A Management System for Network-Sharable Locally Installed Software: Merging RPM and the Depot Scheme Under Solaris

R. P. C. Rodgers and Ziying Sherwin
– Lister Hill National Center for Biomedical Communications

ABSTRACT

Efficient management of locally installed software is a recurring central theme of system administration. We report here on an experimental merger of two previously independent systems: Redhat's RPM Package Manager (RPM), an open-source database-driven system developed by a major Linux vendor to manage software on a single host; and, an enhanced version of depot, a well-established set of conventions used to manage software that is installed on a server and shared over a network with multiple (possibly heterogeneous) clients. The combination remedies shortcomings in both systems, but to be fully effective, extensions to RPM are required, particularly to its database system. The results of this study point the way toward a second-generation network-distributed version of RPM.

Introduction

Management of the operating system (OS) software on a computer can be time consuming; at many sites, though, the amount of OS software is dwarfed by the amount of additional, locally installed, software arising from a variety of sources. Such software generally provides the services which justify the very existence of the computing facility. The proper installation and maintenance of software is at the heart of system administration, and has a major influence on the utility, reliability, and security of a facility. The work presented here attempts to merge the complimentary features of two open source software installation and management systems: one, known as depot, is a system designed for managing network-shared software; the other, RPM, is designed to manage software on a single host.

Prior Work

One of the earliest attempts to attack the problem of creating and maintaining a network-shared local software repository was the NIST depot scheme [1]. The depot conventions were widely perceived as too complex for smaller facilities run by non-professional administrators, leading to simplified derivatives such as depot-lite [2] and GNU Stow [3]. Colyer, et al. of the Andrew project at CMU offered extensions to the original NIST scheme, including the notion of a software "collection" [4, 5]. Abbey and colleagues at the Advanced Research Laboratory at the University of Texas, Austin (ARL:UT), created a set of perl scripts, opt_depot, which facilitated use of the depot conventions [6, 7]. Other software management schemes that have been developed include STORE [8], the Application Software Installation Server (ASIS) [9], and the /packages scheme employed at Los Alamos National Laboratory (web-based documents for which have been withdrawn from public access). Most of these publically-available systems were developed on UNIX platforms, and attempted to support the sharing of installed software over multiple systems via filesystem-sharing schemes such as network file system (NFS), where clients might be using hardware and OS software different from that of the server. It is difficult to objectively measure the relative costs and benefits of these different approaches; however, STORE and ASIS are much more complex than depot, and none of these systems has been taken up widely outside of its original site of invention.

Commercially-derived systems exist as well. Sun introduced a software management system (pkgadd, pkgrm, pkginfo, ...) as part of its Solaris 2 operating system, using it to install Solaris itself. Functional components that can be installed independently of other components are carved off into their own named "packages," and a list is maintained of where a package's files are installed. Major Linux vendors have all developed software packaging methods with similar intent. These include: Redhat's RPM Package Manager (RPM) [10], which is also used by the French-based MandrakeSoft [11]; Ximian's Red Carpet [12] (and Redhat's equivalent, up2date client); and Debian's Package Management System [13]. Suse's YaST interface appears to be more concerned with OS installation than ongoing local software installation. Caldera International's Volution [14] product is claimed to manage software and other resources over a network of (multi-vendor) Linux hosts. Of these systems, RPM is likely the most widely used, due to Redhat's substantial share of the Linux market as well as to RPM being available as open source for multiple hardware platforms using different UNIX variants. Numerous open source software applications are distributed as RPM packages. With the exception of Volution, these systems are concerned with management of software on a single host.

Finally, some software applications are bundled with their own installation systems; an example is the XPInstall system employed by the Mozilla web client.

How RPM Works

RPM packages exist in two forms: *binary*-type packages contain executable code for a specific hardware/OS combination, whereas *source*-type packages contain the original source code used to generate the executable binary files. The RPM binary package format is well-defined and consists of four sections: the *lead* (a largely abandoned file structure now used to identify the package), *signature* (the PGP and MD5 data used to validate/authenticate a package), *header* (tag-demarcated information about the package), and *archive* (the files constituting the package, compressed with GNU gzip). For a binary-type RPM package file, the RPM command `rpm -i` does the following: it checks for the presence of any other required packages (*dependency* checking) and for potential conflicts (the overwriting of existing files, or the installation of the current or older versions of already-installed packages); it performs any required pre-installation commands; it installs the files associated with the current package, attempting to preserve local modifications made to configuration files; it performs any required post-installation commands; and, it logs all of the file locations and other package information into the RPM database, which is based on Berkeley DB [15].

