Building RPM packages

Bob Dowling, University of Cambridge Computing Service

Agenda

- Why build packages?
- Overview of package construction
- Simple example
- Extending the simple example
- Less simple example
- Extending the less simple example
- Complex example

Why build packages?

- Simplicity of installation
- Simplicity of upgrade
- Simplicity of management
- Keeps source code, patches and configs together
- *Not just Red Hat Linux!*
- Red Hat, SuSE, Mandrake, …

Overview of package construction

- Spec file
- Source code
- Patches
- Managed unpacking, building and installation

Simple example

- Very simple program
- Crufty old C++ program: `ackerman`
  - Ackermen's function is a mathemtical curiosity
  - you don't need to know the details
- Source code
- Manual page
- Licence file (GPL)

Source code

- Files:
  - `Ackerman.h, Ackerman.cxx, LongArray.cxx, main.cxx`
  - `GPL`
  - `ackerman.1`
- Package name: `ackerman`
- Version of source code: `1`
- Directory name: `ackerman-1`

Processing the source code

- `tar` and `gzip`
  - `$ tar -cf ackerman-1.tar ackerman-1`
  - `$ gzip -9 ackerman-1.tar`
- Put archive in source code directory
  - `/usr/src/redhat/SOURCES`

Specification files

- Controls the package
- Everything to do with the package is here
- Keeps all the information together
- Provides information
- Controls building the package
- *No manual override is permitted*

Creating the Specification file

- `emacs` knows what the template looks like
- Keyed on filenames ending with `.spec`
- Not compulsory to use `emacs`
- But it helps
- Traditional to name spec file after package
- `ackerman.spec`

Initial template spec file

- Three types of entry
- Key: Value
  - `Summary:`, `Source0:`, ...
- %Section followed by content
  - `%description`, `%prep`, `%build`, `%install`, ...
- %Macros for variables and command sequences
  - `%name`, `%version`, `%setup`, ...

Blank `ackerman.spec`

```
1.Summary:
Name: ackerman
Version:
Release: 1
URL:
Source0: %{name}-%{version}.tar.gz
License:
Group:
BuildRoot: %{_tmppath}/%{name}-root

%description

%prep
%setup -q

%build

%install
rm -rf $RPM_BUILD_ROOT

%clean
rm -rf $RPM_BUILD_ROOT

%files
%defattr(-,root,root)


%changelog
* Sun Feb 16 2003 Bob Dowling <rjd4@noether.csi.cam.ac.uk>
- Initial build.
```

First, simple fields

- Correspond to info from `rpm --query --info`
  - Summary: short, one-line description
  - Name: name of the software package
  - Version: version of the original source code
  - Release: number of this *packaging* of the code
  - URL: web page for product
  - License: the licence for the software
  - Group: what category the software falls in

Groups of packages

- Varies a bit (too much) between distributions
- `rpm --query --queryformat '%{group}\n' package`
- Format is Category/Subcategory

Groups used by RHL

- Amusements/Games, Amusements/Graphics
- Applications/…
  - Archiving, Communications, CPAN, Databases, Editors, Emulators, Engineering, File, Internet, Multimedia, Office, Productivity, Publishing, System, Text, Utilities
- Development/…
  - Debuggers, Languages, Libraries, System, Tools
- Documentation, Networking, System Environment, User Interface, Utilities

Build-specific parameters

- `Source0:`
  - Primary source files, tarred and gzipped
  - Note default naming scheme assumed
  - `%{name}-%{version}.tar.gz`
- `BuildRoot:`
  - Don't do install directly to real locations
  - Build a fake tree to install into
- There will be other parameters later

%Macros

- `%name`
  - Whatever is given by the `Name:` line
- `%version`
  - Whatever is given by the `Version:` line
- `%_tmppath`
  - A system macro for where installations should go
  - `/var/tmp` on RHL systems
  - Underscores indicate system-wide macros

`%description` section

- `%description` defines a section
  - Everything up to the next %section
- `%description` gives the paragraph-style info

```
%description
This package provides a single command, ackerman that
calculates the ackerman(m,n) function, internally
remembering previously calculated values to avoid
recalculation.
```

The `%prep` section

```
%prep
%setup -q
```

- Preparation of the source code
- The `%setup` macro:
  - Unpacks the `.tar.gz` file
  - `-q` makes it **q**uieter (don't see every file unpacked)
  - Sanity checks

Running the %setup section

- rpmbuild -bp ackerman.spec

```
Executing(%prep): /bin/sh -e /var/tmp/rpm-tmp.7425
+ umask 022
+ cd /home/rjd4/redhat/BUILD
+ cd /home/rjd4/redhat/BUILD
+ rm -rf ackerman-1
+ /usr/bin/gzip -dc /home/rjd4/redhat/SOURCES/ackerman-1.tar.gz
+ tar -xf -
+ STATUS=0
+ '[' 0 -ne 0 ']'
+ cd ackerman-1
++ /usr/bin/id -u
+ '[' 2049 = 0 ']'
++ /usr/bin/id -u
+ '[' 2049 = 0 ']'
+ /bin/chmod -Rf a+rX,g-w,o-w .
+ exit 0
```

What has been done?

