RPM Pakete bauen

Christoph Wickert

- Paketbetreuer, Übersetzer, Botschafter, Designer, ...
- Mitglied Fedora Steering Committee
- Mitarbeit bei LXDE, Xfce und OLPC
- Selbständiger IT-Berater
- E-Mail: cwickert@fedoraproject.org
- Blog: http://www.christoph-wickert.de

Grundbegriffe

- RPM = RPM Package Manager
- Datenbank unter /var/lib/rpm
- Paket-Namen: <Name>-<Version>-<Release>.<Architektur>.rpm
- Beispiel: bash-3.2-22.fc9.i386.rpm

Grundkenntnisse

- Spec-Dateien bestimmen den Aufbau eines Paketes.
- Eine Spec ist für RPM in etwa das, was das Makefile für make ist.
- Aus der Spec und den Sourcen wird das fertige Paket gebaut
- Im Source RPM befinden sich alle Sourcen und die Spec

Vorbereitungen

- `yum groupinstall 'Development Tools'`
- `yum groupinstall 'Fedora Packager'`

Je nachdem, was für Pakete ihr bauen wollt:

- `yum install 'GNOME Software Development'`
- `yum install 'Xfce Software Development'`

Vorbereitungen

- Pakete *niemals* als root bauen! Ihr könntet euren Systemen schaden zufügen, wenn sich ein Makefile nicht so verhält, wie man es erwartet.
- Deshalb: Als User eine Build-Umgebung einrichten

Build-Umgebung einrichten

- Methode 1:

```
echo '%_topdir %(echo
$HOME)/rpmbuild' > ~/.rpmmacros
mkdir -p rpmbuild
cd rpmbuild
mkdir SOURCES SPECS
```

  (restliche Ordner werden bei Bedarf erstellt)

- Methode 2:

```
rpmdev-setuptree
```

RPM Build-Umgebung

- BUILD: Ordner zum kompilieren des Paktes
- RPMS: Fertige RPM-Pakete
- SOURCES: Quellen (Source-Archive, Konfigurationsdateien, etc.)
- SPECS: SPEC-Datei für die RPMs
- SRPMS: Fertige Source-RPMs

Erstellen einer neuen Spec

```
$ rpmdev-newspec foo
Skeleton specfile (minimal) has been
created to "foo.spec".

$ rpmdev-newspec pyfoo
Skeleton specfile (python) has been
created to "pyfoo.spec".

$ rpmdev-newspec libfoo
Skeleton specfile (lib) has been
created to "libfoo.spec".

$ rpmdev-newspec -t dummy foo
Skeleton specfile (dummy) has been
created to "foo.spec".
```

rpmdev-newspec

- Verfügbare Vorlagen
  - dummy
  - lib
  - minimal
  - ocaml
  - perl
  - php-pear
  - python
  - R
  - ruby

Notwendige Tags

- Name: Name des Programms
- Version: Version des Programms
- Release: Version in Fedora
- Summary: *Kurze* Zusammenfassung
- Group: RPM Gruppe aus /usr/share/doc/rpm-*/GROUPS
- License: Lizenz, also BSD, GPL etc.
- URL: Homepage des Programms
- Source0: Quelle mit voller URL

Notwendige Tags

- Source1: ggf. weiter Quellen
- BuildRequires: Für das Kompilieren benötigte Pakete
- Requires: Runtime-Abhängigkeiten, weglassen, wenn keine Abhängigkeiten bestehen oder alle automatisch erkannt werden
- %description: Ausführlichere Beschreibung, ganze Sätze, Punkt am Ende.

