Building Linux Packages with RPM

S9239, Long Beach
Tuesday 1:30pm, February 24, 2004
Alex deVries <alex@onefishtwo.ca>

Outline

- Why?
- The goal of pristine sources
- Some unobvious suggestions
- How it all fits together
- Before you get started
- Steps to actually build packages
- A real life example: the wu-ftpd package
- More details: dependancies, subpackages, scripts, architectures
- More information

Why build packages?

- It is a best practice to use a package manager
  - Changes are easily tracked
  - Package changes can be rolled back
  - Dependancies are known and met
- Great way to share software development with other people
- Reproducible builds, so you know how to apply future changes
- You're no longer dependant on other people to build your packages

Doing things the hard way

- One of the great but painful goals is to make sure that builds are reproducible
- Pristine sources are a fundamental of rpm building, unlike Debian packages
- rpmbuild will only write source and binary RPMs if the build process was completed from beginning to end*

* Yes, you can circumvent this. But you shouldn't.

Some notes on pristine sources

- Changes to the source tree should be reflected as patch files, not simply a new tarball of modified source
- This ensures that the source of all the code changes can be identified
- To generate diff files:

```
cp -av wu-ftpd wu-ftpd.orig
```

- Make necessary changes

```
diff -ruN wu-ftpd.orig wu-ftpd
```

Unobvious suggestions when building packages

- If a package building does a 'make install', it may overwrite parts of your build environment.
- Two ways to preserve your build environment:
  - Always always use a buildroot
  - Never build as root
- Use ccache to accelerate your builds
- Expect an iterative software development process

Building packages as a non-root user

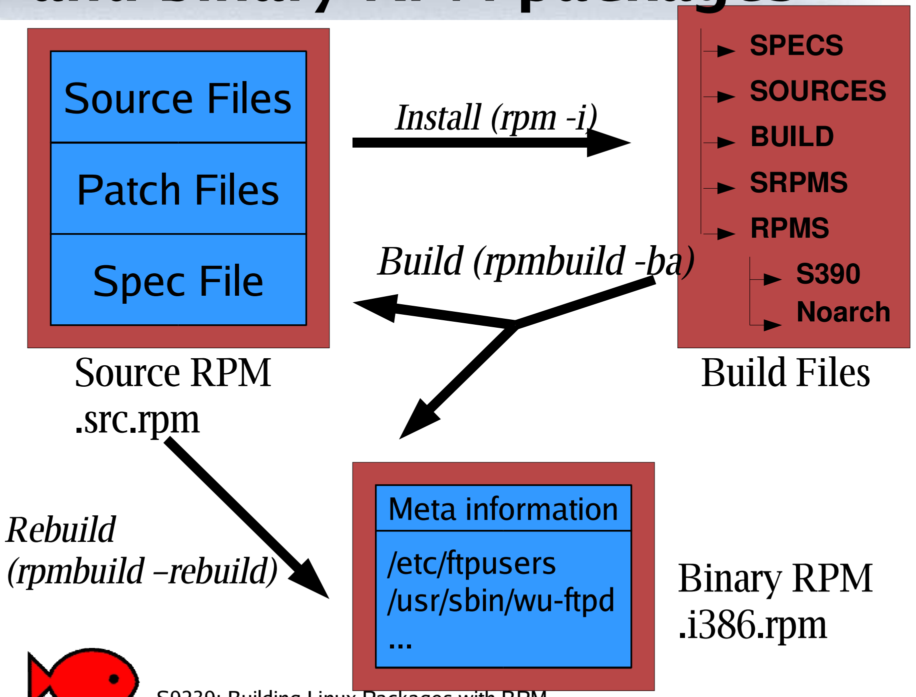
- The default is to do building in /usr/src/redhat (or /usr/src/suse)
- How to set this up:
  - Create a file called ~/.rpmmacros
  - Add: `%_topdir   /home/adevries/rpm`
  - `cd ~/rpm ; mkdir -p RPMS/i386 RPMS/noarch SRPMS SOURCES BUILD SPECS`