For a source-type RPM package file, the `rpm -i` command does much less: it unbundles the source code files and specification file (see below), putting the latter in the SPECS subdirectory. One then uses the `rpm -ba` command to build both binary- and source-type packages.

Both binary- and source-type RPM packages have to be created manually. This process employs various directories that are created when RPM is installed, named BUILD, RPMS, SOURCES, SPECS, and SRPMS, and proceeds as follows:

1. Create a RPM specification (*spec*) file, containing sections which address various aspects of installing and uninstalling a package. The spec file begins with a mandatory preamble consisting of tag-demarcated fields containing general information about the package and its creator. It then continues with one or more of the following sections, as appropriate (using the formal name, beginning with a percent sign):
   - %prep (script to prepare for building);
   - %build (compile/build the package);
   - %pre (optional script to prepare for installation);
   - %install (copy requisite files into place);
   - %clean (clean the build directory tree);
   - %verifyscript (script to verify correct functioning of the package);
   - %post (optional script to be executed after installation);
   - %preun (optional script to prepare for uninstallation);
   - %postun (optional script to be executed after uninstallation); and,
   - %files (list of files to be installed).

   Spec files can make use of macros, OS-specific conditionals, and division of the package into subpackages that can be treated differently from one another.
2. Place the spec file in the RPM SPECS subdirectory.
3. Place the source code in the RPM SOURCES subdirectory (the %prep section of the spec file instructs RPM how to unpack this source and place it in the RPM BUILD directory; macros are available for dealing with common types of non-RPM source packaging such as gzipped tar files).
4. Execute the `rpm -ba` command. This unpacks the sources and places copies in the RPM BUILD subdirectory (a helpful behavior for software distributions which tamper with their own source during the build procedures, as the original source is left intact), changes permissions as required, builds the system from source, installs the compiled and other files where specified and generates binary- and source-type RPM packages, placing them in the RPMS and SRPMS subdirectories, respectively. This maneuver will generally have to be repeated at least twice, as the builder will want to save effort by generating the file list section of the spec file by making a recursive directory listing of the files installed from an earlier pass.

As packages are built, the BUILDS subdirectory gets cleaned out by RPM, but files and packages accumulate in the SPECS, RPMS, and SRPMS directories. Several locations require manual cleaning: the SOURCES subdirectory, and a temporary directory in which log files accumulate in association with failed `rpm -ba` commands.

RPM makes use of MD5 checksums to validate both entire packages and individual files within a package (prior to and after installation), and to guide the treatment of an application's configuration files. It also (optionally) employs PGP [16] to create and authenticate digital signatures for packages. RPM provides utilities to search the RPM database to recover information about installed packages, and to easily update and remove them.

The behavior of RPM can be tailored by system-wide and user-specific initialization files. RPM is built on the rpmlib library, which has an Application Programmer's Interface comprising over 60 different functions.

How Depot Works

Henceforth in this paper, depot refers to conventions for software management employed at the U. S. National Library of Medicine (NLM), relying upon modified versions of the ARL:UT perl scripts (originally known as opt_depot, and building upon the earlier work at NIST [1] and CMU [4, 5]).

The method employs the following directory tree on a server: */depot_server/<hardware-type>/<OS-type>/package*, which allows the server to provide files for multiple arbitrary hardware/OS combinations. An individual package exists within its own subdirectory within the above path, and is named for the package and its version number (for example, */depot_server/sparc/SunOS5.8/package/gcc_3.0*).