Optionale Tags

- Epoch: Epoche, übertrumpft Version und Release
- BuildRoot: Temporärer Ordner, in dem gebaut wird, *früher notwendig!*
- BuildArch: Architektur, für die gebaut wird, z. B. x86_64, noarch
- ExcludeArch: *nicht* für eine Architektur bauen
- ExclusiveArch: *nur* für diese Architektur bauen

Optionale Tags

- Patch0: Wie Source, nur für Patches
- Provides: Paket stellt etwas zu Verfügung
- Obsoletes: Paket ersetzt ein anderes
- Conflicts: Paket kollidiert mit einem anderen Paket
- AutoReq, AutoProv, AutoReqProv: no deaktiviert automatisch generierte Requires, Provides oder beides

Optionale Tags

- Distribution: Distribution (bei Fedora weglassen)
- Vendor: Hersteller (bei Fedora weglassen, macht das Build-System)
- Packager: Autor des Paketes (bei Fedora weglassen)

Phasen des Paketbaus

- %prep: Entpacken der Sourcen, Patches anwenden etc.
- %build: Übersetzen der Software
- %install: Installation ins temporäre Verzeichnis BUILDROOT
- %clean: Aufräumen

Phasen der Paketinstallation

- %pre: Befehl vor Installation
- %post: Befehl nach Installation
- %preun: Befehl vor Deinstallation
- %postun: Befehl nach Deinstallation
- %pretrans: Befehl ganz am Anfang der Transaktion (seit rpm 4.4)
- %posttrans: Befehl ganz am Ende der Transaktion (seit rpm 4.4)

Reihenfolge der Phasen

1.%pretrans neues Paket

2.%pre des neuen Paketes

3.neues Paket installieren

4.%post des alten Paketes

5.%preun des alten Paketes

6.altes Paket entfernen

7.%postun des neue Paketes

8.%posttrans des neuen Paketes

rpmbuild

- -bs foo.spec: build source
- -bb foo.spec: build binary
- -ba foo.spec: build all
- --rebuild foo-1.0.src.rpm: rebuild
- --nodeps: Abhängigkeiten nicht beachten (wie bei -bs)
- --clean: nach erfolgreichem Bauen aufräumen (wie bei --rebuild)

Ein erster Versuch

```
rpmbuild -ba foo.spec
```

(Zeit für eine Live-Demo)

Die %files-Sektion

- Gibt an, welche Dateien in dem Paket sind
- Gibt an, um was für Dateien es sich handelt
- %config: Konfigurationsdateien
  - Wenn verändert, werden sie beim Update als *.rpmold gesichert
  - %config(noreplace): Wenn geändert, wird beim Update die neue Version als *.rpmnew angelegt

Die %files-Sektion

- %doc: Dokumentation
  - voller Pfad im Paket: Kennzeichnet Dokumentation
  - nur Dateiname: kopiert Dateien in das Paket
- %lang: Sprachspezifische Dateien
  - %lang(de) = Deutschsprachig
- keine absoluten Dateipfade, sondern Macros verwenden

Wichtige Ordner-Makros

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

Weitere Ordner-Makros

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

Dateien und Ordner in %files

- Dateien nicht doppelt auflisten.
- Keine Dateien auflisten, die schon zu einem anderen Paket gehören. Wenn doch notwendig: Conflicts Tag setzen
- Keine Ordner auflisten, die schon einem anderen Paket gehören, sondern es als Requires hinzufügen
- Darauf achten, dass alle Ordner in Paket enthalten sind, sonst bleiben leere Ordner bei Deinstallation zurück

Beispiel für %files-Sektion

- hicolor-icon-theme enthält alle Ordner in /usr/share/icons/hicolor/

```
...
Requires: hicolor-icon-theme
...
%files

%defattr(-,root,root,-)

%doc AUTHORS ChangeLog COPYING

%{_bindir}/foo

%{_datadir}/foo/

%{_datadir}/icons/hicolor/*/apps/foo.png

%{_datadir}/icons/hicolor/scalable/apps/foo.svg
```

Die feine englische Art

- Die Spec sollte leserlich sein.
- Die Spec sollte für sich sprechen. Andere Packager sollten sie verstehen, ohne die Source(n) zu kennen.
- Kommentare nutzen, wenn Dinge nicht ersichtlich sind.
- Nicht zu viele Wildcards in der %files Sektion nutzen, damit keine unerwünschten Dateien im Paket landen.

