Astronomical Data Analysis Software and Systems XII
ASP Conference Series, Vol. 295, 2003
H. E. Payne, R. I. Jedrzejewski, and R. N. Hook, eds.

Source Code Management and Software Distribution using Open Source Technologies

Martin Bly

Rutherford Appleton Laboratory, Chilton, Didcot, Oxfordshire OX11 0QX, United Kingdom

Alasdair Allan

School of Physics, University of Exeter, Stocker Road, Exeter EX4 4QL, United Kingdom

Tim Jenness

Joint Astronomy Centre, 660 N. A'ohōkū Place, University Park, Hilo, HI 96720

Abstract. The Starlink Software Collection (USSC) runs on three different platforms and contains approximately 130 separate software items, totaling over 6 million lines of code. Distribution of such large software systems and installation at multiple remote sites has always been problematic due to the complex web of inter-dependencies such systems invariably generate.

The rise of the Open Source movement has brought standard tools into common use to cope with such large and complex tasks. The RedHat Package Manager (RPM) software is one such which is available for many platforms. We have shown it is possible to automate the distribution and installation of the Starlink Software using RPM. We anticipate that this will vastly simplify installation and package management for Systems Administrators who must support the USSC in production data processing environments.

1. Introduction

The Starlink Software Collection[1] (Bly et al. 2003) is a large collection of software packages comprising subroutine libraries, applications packages and utilities for astronomical data reduction and analysis. The whole collection is governed by a set of interdependencies which not only complicates the process of building the software but also which packages depend on which others at runtime.

Each package has its own `makefile` which contains the rules defining how and in what order the components should be built, and their dependencies on other packages. However, none of the packages are able to trigger the building

[1] `http://www.starlink.ac.uk/`

of another package — this is traditionally done by a master `makefile` which builds the packages in the correct order.

The master `makefile` is maintained by the Software Manager but has two disadvantages — it does not express the package dependencies at run-time, and contains only a simple ordered-list of packages to build — making it difficult to slot a new package into a suitable place in the build sequence. A new approach to managing the build and installation dependencies was needed. We have investigated 'wrapping' the USSC using the RedHat Package Manager (RPM) system (Bailey, 1997) and have shown it is possible to automate the building and distribution and simplify the installation and maintenance of the USSC for Systems Administrators and users.

2. Why Choose the RPM System

The RedHat Package Manager[2] is becoming ubiquitous in the Linux world, having been adopted by most of the major distributions. It is also supported on may other Unix systems including those supported by Starlink. The RPM system has several advantageous features:

Tracking mechanism – it keeps track of installed packages and package version numbers, and all files associated with each package.

Dependency and dependents checking – RPM checks dependencies of the packages it is processing and warns of conflicts and unfulfilled dependencies, and checks for packages that are dependent on those being processed.

Query capabilities – it has a full suite of query capabilities that provide information about the package, its dependencies and status.

Relocation – RPM can install packages in locations other than the one they were intended to go (provided the set is re-locatable).

Adaptability – the RPM system works with existing build systems, and can easily be used to provide a wrapping for existing package build systems, from the simple to the most complex of systems.

Ease of use – installation of patches and updates can be automated.

Open-Source – the RPM package is open-source and runs on many operating systems.

3. Wrapping the Starlink Packages

Each Starlink package has its own makefile and documents which conform to a standard template, although the makefile and documents may have slight variations from the standard. This makes them suitable for processing to provide RPM with the information it requires.

---

[2] `http://www.rpm.org/`

```
[ast]
group=Starlink/Libraries
version=1.5.8
suns=sun210,sun211
requires=htx
buildrequires=sla,ems,chr,sae
fixup=STARBIN/ast_dev
summary=AST - A Library for Handling World Coordinate\
Systems in Astronomy
abstract=The AST library provides a comprehensive range\
of facilities for attaching world coordinate systems to\
astronomical data, for retrieving and interpreting that\
information and for generating graphical output based on it.
```

Figure 1. A typical `depend.ini` dependency file expressing build and installation dependencies.

RPM requires a 'spec' file for a package, to define the various dependencies. These can be generated by hand or automatically by processing a master dependency list. A template dependency file `depend.ini` was created by hand listing the software group, version, and list of SUNs (Starlink User Notes — the documents) and then a Perl script is used to extract summary and abstract information from the package documents. The **buildrequires** and **fixup** lines are then added by hand examination of the `makefile` to see which files are edited and in what way at installation time. **Buildrequires** expresses the additional packages required at build time and **fixup** expresses files that have to be changed at installation.

A dependency file may contain details of more than one package — the abstract extraction script `getabst.pl` processes all the package entries. Where the document listed in the `depend.ini` file does not have the appropriate LaTeX keywords, the keys are left blank. Since this is demonstrating the concept, the system depends on an existing Starlink installation with source files from which it extracts its data.

Once a dependency file is ready, the RPM 'spec' file(s) can be generated. This is the file that controls what RPM does when building and manipulating the package. A Perl script `mkspec.pl` has been produced to interpret the dependency file and generate 'spec' files based on a template for all packages listed in the dependency file.

Existing Starlink installations do not have the source packaged in a single tarball though the makefiles can provide them via the `export_source` target which generates a compressed tarball of the appropriate files. The master `makefile` can generate tarballs for all the packages.

The next step is to create a set of links for each package from an existing USSC installation to the standard location for RPM build directories. A Perl script `mklinks.pl` does this for all the packages in the dependency file, creating links from `/usr/src/redhat/SOURCES` to the source files. Since the default

location is `/usr`, one has to be `root` for this and the remaining steps. Once the links are in place, the 'spec' files are copied to the `/usr/src/redhat/SPECS` directory and the `rpm` program can take over to build the RPMs, both source and installation sets.

4. Results

The resulting RPM files are re-locatable so you don't have to install the packages in the default location (`/star`). Using the `--relocate` command line switch one can instead direct the package to any chosen path — since the normal location is `/star`, one has to be careful where there is an existing (non-RPM) USSC installation. Most packages are easily re-locatable and do not require special tricks to be detailed in the `depend.ini` file. However, some packages do have complex installation requirements and these need to be carefully expressed. The whole collection can be processed to build RPM sets and installations made based on them.

4.1. Problems

RPM itself requires access to the standard RPM database of dependencies needed to track all the files. This is owned by `root` so general users cannot create and install packages using the standard RPM distribution. At the time this work was undertaken, the facilities in RPM to allow alternative databases didn't work, however a distribution that can use a database elsewhere should be possible.

This work is proof of concept. The technology for extracting dependencies and details from the existing build systems is rather basic, and some Starlink packages are under regular development which tends to change the dependencies. Stable packages such as libraries are easier to deal with. Nevertheless we have demonstrated that RPM can be adapted to deal with an alien software management system.

References

Bailey, E. 1997, "Maximum RPM", Sams, ISBN: 0672311054

Bly, M. J., Giaretta, D. L., Taylor, M. B., & Currie, M. J. 2003, this volume, 445