Within such a package directory, individual files must be installed within the following subdirectories (this list is locally configurable): *app-defaults* (X windows app-defaults files), *bin* (binaries), *html* (HTML documentation), *include* (include files), *info* (TeXinfo files), *javaclass* (Java class files), *lib* (library files), *man* (UNIX manual pages), *pdf* (PDF documentation), and *sbin* (administrative binaries); UNIX manual pages are organized within a package's *man* subdirectory in subdirectories (*man1*, *man1m*, *man3*, . . .) in accord with System V UNIX manual section numbering conventions. If a package has requirements which preclude following this convention (as for example, with some commercial software), it is installed within a subdirectory named *vendor*, and links are made from files or subdirectories within the *vendor* subdirectory to the appropriate *app-defaults* . . . *sbin* subdirectories.

On a depot client, a directory with a name of the form */depot_mount/<server-name>/package* is present, where *<server-name>* represents a particular depot server. One such directory is present for each depot server that is providing software to this client. Package subdirectories from each server's */depot_server/<hardware-type>/<OS-type>/package/* directory are mounted into the client's corresponding */depot_mount/<server-name>/package* directory, using a network file-sharing scheme such as NFS.

The client also has a */depot* directory, which contains subdirectories as listed above for the server package directories (*app-defaults* . . . *sbin*, excluding *vendor*). Entire packages from */depot_mount/. . .* are symbolically linked into */depot/package* (for example, */depot/package/ gcc_3.0*). In addition, the files appearing within a package are linked into the corresponding directories of */depot* (for example, */depot/package/gcc_3.0/bin/gcc* is linked to */depot/bin/gcc*). Finally, if a package must write into host-specific files (for example, log or database files), these are placed in the directory */var/depot/<package-name>*

A server (or standalone host) may act as a client to itself, and possess */depot_mount*, */depot*, and */var/depot* directories as well, although we employ symbolic links rather than NFS to provide files to the */depot_mount* tree in that case.

The behavior of depot on a client is controlled by configuration files: */depot/site* controls the mounting of files from multiple depot servers; */depot/.exclude* prevents specified packages from being linked into */depot/package*; and, */depot/.priority* controls which packages have priority for linking into */depot/{bin, html, . . . sbin}* when there are name conflicts between individual files coming from different packages.

Thus far we have described *shared* depot packages, which a server provides to one or more depot client hosts. Packages that are specific to a client (for example, node-locked commercial products, or software that requires hardware that is specific to the client) can be installed directly into */depot/package* as a *local* package; its files are linked into */depot/{bin, html, . . ., sbin}* along with the shared packages, subject to the same configuration files.

Although this description may make depot seem complicated, in practice it is not. The main labor is learning how to make a new software package conform to depot's package directory structuring conventions. We often employ script wrappers to encapsulate actual binaries, which allows us to set up various environment variables for a given application, freeing the user from having to do so. The perl scripts of the ARL:UT depot system automate maintenance of the underlying system of links.

Shared and Complementary Features of Depot and RPM

Both RPM and depot provide structured, disciplined means of managing software installations. Not surprisingly, they address many issues in common, but with varying degrees of rigor:

1. Both RPM binary-type and installed depot packages can be easily installed on additional hosts. In the case of RPM, this is done by copying the binary-type package to the target host and using the rpm -i command; in the case of depot, this is done by copying the directory for the installed package from */depot_server/<hardware-type>/<OS-type>/package* (for a shared package), or */depot/package* (for a local package) to the target host (either client or server) and running a depot script to update the requisite symbolic links (cron can be used to automate the latter). Both reduce the number of times that a given piece of software must be installed from scratch: the original installer/packager can better afford to focus upon installing the package correctly, as it must be done only once.
2. Both allow easy uninstallation of packages. With depot, packages can be cleanly removed by a single rm command followed by execution of a depot script to clean up unresolved symbolic links (using cron to automate this if

desired). RPM automates the taking of additional actions through its uninstallation scripts.

3. Both methods attempt to document the installed software. At NLM, each depot package has in its root directory a manually constructed file named README.LOCAL, which includes a header containing defined fields describing the package, its author and origin, and its installer. There is considerable overlap with the information contained in the RPM spec file.
4. Both systems can support multiple versions of a given package, if and only if the package in question does not require the use of absolute file paths; with RPM, this requires use of the `--relocate` flag.

