

**Nach dem ersten RPM
gehts erst
richtig los...**

Chemnitzer Linux-Tage 2013, Chemnitz

Robert Scheck

fedora 

Robert Scheck

Fedora Package Maintainer und Provenpackager
Fedora Ambassador und Ambassador Mentor
Unterstützung der Website-/Übersetzungsteams
Open Source Contributor und Software-Entwickler

Mail: robert@fedoraproject.org

Web: <http://fedoraproject.org/wiki/RobertScheck>



RPM in Kürze

RPM = RPM Package Manager

Spec-Datei ist für RPM wie Makefile für make

Format: `<Name>-<Version>-<Release>.<Arch>.rpm`

Einige Beispiele:

- ▶ `bash-4.2.24-1.fc18.i686.rpm`
- ▶ `fedora-release-17-1.noarch.rpm`

Source-RPM: `bash-4.2.24-1.fc18.src.rpm`



Grundwissen

Installation und Ausführung von:

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

Ein neues leeres Spec-Skelett erhält man mit:

- ▶ `rpmdev-newspec <name>`

RPM-Paketbau wird normalerweise gestartet mit:

- ▶ `rpmbuild -ba <name>.spec`



Sicherheitshinweise

Pakete niemals als „root“-Benutzer bauen

- ▶ Wenn ein Makefile oder die Software sich nicht wie erwartet verhält, kann das gesamte System beschädigt werden!
- ▶ Sandbox-System in Erwägung ziehen:
<http://fedoraproject.org/wiki/Projects/Mock>



Spec-Aufbau und Phasen

- ▶ Meta-Informationen: Name, Beschreibung, ...
- ▶ `%prep`: Quellen entpacken, Patches anwenden
- ▶ `%build`: Kompilieren der Software
- ▶ `%install`: Installieren nach `BuildRoot`
- ▶ `%clean`: Aufräumen
- ▶ `%files`: Liste der Dateien/Verzeichnisse
- ▶ `%changelog`: Liste der Änderungen



Makros

- ▶ Makros sind einfache Text-Ersetzungen
 - ▶ Makros können Parameter haben
- ▶ Ermöglichen generische Spec-Dateien
- ▶ Sind teilweise auch distributionsspezifisch
 - ▶ Betrifft Makroname als auch Wert des Makros
- ▶ Aufbau: %<Makroname> bzw. %{<Makroname>}
- ▶ <http://www.rpm.org/wiki/PackagerDocs/Macros>



Beispiele für Makros

```
%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
```



Umgang mit Makros

- ▶ Konfiguration anzeigen: `rpm --showrc`
- ▶ Suchen: `rpm --showrc | grep <Makro>`

- ▶ Makro auflösen:

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

```
$ rpm --eval %prep
```

```
%prep
```

```
LANG=C
```

```
export LANG
```

```
unset DISPLAY
```



Gängige Verzeichnis-Makros

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



Gängige Verzeichnis-Makros

- ▶ `%{_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

- ▶ Anpassung des Quellcodes der Software
 - ▶ Upstream-Tarball sollte nicht angepasst werden
 - ▶ Patches können Upstream oder Downstream sein
- ▶ Anwendung in %prep mittels Makro %patch
- ▶ Reihenfolge wird in Spec-Datei festgelegt
- ▶ `<Paketname> - <Paketversion> - <Name> . patch`
- ▶ „Fuzzy Patches“ vermeiden



Beispiel für Patch

```
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

# ...
```



Skriptlets

- ▶ Ausführung von Befehlen bzw. Skripten
 - ▶ „Hooks“ während (De)Installation und Update
- ▶ `%pre(un)`: Vor (De)Installation
- ▶ `%post(un)`: Nach (De)Installation
- ▶ Seit RPM 4.4:
 - ▶ `%pretrans`: Ganz zu Beginn der Transaktion
 - ▶ `%posttrans`: Ganz am Ende der Transaktion
- ▶ `%trigger`: Interaktion zwischen verschiedenen RPM-Paketen



Beispiele für Skriptlets

```
%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
```



Unterpakete

- ▶ Pakete in Unterpakete unterteilen, um Platz zu sparen und/oder ungewollte Abhängigkeiten zu vermeiden

```
%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
```