Using ccache

- Having to rebuild a lot of C source from scratch every time can be very time consuming, especially on slow architectures

- How to use ccache:
  - Install the ccache RPM
  - Before building, run:
    ```
    export CC='ccache gcc'
    ```
  - All future compiling will first access the ccache

The relationship between source and binary RPM packages



Pieces of an RPM

- In a source RPM:
  - Upstream sources
  - The SPEC file

```
[adevries]$ rpm -qlp samba-3.0.2rc2-1.src.rpm
filter-requires-samba_rh8.sh
filter-requires-samba_rh9.sh
samba-3.0.2rc2.tar.bz2
samba3.spec
```

- In a binary RPM:
  - Meta information
  - Files to install, with signatures
  - Scripts for pre- and post- installation, pre- and post-installation

Rebuilding an existing source RPM

- The simplest case, building a binary RPM from a source RPM
- This will help you prove that you know what's in the binaries you're running
- Just run:

```
    rpmbuild -rebuild foo-3.2-1.src.rpm
```

What's involved in creating a new RPM

- First, outside of RPM:
  - Get the upstream source
  - Read the installation instructions!
  - Apply your own patches, if you need to
  - Make sure it builds
- Then, using rpmbuild:
  - Copy the sources and patches to the right directories
  - Setup a starting SPEC file
    - Write the meta data
    - Write the %prep section

Inside a SPEC file

- Everything starting with a % is an RPM macro
- Sections:
  - Metadata: name, version, summary, packager, etc
  - %prep – unpacking sources, applying patches
  - %build – instructions to build (possibly compile) the software
  - %install – copy the files and directories under a buildroot directory the way they should appear in the binary pakage
  - %clean – clean up the build environment
  - %files – listing of files that should be copied in
  - Scripts – shell scripts that should be run when the package is (un)installed
  - %changelog – list of changes to the spec file

A real world example

- Our example: wu-ftpd, a simple FTP server
- There are actually RPMs out there for this version, but we'll walk through the creation of this as if there weren't

First, make sure the thing builds

- Do what you'd normally do to compile the software
  - Unpack the source (tar -xzvf)
  - Apply your patches(patch < ...)
  - Compile (./configure ; make)

Now, let's create a SPEC file...

1. Setting up Metadata (½)

General info

Requirements

Where to do the build

```
Summary: An FTP daemon provided by Washington
   University.
Name: wu-ftpd
Version: 2.6.2
Release: 1
License: BSD
Group: System Environment/Daemons
URL: http://www.wu-ftpd.org/
Source: ftp://ftp.wu-ftpd.org/pub/wu-ftpd/wu-
   ftpd-2.7.0-20020304.tar.bz2
Source1: ftpd.log
Source2: ftp.pamd
Source3: wu-ftpd-xinetd
Source4: ftpaccess
Patch0: wu-ftpd-2.6.0-redhat.patch
Patch1: wu-ftpd-2.6.0-owners.patch
Provides: ftpserver
Prereq: fileutils, openssl
Requires: xinetd, /etc/pam.d/system-auth
Buildroot: /tmp/%{name}-root
```

Setting up Metadata (2/2)

```
%description
The wu-ftpd package contains the wu-ftpd FTP (File Transfer
   Protocol)
server daemon. The FTP protocol is a method of transferring files
between machines on a network and/or over the Internet. Wu-ftpd's
features include logging of transfers, logging of commands, on the
   fly
compression and archiving, classification of users' type and
   location,
per class limits, per directory upload permissions, restricted
   guest
accounts, system wide and per directory messages, directory alias,
cdpath, filename filter, and virtual host support.
```

2. A first try of a %prep section

Header *Quiet* *Into a dir called*

Unpack

```
%prep
%setup -q -n %{name}
%patch0 -b .redhat
%patch1 -b .owners
find . -type d -name CVS |xargs rm -rf
```

