RPM packaging workshop

Development Innovations 2014, Phnom Penh

Robert Scheck

fedora™

Robert Scheck

Fedora Package Maintainer and Provenpackager

Fedora Ambassador and Ambassador Mentor

Part of Fedora Websites and Translation teams

Open Source Contributor and Software Developer

Mail: robert@fedoraproject.org
Web: https://fedoraproject.org/wiki/RobertScheck

Basic knowledge

RPM = RPM Package Manager

Spec file is for RPM what a Makefile is for „make“

Naming: `<name>-<version>-<release>.<arch>.rpm`

Some examples:

```
bash-4.3.30-2.fc21.i686.rpm
fedora-release-21-0.16.noarch.rpm
```

Source: `bash-4.3.30-2.fc21.src.rpm`

Preparing the system

RPM needs some predefined directories:

```
drwxr-xr-x 2 robert users 4096 Nov 15 14:00 BUILD
drwxr-xr-x 2 robert users 4096 Nov 15 14:00 BUILDROOT
drwxr-xr-x 2 robert users 4096 Nov 15 14:00 RPMS
drwxr-xr-x 2 robert users 4096 Nov 15 14:00 SOURCES
drwxr-xr-x 2 robert users 4096 Nov 15 14:00 SPECS
drwxr-xr-x 2 robert users 4096 Nov 15 14:00 SRPMS
```

Get them by installing and executing tools:

- `yum install rpmdevtools`
- `rpmdev-setuptree`

RPM build directories

`BUILD`	Where source code gets compiled
`BUILDROOT`	Installation directory during build
`RPMS`	Contains created RPM packages
`SOURCES`	Source files (archives, patches, ...)
`SPECS`	Contains the „make“ files for RPM
`SRPM`	Holds the created SRPM packages

The RPMS directory contains subdirectories per CPU architecture like `i386`, `i586`, `i686`, `x86_64`, `noarch`

Before starting

Build your packages never as „root“ user

- If a Makefile or the software doesn't behave during the compiling as you've expected, you maybe could damage your whole system!

- Think about a sandbox system: https://fedoraproject.org/wiki/Projects/Mock

Spec file tags

`Name:` Upstream name of the program/software

`Version:` Upstream version number

`Release:` Build/version number in Fedora

`Summary:` Really short summary (max. 80 chars)

`URL:` Website of the upstream program/software

`Source:` URI to the source code tarball/archive

Spec file tags

`Group:` RPM software group according to the `/usr/share/doc/rpm-*/GROUPS` file

`License:` Package license according to the https://fedoraproject.org/wiki/Licensing page

`BuildRequires:` Packages the software needs to build successfully

`BuildArch:` Build limitations (only for specials or CPU independent packages required)

Proper way to behave

- English language, charset: US-ASCII or UTF-8
- Spec file should be readable (no obfuscation)
- Comments can be used for non-obvious things
- Every packager should be able to read the file without knowing the software or the sources
- Do not use too much wildcards in the `%files` section, too much could get included into the package

Some hints and tricks

Which license(s) does my package use?

- `COPYING` or `LICENSE` file, source code headers, ask developer/upstream if unclear

How to figure out the build requirements?

- Try, look for errors and the install packages
- `yum groupinstall 'Development Tools'`
- `yum grouplist | grep Development`

And here we go...

A new empty skeleton can be achieved by:

- `rpmdev-newspec <name>`

RPM package building is started usually using:

- `rpmbuild -ba <name>.spec`

Spec file sections

- Meta informationen: Name, description, …
- `%prep`: Extract sources, apply patches
- `%build`: Compile the source code
- `%install`: Installation into `BuildRoot`
- `%clean`: Clean up
- `%files`: List of files and directories
- `%changelog`: List of changes of the package

Macros

- Macros are simple text substitutions
  - Some macros take parameters
- Macros allow generic spec files
- But: Partially dependent on Linux distribution
  - Macro name and value of the macro
- Format: `%<macro>` or `%{<macro>}`
- http://www.rpm.org/wiki/PackagerDocs/Macros

Macro examples

```
%build
%configure --disable-static
make %{?_smp_mflags}

%install
make DESTDIR=$RPM_BUILD_ROOT install

%post -p /sbin/ldconfig

%postun -p /sbin/ldconfig

%files
%doc AUTHORS NEWS README
%{_bindir}/idn2
```

Macro usage

- Show configuration: `rpm --showrc`
- Search: `rpm --showrc | grep <macro>`
- Expand macro:

```
$ rpm --eval %{_datadir}
/usr/share

$ rpm --eval %prep
%prep
LANG=C
export LANG
unset DISPLAY
```

Common directory macros

- %{_prefix} = /usr
- %{_exec_prefix} = %{_prefix}
- %{_bindir} = %{_exec_prefix}/bin
- %{_sbindir} = %{_exec_prefix}/sbin
- %{_lib} = /lib or /lib64
- %{_libdir} = %{_exec_prefix}%{_lib}
- %{_datadir} = %{_prefix}/share
- %{_sysconfdir} = /etc