- ▶ Unterpaket kann seit RPM 4.6 auch BuildArch: noarch sein



Hardware-Architekturen

- ▶ `BuildArch`: Paket nur für angegebene CPU-/ Hardware-Architekturen bauen, z.B. `noarch`
- ▶ `ExcludeArch`: Paket beim Bauen von diesen CPU-/ Hardware-Architekturen ausschließen

```
ExcludeArch: sparc64 %{alpha}
# ...
%ifnarch s390 s390x
BuildRequires: libraw1394-devel
%endif
```

- ▶ Makros `%if(n)arch` benutzen



Unterschiedliche Versionen

- ▶ Ziel: Gleiche Spec-Datei für verschiedene Versionen einer Distribution verwenden
- ▶ Kleinsten gemeinsamen Nenner ermitteln
 - ▶ Optionale Tags/Makros wie `BuildRoot`, `%clean`
- ▶ Makros für unterschiedliches Verhalten
 - ▶ Distributionsspezifische Makros oder eigene Hacks
- ▶ `BuildRequires` eventuell auf Header-Dateien



Verschiedene Distributionen

- ▶ Ziel: Gleiche Spec-Datei für verschiedene Linux-Distributionen verwenden
- ▶ Distributionsspezifische Makros für Pakete in `Requires` bzw. `BuildRequires` verwenden
- ▶ Möglichst viele generische bzw. RPM-eigene Makros für Pfade verwenden
- ▶ `BuildRequires` eventuell auf Header-Dateien
- ▶ Spec-Datei für schlimmstenfalls RPM 3.0.x
- ▶ Möglichst explizit statt implizit



Generische RPM-Pakete

- ▶ Ziel: Gleiches binäres RPM-Paket für alle Linux-Distributionen und CPU-Architekturen
 - ▶ Eigentlich nicht der Sinn und Zweck von RPM
 - ▶ Notfalls statisch linken und/oder noarch verwenden
- ▶ RPM-Paket sollte *immer* je Linux-Distribution, Version und Architektur durchgebaut werden
- ▶ Praxis: Fehlerhafte/fehlende Abhängigkeiten in RPM-Paketen von Adobe, Skype, HP, Dell, TeamViewer, F-Secure, ...
 - ▶ 32 Bit-RPM auf 64 Bit-System



Debug-Informationen

- ▶ Kompilierter Quellcode (also Objektdateien) enthält standardmäßig Symbole
- ▶ Symbole werden während `make` oder `make install` oftmals mittels `strip` entfernt
 - ▶ Gegebenenfalls in Spec-Datei deaktivieren
- ▶ RPM entfernt unbenötigte Symbole und lagert diese in `-debuginfo` RPM-Unterpaketen aus
 - ▶ Spart relativ viel Bandbreite und Speicherplatz
 - ▶ Für Fehlersuche mittels GDB bei Bedarf nachinstallieren



Reproduzierbare Ergebnisse

- ▶ Lokales System hat eventuell eine optionale Bibliothek installiert, die ein anderes System nicht installiert hat
- ▶ Sandbox/Buildsystem wie „mock“ verwenden
 - ▶ Chroot-Umgebung mit Minimalinstallation
 - ▶ Automatische Nachinstallation der RPM-Pakete aus `BuildRequires` in der Spec-Datei
 - ▶ RPM-Paket wird innerhalb des Chroots gebaut
- ▶ *Ein* System für verschiedene Distributionen/Architekturen



Vorteile durch Mock

- ▶ Beispiel: Mock auf CentOS 6 unter 64 Bit
 - ▶ RPMs für CentOS 5 und 6, Fedora 17, 18, 19 und Rawhide für jeweils 32 und 64 Bit
 - ▶ Theoretisch um jede Distribution erweiterbar; Yum-kompatibles Repository erforderlich → createrepo
- ▶ RPM & Mock für natives Kompilieren optimiert
 - ▶ Cross-Compiling bringt teilweise weitere Probleme
 - ▶ Emulator (QEMU, Linaro, Hercules) verwenden
- ▶ Koji baut auf Mock auf, bietet u.a. RPC und Überwachungen



Paket nach Fedora bringen

Fedora-Webseite besuchen & Account erstellen:

- ▶ <http://join.fedoraproject.org/>

Anleitungen, Regeln und Abläufen folgen:

- ▶ <http://fedoraproject.org/wiki/PackageMaintainers/Join>

Wichtig: RPM-Paket aktiv pflegen & aktualisieren!



Fragen?



fedora™

Vielen Dank!