Keep a copy of the original

Apply patch n

Clean up CVS directories

Try a build...

```
[adevries@cubalibre redhat]$ rpmbuild -ba SPECS/wu-ftpd.spec1
Executing(%prep): /bin/sh -e /var/tmp/rpm-tmp.40603
+ umask 022
+ cd /home/adevries/onefishtwo/redhat/BUILD
+ LANG=C
+ export LANG
+ cd /home/adevries/onefishtwo/redhat/BUILD
+ rm -rf wu-ftpd
+ /usr/bin/bzip2 -dc /home/adevries/onefishtwo/redhat/SOURCES/wu-
ftpd-2.7.0-20020304.tar.bz2
+ tar -xf -
...
+ /bin/chmod -Rf a+rX,g-w,o-w .
+ echo 'Patch #0 (wu-ftpd-2.6.0-redhat.patch):'
Patch #0 (wu-ftpd-2.6.0-redhat.patch):
+ patch -p0 -b --suffix .redhat -s
The text leading up to this was:
--------------------------
|--- wu-ftpd-2.6.0/src/pathnames.h.in.patch0    Sun Oct  3 09:13:09
1999
|+++ wu-ftpd-2.6.0/src/pathnames.h.in   Thu Oct 21 11:36:20 1999
--------------------------
File to patch:
```

Oops! Fixing the %prep section

- Patch was applied at the wrong level:
- In wu-ftpd-2.6.0-redhat.patch:

```
--- wu-ftpd-2.6.0/src/pathnames.h.in.patch0     Sun Oct  3 09:13:09
   1999
+++ wu-ftpd-2.6.0/src/pathnames.h.in    Thu Oct 21 11:36:20 1999
```

- On build system:

```
[adevries@cubalibre redhat]$ ls BUILD/
wu-ftpd
```

- The solution: apply the patches one directory level in, so change it to:

```
%patch0 -p1 -b .redhat
%patch1 -p1 -b .owners
```

3. A first try of a %build section

Header

Custom build script

```
%build
%configure -enable-quota -enable-pam --disable-rfc931 --enable-ratios
\
        --enable-passwd --disable-dnsretry --enable-ls --enable-ipv6
\
        --enable-tls

sed -e "s/\/\* #undef SHADOW_PASSWORD \*\//#define SHADOW_PASSWORD
1/g" src/config.h

# Make the version what we want it to be
cat >src/newvers.sh <<EOF
echo 'char version[] = "Version wu-%{version}-%{release}";' >vers.c
EOF
chmod 0755 src/newvers.sh
make
```

Other custom build instructions

4. A first try of the install section

Header *Remove the old build root and recreate it*

Install into the build root

```
%install
rm -rf $RPM_BUILD_ROOT
mkdir -p $RPM_BUILD_ROOT/etc $RPM_BUILD_ROOT/usr/sbin
make install DESTDIR=$RPM_BUILD_ROOT
install -c -m755 util/xferstats $RPM_BUILD_ROOT/usr/sbin
cd rhsconfig
install -c -m 600 ftpusers  ftphosts ftpgroups
   ftpconversions $RPM_BUILD_ROOT/etc
strip -R .comments $RPM_BUILD_ROOT/usr/sbin/* || :
mkdir -p $RPM_BUILD_ROOT/etc/{pam,logrotate}.d
install -m 644 %{SOURCE1}
   $RPM_BUILD_ROOT/etc/logrotate.d/ftpd
install -m 644 %{SOURCE2} $RPM_BUILD_ROOT/etc/pam.d/ftp
ln -sf in.ftpd $RPM_BUILD_ROOT/usr/sbin/wu.ftpd
ln -sf in.ftpd $RPM_BUILD_ROOT/usr/sbin/in.wuftpd
mkdir -p $RPM_BUILD_ROOT/etc/xinetd.d
install -m644 %{SOURCE3} $RPM_BUILD_ROOT/etc/xinetd.d/wu-
   ftpd
install -c -m0600 %{SOURCE4} $RPM_BUILD_ROOT/etc
```

