - Clean
  - Files
  - Changelog

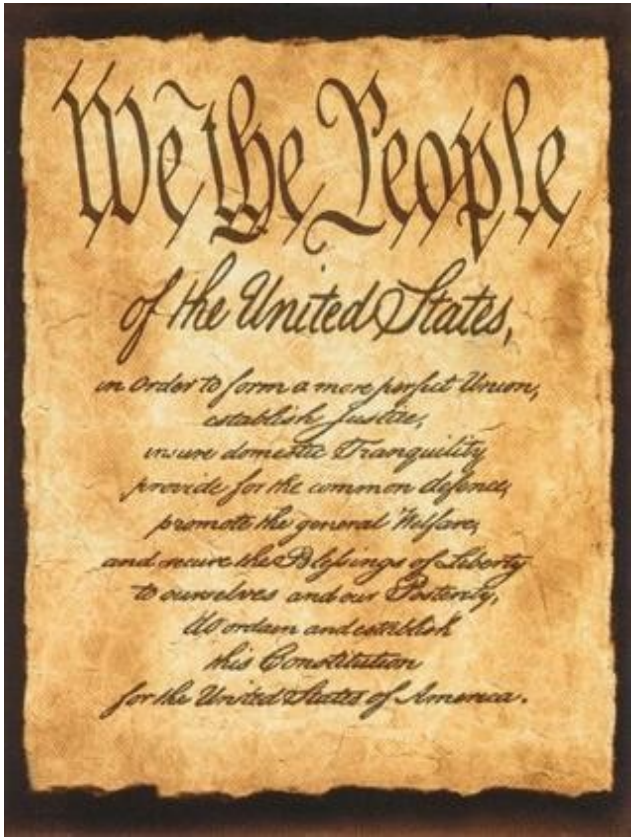


BAKED ALASKA POLLOCK
with Zucchini

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
- Defines package characteristics
  - Name/Version/Group/License
  - Release tracks build changes
  - Sources/Patches
  - Requirements
    - Build & Install
  - Summary/Description
  - Custom macro definitions



Preamble Example

Name:	helloworld
Version:	1.1
Release:	3
Summary:	An application that prints “Hello World!”
License:	GPLv2+
Group:	System Environment/Base
Source0:	http://helloworld.com/helloworld-1.1.tar.gz
Patch0:	fixtypo.patch
BuildRoot:	%{_tmppath}/%{name}-%{version}-%{release}-root-%(%{__id_u} -n)
BuildArch:	noarch

%description

This program prints hello world on the screen to avoid the “programmers curse”. The Programmmers Curse states that unless your first example is “Hello World!”, then you will be cursed, and never able to use that tool.

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
```

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

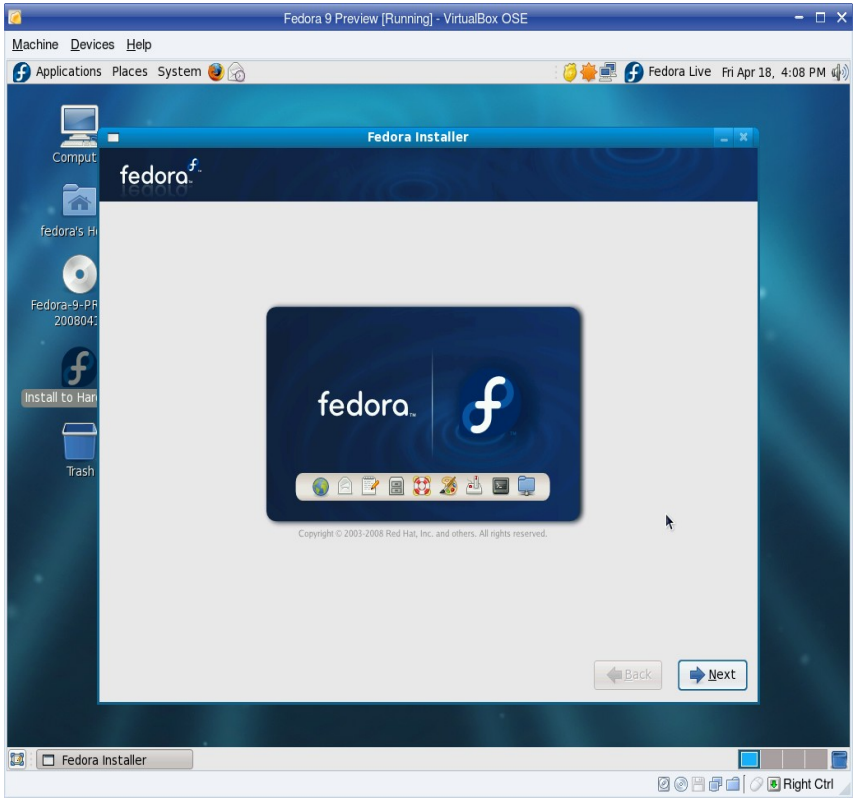
  %build
  %configure
  make %{?_smp_mflags}
- If your package uses scons, cmake, alter accordingly.

Understanding the Spec: Install

- Creates buildroot
- Lays out filesystem structure
- Puts built files in buildroot
- Cleans up unnecessary installed files
- Example of an %install section

```
%install
rm -rf %{buildroot}
mkdir -p %{buildroot}%{_bindir}
cp helloworld.sh %{buildroot}%{_bindir}/helloworld
```

- %{_bindir} ?
  - That is just a macro for /usr/bin.
- %{buildroot} ?
  - That is where your built files get installed to, before they are copied into your binary RPMs.
  - It is also defined in your Preamble.



Understanding the Spec: Clean & Files

- Clean removes the buildroot
- Files: List of package contents
  - If it is not in %files, it is not in the package.
  - RPM WILL complain about unpackaged files.
    - Please, please, please. Don't ever hack around this and generate files in %post.
- Example of %clean & %files sections

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

%files
%defattr(-,root,root,-)
%attr(0755,gold,fish) %{_bindir}/helloworld
```

Understanding the Spec: Changelog

- Used to track package changes
- Not intended to replace source code Changelog
- Provides explanation, 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

Best Practices

- K.I.S.S.
- Use patches, not rpm hacks
- Avoid scriptlets, minimize pre/post wherever possible
- Use Changelog
- Look to Fedora packages
- 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 rpm from inside a spec file.
  - Remember Ghostbusters? Crossing the streams? Bad.

Good Packages Put You In Control

- Practice makes perfect
- Integration with yum, Red Hat Network Satellite
- Simplify, standardize, save time and sanity
  - Build once, install many.

Useful Links

- Fedora Packaging Guidelines:
  http://fedoraproject.org/wiki/Packaging/Guidelines
  http://fedoraproject.org/wiki/Packaging/ReviewGuidelines
- Maximum RPM:
  http://www.rpm.org/max-rpm-snapshot/
- Fedora CVS Tree (contains lots of example specs)
  http://cvs.fedora.redhat.com/viewcvs/rpms/
- Rpmlint website:
  http://rpmlint.zarb.org/cgi-bin/trac.cgi