The two systems compliment one another in a number of important respects:

1. Applicability over a network. Unlike RPM, depot is inherently designed to accommodate use of a network. Multiple depot servers can provide redundancy (through NFS roll-over). Collaboration is facilitated, as different workgroups can specialize in particular types of software on their own depot server, and share the results within their larger organization.
2. Dependency checking. Under NLM depot, the installer places dependency information in the README.LOCAL file that gets installed with the package. This manual process is error-prone. RPM employs the UNIX ldd command to determine which libraries the package requires (its ''dependencies''), and logs this information into the RPM database. At installation and uninstallation time, dependencies can be handled rigorously.
3. Package documentation. The README.LOCAL file remains as a human-readable document with the installed package; various components of the RPM spec file become part of the RPM database record, but the spec file does not remain online as part of the installed package. Unlike README.LOCAL, a spec file can contain procedural components (various scripts) that get automatically invoked at the appropriate moment. Unlike the spec file, README.LOCAL includes (manual) instructions for host- and user-specific installation steps that a package may require to become fully functional, reflecting the multi-host networked nature of depot. For example, the GNU findutils system requires that a database be built on each client host, and Sun's StarOffice system requires that each user run an initialization script prior to using the package. README.LOCAL also contains transcripts of the installation procedure and copies of email correspondence with other parties in connection with the software.
4. User environment. Under depot, search paths are short and simple (*/depot/bin*, */depot/man*, . . .).

The Experimental Environment

The study was done using an UltraSPARC 2 as the depot server, and an UltraSPARC 2 and two UltraSPARC 60 machines as depot clients. The machines were operating under Solaris 2.[5-8]. We standardized the naming and NFS automounting schemes used by depot as described above. At our request, Abbey and colleagues added new features to the original ARL:UT opt_depot scripts: the ability to mount software packages on a client from multiple depot servers; and, improved configurability of the perl scripts. We installed and used depot routinely for a period of four years, successfully supporting as many as two different versions of Solaris concurrently. At the time of writing, the depot server contained 321 shared packages and 21 local packages for Solaris 2.8. Source for the SPARC-compatible version RPM 4.0.2 was obtained from the web site http://www.rpm.org . Eighteen lines of the code had to be modified to get it to compile under Solaris 2.8 using gcc 3.0. RPM was installed as a shared depot package on the depot server, with the RPM database files placed in */depot/package/rpm_4.0.2/vendor/var/lib/rpm*. Information about all shared packages is logged into this database. On each client, the RPM database is installed in */var/depot/rpm_4.0.2/local_db*, and information about local packages is placed there. The need for and limitations of using two databases is discussed below.

To allow RPM dependency checking to operate, we employed a script, vpkg-provides2.sh, provided with the Solaris version of RPM, which uses the information provided by Sun's proprietary package database to create entries for ''virtual'' RPM packages (there were 564 such packages on our depot server). We then installed a number of packages using RPM, while following the depot conventions: a library (libpcap 0.4); an application depending upon that library (snort 1.7); a self-standing source application (wget 1.6), and, a commercial pre-built binary application (netscape 4.77).

Results/Discussion

The RPM/depot merger experiment suggested a number of technical directions for future work:

1. The creation of ''virtual'' RPM packages from Sun's proprietary package format is slow, and the script must be rerun when additional Sun packages are installed. Ideally, Sun should use RPM/depot; otherwise, the script should be improved, and wrappers created for the Sun package tools to invisibly and reliably integrate them with RPM/depot.
2. RPM and depot functionalities should be coupled. RPM spec file macros could be used to invoke requisite depot scripts during installation/uninstallation, and to automate actions now taken manually. Alternatively, the scripts could be invoked directly from RPM source code.

