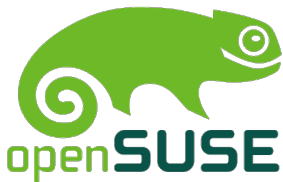


# Basic packaging introduction for openSUSE



Tomáš Chvátal et al.  
tchvatal@suse.com  
Packaging/L3 - Packaging

2017/09/08

# Introduction

# What actually is packaging

---

- Milk/Amazon?
- More like process to deliver software stack in compact and verified form to user/customer

# Why do we need packages at all?

---

- We need to be able to deliver software to users
- We need to isolate needed components
- We need to ensure proper testing of such software
- We need to compile all various software stacks together
- We need to provide comprehensive solutions for some tasks
  - Dependencies
  - Updating and migration
  - User management
  - Postinstallation configuration

# Packages

---

- Most operating systems have some type of packaging
  - Unix/Linux
  - Windows
  - Realtime operating systems
- Many popular languages have their own package system/registry
  - CPAN (perl)
  - pypi (python)
  - cabal (haskell)
- Container/sandbox formats
  - docker (dockerfiles, Open Containers Initiative)
  - flatpak
  - appimage

# Packages on Linux

---

- Most distributions are leveraging some package/software management
  - Debian/Ubuntu (apt + dpkg)
  - Arch linux (pacman)
  - Gentoo linux (portage)
  - rpm based distributions (rpm + zypper/yum/dnf)

# What are packages for the user?

---

- Collection of files and their respective permissions
- Metadata containing information about the software runtime dependencies
- Initial configuration provider

# What does it provide for the packager?

---

- Tool to provide the container to the user in a unified way
- In our distributions the `.spec` file specifying various areas of what should be done
- Tool to verify software is buildable and distributable on various distribution codestreams



Specfile surgery

## Sample initial spec

---

```
$ vim whatever.spec
```

# Preamble

## Easy parts

---

- Name, Version, Summary, Url, Patch
- License - SPDX  
<https://github.com/openSUSE/spec-cleaner#spdx-licenses>
- Group -  
[https://en.opensuse.org/openSUSE:Package\\_group\\_guidelines](https://en.opensuse.org/openSUSE:Package_group_guidelines)
- Source - with full URL path
- %description

# BuildRequires

---

- Should contain what your package needs for build
- Preferably it should be version limited (based e.g. on `configure.ac`)
- If something is wrong here the package should not build

# Requires

---

- Used for run-time dependencies
- Automatically populated for shared libraries
- Basically, all your application needs to run ought to be there
- If you get it wrong user might notice during runtime and not sooner so be careful

## Requires - scriptlets

---

- Special case of require needed only for scriptlet, not during runtime of the application
- Used to request just something extra for the specific phase

## Requires - specialities

---

- `%requires_eq` - for exactly same version requirement
- `%requires_ge` - translates to  $\geq$  on the requirement



## Example

---

```
BuildRequires: libvisio-devel >= 1.2.3
BuildRequires: cmake(GLEW) < 2.0
BuildRequires: pkgconfig(X11) => 0.9
BuildRequires: python-imaging
BuildRequires: perl(imaging)
Requires(post): update-alternatives
Requires: python-imaging
Requires: tex(yhmath)
%requires_eq perl
```

# Conflicts

---

- Used to block installation of the conflicting packages
- Usually used in case multiple packages have and provide the same set of files
- All of such packages must have the `Conflicts` specified in them to ensure it works

## Provides/Obsoletes

---

- Generally used to swap one package for another
- Provides/Obsoletes should be always versioned
- Do not obsolete unless 100% replacement

# Example

---

```
Conflicts: libwriterperfect  
Provides: liboldpackage = %{version}  
Obsoletes: liboldpackage < %{version}  
Provides: alternativepackage = %{version}
```

## Recommends/Suggests/Supplements/Enhances

---

- Weak dependencies where Recommends is soft version of Requires
- It is not required to have packages in this category installed contrary to Requires
- Supplements is mirror of Recommends, where other package states it is an addition that is to be installed if package A is present
- Suggests and Enhances are weaker version that do not get autoinstalled, just mentioned by zypper

## Example

---

```
# package will install wine if it gets installed
Recommends: wine
# if pulseaudio gets installed this package will as well
Supplements: pulseaudio
# if both vim and util-linux are installed the package
# will be installed too
Supplements: packageand(util-linux:vim)
# honorable mention of k3b in zypper output
Suggests: k3b
# if user installs xonotic this package will be
# mentioned by zypper
Enhances: xonotic
```

# Subpackages

---

- Subpackages carry the syntax logic for the main spec package
- They have it's own provides/requires/scriptlets/files/etc., but `BuildRequires` should be at the main package for readability
- There are two types, appending name (i.e. bla and bla-python) or completely renaming one (i.e. blas.spec to provide libblas1)
- All subpackages must have their own `Summary`, `Group`, and `%description`

## Example

---

```
%package python # generates bla-python
```

```
Summary: python bindings for bla
```

```
Group:    some/group
```

```
%description python
```

```
The python bindings providing a, b, and c for bla
```

```
%package -n format_spec_file # generates format_spec_file
```

```
Summary:          Binding replacing OBS service format_spec_file
```

```
Group:            Development/Tools/Other
```

```
%description -n format_spec_file
```

```
Alternative provider of format_spec_file functionality
```



Prepare

## Prepare phase

---

- Phase used for source unpacking and preparations
- All the patches should be applied here
- All the source code changing operations should happen here

Build/Check

## Build phase

---

- Phase used for source configuration and compilation
- The configuration step should use macros