- Look in BUILD directory
- Source code unpacked

```
$ ls ../BUILD/
ackerman-1

$ ls ../BUILD/ackerman-1/
Ackerman.cxx  Ackerman.h  GPL  LongArray.cxx  Makefile
ackerman.1  main.cxx
```

`%build`: Compiling the software

- Simplest case
- Just need to run make
- Only instruction in the `%build` section

```
%build
make
```

Running the %build section

- `rpmbuild -bc ackerman.spec`
- Runs `%prep` and `%build` sections
- No scope for manual intervention

```
$ rpmbuild -bc ackerman.spec
Executing(%prep): /bin/sh -e /var/tmp/rpm-tmp.58273
…
Executing(%build): /bin/sh -e /var/tmp/rpm-tmp.58273
+ umask 022
+ cd /home/rjd4/redhat/BUILD
+ cd ackerman-1
+ make
…
```

`%install`: Installing the software

- Just need to install the software now
- Don't install it in the final location
- Install it in the “build root” instead
- The default removes any existing build root
- Many Makefiles come with an “install” target
- We have to do this one manually

The %install section

```
%install
rm -rf $RPM_BUILD_ROOT
install -D ackerman ${RPM_BUILD_ROOT}/usr/bin/ackerman
install -D ackerman.1 ${RPM_BUILD_ROOT}/usr/share/man/man1/ackerman.1
install -D GPL ${RPM_BUILD_ROOT}/usr/share/docs/ackerman-1/GPL
```

- `install` command
- `-D` option build intervening directories
- Building the complete directory structure

Running the `%install` section

- `rpmbuild -bi ackerman.spec`
- Runs `%prep`, `%build`, `%install`
- Various phases of installation to note
- Not just a simple installation!

`%install`: Phase one

```
Executing(%install): /bin/sh -e /var/tmp/rpm-tmp.22989
+ umask 022
+ cd /usr/src/redhat/BUILD
+ cd ackerman-1
```

- Basic set up prior to running what we specified

`%install`: Phase two

```
+ rm -rf /var/tmp/ackerman-root
+ install -D ackerman \
  /var/tmp/ackerman-root/usr/bin/ackerman
+ install -D ackerman.1 \
  /var/tmp/ackerman-root/usr/share/man/man1/ackerman.1
+ install -D GPL \
  /var/tmp/ackerman-root/usr/share/docs/ackerman-1/GPL
```

- This is the section we wrote
- Macros have been expanded
- Environment variables substituted

`%install`: Phase three

```
+ /usr/lib/rpm/brp-compress
+ /usr/lib/rpm/brp-strip
+ /usr/lib/rpm/brp-strip-comment-note
```

- Not part of our script
- Compresses man pages and info pages
- Strips binaries

`%install`: Phase four

```
Processing files: ackerman-1
PreReq: rpmlib(PayloadFilesHavePrefix) <= 4.0-1\
 rpmlib(CompressedFileNames) <= 3.0.4-1
Requires(rpmlib): rpmlib(PayloadFilesHavePrefix) <= 4.0-1\
 rpmlib(CompressedFileNames) <= 3.0.4-1
```

- Automatic processing of all files in the build root
- Dependencies determined
  - Only some can be automatically determined

`%install`: Phase five

```
Checking for unpackaged file(s): /usr/lib/rpm/check-files\
 /var/tmp/ackerman-root

error: Installed (but unpackaged) file(s) found:
   /usr/bin/ackerman
   /usr/share/man/man1/ackerman.1.gz
   /usr/share/docs/ackerman-1-1/GPL
```

- Final phase
- “Unpackaged files”
  - files in the build root but not in the package's file list

%files: List of files in the package

- Give an explicit list of files in the package
- Not automatically generated
- We will see annotations in this section later

`%files`: Our files

```
%files
%defattr(-,root,root)
/usr/bin/ackerman
/usr/share/man/man.1/ackerman.1.gz
/usr/share/doc/ackerman-1/GPL
```