The %file section

Header

Default attributes of next files

Config file

```
%files
%defattr(-,root,root)
%config(noreplace) /etc/xinetd.d/wu-ftpd
%doc README ERRATA CHANGES CONTRIBUTORS
%doc doc/HOWTO doc/TODO doc/examples
%/usr/man/*/*.*
%config /etc/ftp*
%config /etc/pam.d/ftp
%config /etc/logrotate.d/ftpd

%defattr(0755,bin,bin)
/usr/sbin/*
/bin/*
```

Glob files together

Special notes about %files

- If a file isn't listed in %files, it won't make it into the binary RPM
- Setting mode and owner can let you put setuid or root owned files into a binary RPM that you could never create on the build system as a non-root user
- Files marked %config are renamed on upgrading, not replaced
- Files marked %doc are not installed when package is added with –nodocs

Trying the build

- Rebuild quickly with:

```
Rpmbuild -bi -short-circuit
```

- We get more errors:

```
Processing files: wu-ftpd-2.6.2-1
error: File not found: /var/tmp/wu-ftpd-
root/etc/pam.d/ftp
...

RPM build errors:
    File not found: /var/tmp/wu-ftpd-
root/etc/pam.d/ftp
```

Fix the build, and rebuild from scratch

- Add the missing file installation to the %install section, and rebuild using:

```
 rpmbuild -ba wu-ftpd.spec
```

- And the result is:

```
...
Wrote: redhat/SRPMS/wu-ftpd-2.6.2-1.src.rpm
Wrote: redhat/RPMS/i386/wu-ftpd-2.6.2-1.i386.rpm
```

- Yay!

Dependancies

- rpmbuild tries to identify dynamically linked files automatically
- If it detects a file is a Linux binary, it will determine shared libraries using ldd, eg.:

```
[adevries@cubalibre bin]$ ldd ftpd
        libcrypt.so.1 => /lib/libcrypt.so.1 (0x40025000)
        libnsl.so.1 => /lib/libnsl.so.1 (0x40053000)
        libresolv.so.2 => /lib/libresolv.so.2 (0x40069000)
        libssl.so.2 => /lib/libssl.so.2 (0x4007b000)
        libcrypto.so.2 => /lib/libcrypto.so.2 (0x400ab000)
        libpam.so.0 => /lib/libpam.so.0 (0x4017f000)
        libdl.so.2 => /lib/libdl.so.2 (0x40187000)
        libc.so.6 => /lib/i686/libc.so.6 (0x42000000)
        /lib/ld-linux.so.2 => /lib/ld-linux.so.2
(0x40000000)
```

- You can add other dependancies too, or override the default mechanism

Subpackages

- Sometimes, packages are large enough that not everybody wants all the files associated with a piece of software
- Making subpackages allows
- For example, vim:
  - vim-common-6.1-18.8x.1: common files for all vim packages
  - vim-minimal-6.1-18.8x.1: just the minimal files to get vim running, requires vim-common
  - vim-enhanced-6.1-18.8x.1: the enhanced and larger files, requires vim-common

Scripts

- Scripts available: %postinstall, %postuninstall, %preinstall, %preuninstall
- These are generic bash scripts that are to be run on the target system during package installations, deletions or upgrades
- These can be queried with:

```
rpm -q -scripts packagename
```

Architectures

- By default, packages built are of the same architecture and OS as your build environment
- The 'noarch' RPM is a special one, and will build a binary package which can install anywhere
- You can set the target build architecture and OS of a package or subpackage with:

```
Buildarch: vax
BuildOS: Linux
```

- You can, with some difficulty, build binary packages for other architectures. This will probably rely on using a cross compiling toolchain.

More RPM Information

- Sadly, RPM documentation is somewhat incomplete
- Maximum RPM is still a good enough, it is available at http://www.rpm.org

Questions?

- Alex deVries <alex@onefishtwo.ca>
- Please fill out the evaluation cards! This is session S9239.