Common directory macros

- %{_libexecdir} = %{_exec_prefix}/libexec
- %{_infodir} = /usr/share/info
- %{_mandir} = /usr/share/man
- %{_localstatedir} = /var
- %{_sharedstatedir} = /var/lib
- %{_initddir} = %{_sysconfdir}/rc.d/init.d
- %{_includedir} = %{_prefix}/include

Patches

- Used to adapt/change the source code
  - Upstream tarball should not be changed
  - Patches can be upstream or downstream
- Applied using macro `%patch` in `%prep` section
- Order is set in spec file
- `<pkgname>-<pkgversion>-<name>.patch`
- Avoid „fuzzy patches“

Patch example

```
Name:     moon-buggy
Version:  1.0.51
# …
Source:   http://seehuhn.de/media/programs/ ↩
            %{name}-%{version}.tar.gz
Patch0:   moon-buggy-1.0.51-pause.patch
# …

%prep
%setup -q
%patch0 -p1 -b .pause

# …
```

Scriptlets

- Execution of commands or scripts
  - „Hooks“ during (un)installation and update
- `%pre(un)`: before (un)installation
- `%post(un)`: after (un)installation
- Since RPM 4.4:
  - `%pretrans`: At the beginning of the transaction
  - `%posttrans`: In the end of the transaction
- `%trigger`: Interaction between different RPM packages

Scriptlet examples

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

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

%postun -p /sbin/ldconfig
```

Scriptlets

- Execution of commands or scripts
  - „Hooks“ during (un)installation and update
- `%pre(un)`: before (un)installation
- `%post(un)`: after (un)installation
- Since RPM 4.4:
  - `%pretrans`: At the beginning of the transaction
  - `%posttrans`: In the end of the transaction
- `%trigger`: Interaction between different RPM packages

Subpackages

- Separate RPM packages in one or multiple subpackages to save disk space or to avoid unwanted dependencies

```
%package pgsql
Summary:        A PostgreSQL database module for PHP
Requires:       php-pdo%{?_isa} = %{version}-%{release}
BuildRequires:  krb5-devel, openssl-devel, ↩
                postgresql-devel

%description pgsql
Back-end support in PHP for PostgreSQL
```

- `BuildArch: noarch` possible for subpackage since RPM 4.6

Hardware architectures

- `BuildArch`: Build the package only for given CPU/hardware architecture, e.g. `noarch`
- `ExcludeArch`: Exclude package during build from the given CPU/hardware architectures

```
ExcludeArch: sparc64 %{alpha}

# …

%ifnarch s390 s390x
BuildRequires: libraw1394-devel
%endif
```

- Use `%if(n)arch` macros

Different distribution versions

- Goal: Use same spec file for different versions of a Linux distribution
- Figure out smallest/lowest common base
  - Optional tags/macros such as `BuildRoot`, `%clean`
- Use macros for different behaviour/paths
  - Distribution specific macros or own hacks
- `BuildRequires` eventually for header files

Different Linux distributions

- Goal: Use the same spec file for different Linux distributions
- Use distribution specific macros for packages at `Requires` and `BuildRequires`
- Replace as much paths as possible by generic or RPM internal/default macros
- `BuildRequires` eventually for header files
- Worst case: RPM 3.0.x compatible spec file
- Avoid implicit, favor explicit

Generic RPM packages

- Goal: Same binary RPM package for all Linux distributions and CPU architectures
  - Not really spirit and purpose of RPM
  - Static linking and/or `noarch` usage if unavoidable
- RPM package should *always* be build per Linux distribution, version and architecture
- Real life: Wrong or missing dependencies in RPM packages of e.g. Adobe, Skype, HP, Dell, TeamViewer, F-Secure, ...
  - 32 bit RPM on 64 bit system

Debug information

- Compiled source code (socalled object files) contain symbols by default
- Symbols are often removed by `strip` during `make` oder `make install`
  - Disable in spec file if needed
- RPM removes unneeded symbols and moves them into -debuginfo subpackage RPM
  - Saves usually a lot of bandwidth and disk space
  - Install only afterwards if needed for debugging using GDB

Reproducible results

- Local system has maybe an optional library installed which another system hasn't installed
- Use sandbox/build system such as „mock“
  - Chroot environment with minimal installation
  - Automagic installation of RPM packages based on `BuildRequires` in the spec file
  - RPM package itself is build within the chroot
- *One* system for different Linux distributions/architectures

Advantages due to mock

- Example: Mock on CentOS 7 with 64 Bit
  - RPMs for CentOS 5, 6 and 7, Fedora 19, 20, 21 and Rawhide for 32 and 64 bit each
  - Theoretically extendable for each distribution; yum compatible repository required → `createrepo`
- RPM & mock are optimized for native building
  - Cross compiling might cause new/further issues
  - Use emulator (QEMU, Linaro, Hercules)
- Koji is built on top of mock and offers e.g. RPC and monitoring

Get your package into Fedora

Go to the Fedora website and create an account:

- https://fedoraproject.org/join-fedora

Follow the howtos and guidelines to get it into:

- https://fedoraproject.org/wiki/PackageMaintainers/Join

Actively maintain your package and care about!

Questions?

fedora™


Thank you!