- %defattr — Default attributes
  - (mode, user, group)
- Three files
  - Manual page compressed

Run the installation again

- `rpmbuild -bi ackerman.spec`
- No errors this time
- All files accounted for
- Ready to build the package

Building the package

- `rpmbuild -ba ackerman.spec`
- -a: All
- Binary package
- Source package

Building the package: phase one

```
$ rpmbuild -ba ackerman.spec

"a" for "all".

Wrote: /usr/src/redhat/SRPMS/ackerman-1-1.src.rpm
Wrote: /usr/src/redhat/RPMS/i386/ackerman-1-1.i386.rpm
```

- Runs `%prep`, `%build`, `%install`
- Builds two package files
  - Source
  - Binary

Building the package: phase two

```
Executing(%clean): /bin/sh -e /var/tmp/rpm-tmp.80342
+ umask 022
+ cd /usr/src/redhat/BUILD
+ cd ackerman-1
+ rm -rf /var/tmp/ackerman-root
+ exit 0
```

- Cleans up after itself
- We didn't write this bit
- `%clean` section

We can do better

- We've built a very simple package
- We can improve even this one
- Default locations might vary between systems
  - Manual pages
  - Application binaries
  - Related documentation
- Release 2 !

Using macros — Release 2

- Macros for how a particular Linux lays things out
- Source RPM becomes a more general build facility
- Can also track local policies
- Examples:
  - `%{_bindir}`: `/usr/bin`
  - `%{_mandir}`: `/usr/share/man`
  - `%doc`: Special macro for documentation

Applying a patch — Release 3

- The C++ compiler complains about the code
- Packager wants to fix this
- Patch the source code
- Keep patches distinct from original source
- “Pristine sources”
- `diff` and `patch`

Creating the patch

- Unpack and move twice
  - `ackerman-1.original`
  - `ackerman-1.patched`
- Use `diff -cr` to create patch

```
$ cd ../BUILD
$ rpmbuild -bp ../SPECS/ackerman.spec
$ mv ackerman-1 ackerman-1.original
$ rpmbuild -bp ../SPECS/ackerman.spec
$ mv ackerman-1 ackerman-1.patched
$ emacs
$ diff -cr ackerman-1.* > ../SOURCES/ackerman-gcc3.patch
```

Updating the spec file

```
Release: 3
…
Source0: %{name}-%{version}.tar.gz
Patch1: ackerman-gcc3.patch
…
%prep
%setup -q
%patch1 -p 1
```

- Increment release number
- Add a `Patch1:` line alongside `Source0:`
- Add a `%patch1` line in `%prep` section

Final tweak: Build dependencies

- What did we need to build the package?
- `gcc`/C++, version ≥ 3
- `make`, `install` (can often take these for granted)

```
BuildPreReq: gcc-c++ >= 3.0, make, fileutils
```

Half time summary

- What have we seen so far?
  - Simple source code
  - Simple build
  - Simple patch
  - Simple documentation
- Let's make things more complex !
- POV-Ray — a ray-tracing package

`povray` package

- Packager and author(s) really are distinct this time
- Large package
- Complex build requirements
- Large amounts of HTML documentation
- `http://www.povray.org/`

`povray.spec` — top section

```
Summary: The Persistence of Vision Raytracer
Name: povray
Version: 3.50c
Release: 1
URL: http://www.povray.org/
Source0: povuni_s.tgz
License: Application specific
Group: Applications/Multimedia
BuildRoot: %{_tmppath}/%{name}-root

%description
The Persistence of Vision Raytracer is a high-quality,
totally free tool for creating stunning three-dimensional
graphics.
```

`povray.spec` — `%prep` section

- `povray_s.tgz` unpacks to give directory `%{name}-%{version}/`
- So no need to change this section (yet).
- Run `rpmbuild -bp` just to be sure.

```
%prep
%setup -q
```

`povray.spec` — `%build` section

- Classic installation strategy:
  - Configure
  - Make
  - Install
- Very widespread approach
- `rpmbuild` can work well with this

`%build` — Configure

- `./configure --help`
- Need to set
  - `--prefix=%{_prefix}`
  - `--mandir=%{_mandir}`
  - etc!
- There's a macro to help:
  - `%{configure}`

`%build` — Make

- `./configure` builds `Makefiles`
  - These use `DESTDIR` to specify Build Root.
  - Everything else built in by `%{configure}`
- `make DESTDIR=${RPM_BUILD_ROOT}`
  - Good habit to set `DESTDIR` even here.
  - Some packages build their own installer!
  - Should there be a `%{make}` macro ?