`%configure`

`%cmake`

- The building should happen in threads

`make % {?_smp_mflags}`

## Check phase

---

- Phase used for test compilation and testsuite execution
- If package has testsuite it must be run and all failures should be coordinated with upstream
- We rely on this phase to do the first round of "sanity checking" for the software stack

Install

# Install phase

---

- Phase used to install the software files to the proper locations
- Also used to install additional files we might need to deliver (systemd units, ...)
- Simply just commands telling where to put what from within the compiled sources
- It is pretty good idea to rather use `install` command with proper mode than `cp/mkdir/...`

# Scriptlets



## Install phase

---

- Scriptlets are shell or lua scripts executed during various phases of the package install/removal or update
- <https://fedoraproject.org/wiki/Packaging:Scriptlets>
  - %pretrans - LUA only
  - %pre
  - %post
  - %preun
  - %postun
  - %posttrans

# Example

---

Requires(post): update-alternatives

%post

```
update-alternatives --install %{_javadocdir}/el_api.jar el_api \  
    %{_javadocdir}/%{name}-el-%{elspec}-api.jar 2030
```

# Filelists

## Files section

---

- Part of the spec stating where should be what files present among the split subpackages
- Can also contain exact specification for permissions/user/group they should contain

## Example

---

```
%files devel
%defattr(-,root,root)
%license COPYING
%doc doc/README.md
%config %[_sysconfdir]/%{name}.conf
%config(noreplace) %[_sysconfig]/%{name}-user.conf
%[_libdir]/*.so
%[_libdir]/pkgconfig/libwps*.pc
%[_includedir]/libwps-*
%attr(600, user, group) %[_sbindir]/secret
```

# Changelog

# Changelog

---

- The `%changelog` section in openSUSE spec files is populated by build scripts
- We keep the content in `%name.changes` file
- It should contain information that should be delivered to user about the changes
- The content can be pre-filled by using `osc vc` command

## Example changelog entry

---

-----  
Tue Sep 13 12:53:23 UTC 2016 - tchvatal@suse.com

- Switch to gold, we need to use less memory when linking
- Expand constraints for the debug symbols
- Use default chromium values from master\_preferences on first run rather than pseudo-duplicating in shellscript, bugs should be fixed in the masterprefs
- Add patch to fix build on 4.5+ kernels with systemlibs:
  - \* chromium-sandbox.patch



# Advanced topics

## Advanced dependencies

---

- Since rpm 4.13

Requires: (pkgA or pkgB or pkgC)

Requires: (pkgA or (pkgB and pkgC))

Requires: (pkgA >= 3.2 or pkgB)

Supplements: (foo and (lang-support-cz or lang-support-all))

Conflicts: (pkgA and pkgB)

Recommends: (myPkg-langCZ if langsupportCZ)

Requires: (myPkg-backend-mariaDB if mariaDB else sqlite)

## Library packaging

---

- We require to be subpackage named as the soname they provide
- Must run `ldconfig` after install, uninstall, and update
- Should have proper soname data set
- Static libraries should be avoided unless absolutely required

# Example

---

```
%package -n libbla1
Summary: library for bla
Group: System/Libraries

%description -n libbla1
Shared library to operate with bla

%post -n libbla1 -p /sbin/ldconfig
%postun -n libbla1 -p /sbin/ldconfig

%files -n libbla1
%{_libdir}/libbla1.so.*
```

# Conditions

---

- Usually used for distinguishing between various codestreams we provide
- [https://en.opensuse.org/openSUSE:Build\\_Service\\_cross\\_distribution\\_howto#Detect\\_a\\_distribution\\_flavor\\_for\\_special\\_code](https://en.opensuse.org/openSUSE:Build_Service_cross_distribution_howto#Detect_a_distribution_flavor_for_special_code)
- Simply conditions as in any other programming language
- Alternatively, once can use build conditionals

# Example

---

```
%if 0%{?something}  
do stuff  
%else  
do other stuff  
%endif
```

```
%if 0%{?suse_version} >= 1315  
%bcond_without wayland  
%else  
%bcond_with wayland  
%endif  
...  
%if %{with wayland}  
...  
%endif
```

## Conditions cast sample

---

```
0%{?suse_version} => "01320" => 1320
0%{?something_undefined} => "0" => 0
#
# => anything > 0 is true
%if 0%{?suse_version}
#
# => any suse with version bigger than 13.2
%if 0%{?suse_version} >= 1320
#
# => anything before 13.2?
%if (0%{?suse_version} && 0%{?suse_version} <= 1320)
```

So how to do good packaging



# Hints

---

- Be lazy! Offload to upstream rather than inventing hacks
- Use the tools we have (osc, spec-cleaner)
- Inspire yourself and use the same approach to problems like the other distributions
- Build from Pristine Sources
- Document patches in changelog
- Source Patches are meaningful on several levels
- Respect the guidelines or ask if unsure about them

Endnote

## Further reading/Contact points

---

- <https://duncan.codes/tutorials/rpm-packaging/>
- [https://en.opensuse.org/openSUSE:Packaging\\_guidelines](https://en.opensuse.org/openSUSE:Packaging_guidelines)
- <http://pes.suse.de/packagers>
- [opensuse-packaging@opensuse.org](mailto:opensuse-packaging@opensuse.org)
- [pack@suse.cz](mailto:pack@suse.cz)
- [#opensuse-factory@freenode.net](https://freenode.net/#opensuse-factory)
- [#pack@irc.suse.cz](https://irc.suse.cz/#pack)

## Thanks/Questions

---

Thank you for your attention.  
Are there any questions?