Lizenzen

- In COPYING oder LICENSE nachsehen, sofern vorhanden
- In die Header des Quellcodes schauen
- Unterscheiden zwischen Versionen, z. B. GPLv2 („Version 2 only“) oder GPLv2+ („or any later version“)
- Den Entwickler fragen, wenn die Lizenz immer noch unklar ist

Abhängigkeiten

- Werden i. d. R. automatisch erkannt
- Die Homepage oder README kann Aufschluss geben.
- Die Ausgabe von ./configure beobachten
- config.log anschauen
- *-devel Pakete deinstallieren, z. B. mit `rpmdev-rmdevelrpms`
- Zum Prüfen Paket in `mock` bauen

Abhängigkeiten *devel-Paket

- BuildRequres des Basispaketes sind Requires des *devel Paketes
- rpm kann Abhängigkeiten aus pkg-config Datei extrahieren:

```
$ grep Requires
/usr/lib/pkgconfig/exo-0.3.pc
Requires: gtk+-2.0 libxfce4util-1.0
```

→ gtk2-devel, libxfce4util-devel

Patches

- Namensschema: <Paketname>-<Paketversion, für die der Patch gemacht wurde>-<Patchname>.patch
- Patches werden in %prep angewandt
- Reihenfolge wird in der Spec festgelegt
- Patches dürfen nicht fuzzy sein

Patches anwenden

Beispiel zum Anwenden von Patches

```
...
Source0: http://foo.org/foo-1.0-tar.gz
Patch0: foo-1.0-world-domination.patch
Patch1: foo-1.0-make-it-suck-less.patch
...
%prep
%setup
%patch0 -p1 -b .world-domination
%patch1 -p1 -b .suck-less
```

Unterpakete

- Pakete in Unterpaket unterteilen, um ungewollte Abhängigkeiten zu vermeiden:

```
%package        plugin
Summary:        Xfce panel plugin for Foo
Group:          User Interface/Desktops
BuildRequires:  xfce4-panel-devel >= 4.4.0
Requires:       %{name} = %{version}-%{release}
Requires:       xfce4-panel >= 4.4.0
%description    plugin
This package contains the foo plugin for the Xfce
panel.
```

noarch Unterpakete

- Seit kurzem unterstützt rpm noarch Unterpakete, nützlich beispielsweise für Dokumentation.
- Base Paket muss architektur-spezifisch sein!
- Ältere Versionen von rpm akzeptieren nur einen BuildArch-Tag. Werden mehrere angegeben, überschreibt der letzte alle vorherigen.

RPM-Makros anpassen

- RPM-Macros können in /etc(rpmmacros oder ~/.rpmmacros angepasst werden:
- %_topdir %(echo $HOME)/fedora/rpmbuild
- %_tmppath /mnt/store/tmp/
- %_gpg_name 1999A427
- %packager Christoph Wickert <cwickert@fedoraproject.org>
- %_missing_doc_files_terminate_build 1
- %_unpackaged_files_terminate_build 0

RPM-Makros erweitern

- Erweiterter Disttag:

  %distname fc

  %distversion %(rpm -qf --qf='%{VERSION}' /etc/redhat-release)

  %dist .%{distname}%{distversion}

- Alphatag für Snapshots (in der Spec):

  %global gitdate

  %global gitversion

  %global .%{gitdate}git%{gitversion}

Makros auflösen

- Konfiguration anzeigen: `rpm –showrc`
- Suchen: `rpm –showrc | grep <macro>`
- Evaluieren:

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

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

Spectool

- spectool ist ein Helfer für den Umgang mit Sourcen und Patches in Spec-Dateien
- Sourcen anzeigen:

```
spectool foo.spec
```

- Sourcen herunterladen:

```
spectool -g -S foo.spec
```

- Patches herunterladen:

```
Spectool -g -P foo.spec
```

Danke für Eure Aufmerksamkeit!

http://fedoraproject.org