`povray.spec` — `%build` section

- This is all we need for the `%build` section:
  - Configure for system
  - Build knowing Build Root
- Test it!

```
%build
%{configure}
make DESTDIR=${RPM_BUILD_ROOT}
```

Test build *fails*!

- We haven't read the `README` closely enough
- We need to edit the file `src/optout.h`
- This is a patch

```
#define DISTRIBUTION_MESSAGE_1 "This is an unofficial
version compiled by:"
#error You must complete the following
DISTRIBUTION_MESSAGE macro
#define DISTRIBUTION_MESSAGE_2 " FILL IN NAME
HERE........................."
#define DISTRIBUTION_MESSAGE_3 " The POV-Ray Team(tm) is
not responsible for supporting this version."
```

Patching the code

- Build a patch to edit `src/optout.h`
- Apply it in the `povray.spec` file
- Build again

```
Patch1: povray-optout.patch
…
%prep
%setup -q
%patch1 -p 1
```

`povray.spec` — `%install` section

- Trivial `%install` section:
- `make install DESTDIR=${RPM_BUILD_ROOT}`
- No `root`, no worries
- Then look to see what's been installed

`%files` — Categories of files (1)

- Configuration files:
  - `/etc/povray.conf`, `/etc/povray.ini`
- The program itself
  - `/usr/bin/povray`
- The manual page
  - `/usr/share/man/man1/povray.1.gz`

`%files` — Categories of files (2)

- Support files
  - `/usr/share/povray-3.5/include/`
  - `/usr/share/povray-3.5/ini/`
- Example scenes and their support scripts
  - `/usr/share/povray-3.5/scenes`
  - `/usr/share/povray-3.5/scripts/`
- Test script
  - `/usr/share/povray-3.5/tests/test.sh`

%files — Categories of files (3)

- Licence files
  - `/usr/share/doc/povray-3.50c/povlegal.*`
- User manual
  - `…/BUILD/povray-3.50c/doc/html/`
- Other documentation
  - `/usr/share/doc/povray-3.50c/README*`
  - `/usr/share/doc/povray-3.50c/gamma*`

`%files` — The configuration files

- There is a macro to place configuration files
  - `%{_sysconfdir}`
  - `%{configure}` knows it
- Also flag the files as configuration files
  - `%config`

```
%config %{_sysconfdir}/povray.conf
%config %{_sysconfdir}/povray.ini
```

`%files` — The program

- Macro `%{_bindir}` for binaries
- Equates to `/usr/bin` on RHL

```
%{_bindir}/povray
```

`%files` — Support files

- Support files, example scenes, support scripts
- All live under `/usr/share/povray-3.5`
- Directory names refer to whole directory tree
- Use the `%{_datadir}` macro ?
  - Not quite the right definition ?

```
%{_datadir}/povray-3.5
```

`%files` — The documentation

- Use the `%{_mandir}` macro too
- Make extensive use of the `%doc` command
- Directory names refer to whole directory tree

```
%{_mandir}/man1/povray.1.gz
%doc README
%doc README.unix
%doc gamma.gif
%doc gamma.gif.txt
%doc povwhere.txt
%doc povlegal.doc
%doc doc/html
```

One more fix …

- `/etc/povray.ini` refers to `/usr/local/share/povray-3.5`
- Not controlled by `configure`
- We will change this
- Replace `povray.ini` with `povray.ini.in`
- This must contain *configure macros*
- `configure` will create `povray.ini`

Inserting a file

- Second source file
- Manually replace `povray.ini` in `%prep` phase

```
Source1: povray.ini.in
…
%prep
%setup -q
%patch0 -p 1
rm -f povray.ini
cp ${RPM_SOURCE_DIR}/povray.ini.in .
```

Creating `povray.ini.in`

- `configure` uses yet another macro syntax!
- `@`*`variable`*`@`

```
Library_Path=/usr/share/povray-3.5
Library_Path=/usr/share/povray-3.5/include
```

```
Library_Path=@datadir@/povray-3.5
Library_Path=@datadir@/povray-3.5/include
```

And that's it

- We have a working package of POV-Ray
- It doesn't show everything
- Build dependencies ?
- Neither example demonstrates specifying runtime dependencies !

Conclusion

- RPM as a build tool
- Keeps original source apart from patches
- Keeps everything together
- Source RPMs as “enhanced source code”
- Enforces a structure on code building
- Further reading: “Maximum RPM”
  - `http://www.rpm.org/max-rpm/`