3. Capabilities of the RPM spec and depot README.LOCAL files should be merged; in particular, RPM needs to know how to trigger the host- and user-specific initialization (and uninstallation) steps required by some software. Other helpful aspects of depot as used at NLM: inclusion of installation transcripts and related email correspondence, and leaving a human-readable document with the installed package. Transcripts of the build could be created by using spec file macros to capture the process in a subshell, redirecting the output to a file which is then incorporated into the package documentation. It would be ideal if the README. LOCAL/spec merger resulted in redundancy, such that the README.LOCAL files could be used to rapidly rebuild a corrupted RPM database, and to manually manage packages in an emergency.
4. Modifications to RPM to allow dependency checking over a network. RPM can only use one database at a time; either the default one, or one supplied following the rpm command line argument `--dbpath`. As described earlier, we installed RPM with two databases: one database (on the server) containing the information for shared packages (shared by all hosts over the network), and a second database for local packages (on each client). Most packages can be shared, and use of the shared package database will succeed much of the time, as it contains records for most of the Solaris virtual packages as well as the shared depot packages. Installation of local packages will fail more often, as the local package database will not contain information about shared packages upon which local ones may depend. This problem could be partially solved by modifying RPM, supporting the searching of a comma-separated list of databases instead of just a single database (an enhancement that has already been requested by other RPM users; see bugzilla request #4137 on the Redhat web site). This is preferable to error-prone work-arounds such as cloning the content of the shared database onto the local databases. This solution is partial, however, because of the way in which depot handles multiple versions of packages and file name collisions, through the depot configuration files. RPM should be able to deal with these files, so as to know how to get to the files a new package needs.
5. User-defined tags are not supported for the RPM spec file and database; such tags would be useful for local extensions to the database.
6. Other RPM enhancements could be achieved more efficiently by means of modest code modifications. When creating an installable RPM binary-type package, its installation path must be configured for either a local or shared depot package. RPM could be altered to allow the package to be designated to be installed as shared or local, and alter the installation automatically. (currently, one can use the RPM `--relocate` command-line option, with appropriate path argument, to install a shared package as a local package, or vice versa).
7. One of depot's strengths is its ability to offer files for a variety of operating systems, whereas RPM is UNIX-specific. If RPM were to be made to support non-UNIX target operating systems, it would have to know how to check for dependencies and to build packages on the target OS. This would be a considerable undertaking compared to the changes suggested earlier. However, one could still concurrently employ RPM/depot for enhanced control of UNIX software, while continuing to use depot as at present to support the other OSs.

Conclusion

Automation and standardization are two means of reducing the considerable costs of administering software on multiple hosts. Depot is a highly efficient means of managing local software, even on standalone systems; its benefits are compounded when used with multiple networked hosts, an environment in which one can use both network-shared and local (client-specific) depot packages. RPM is better at certain things such as documenting packages by database, and dependency checking, but is currently designed for use on a single machine. Our experiment in merging RPM and depot is a qualified success, in that most of our software can be installed and used with RPM/depot without modifying RPM or depot. A fully functioning RPM/depot system, however, requires slight modifications to RPM source code: most importantly, to allow the searching of multiple RPM databases to support dependency checking for local depot packages; less importantly, to couple execution of depot scripts to the execution of RPM commands, and to support the automatic reconfiguration of paths required when installing as a local depot package one that was originally designed to be shared (or vice versa).

Such modifications seem a slight price to pay in order to turn RPM into a network-based software management tool. It would also make the system more attractive as a packaging system for use by other UNIX/Linux vendors. The existence of a single widely-shared system could save time for administrators, by allowing the creation of ftp- and Web-accessible archives of RPM/depot packages for Solaris (and other) platforms, greatly reducing installation effort.

Modifications to RPM to support non-UNIX clients would be more complex than the ones just described, and are harder to justify.

Code Availability

The code and documentation for RPM/depot for Solaris will be available at the time of the conference, from: http://www.etg.nlm.nih.gov .

Acknowledgements

We wish to thank Jeff Johnson of Redhat for invaluable assistance with installing and using RPM under Solaris. Jonathan Abbey of ARL:UT assisted us in understanding and using his opt_depot scripts, and made helpful extensions to them at our request. Both provided helpful remarks about the manuscript, along with Jules Aronson of NLM and Nelson Beebe of the University of Utah.

Funding Sources & Copyright

Both authors are functioning as paid employees within a U. S. government research laboratory and produced this work as part of their routine duties. No additional funding was involved.

Biographical Notes

R. P. C. Rodgers (rodgers@nlm.nih.gov) works in biomedical informatics at the Lister Hill National Center for Biomedical Communications (LHNCBC), where he heads the Emerging Technologies Group. He received a B.A. From Harvard College in 1972, a M.D. from the University of Utah College of Medicine in 1976, and postdoctoral training from the University of London, University of Louvain, the National Cancer Institute, and the University of California, San Francisco (UCSF). He served on the faculty at UCSF prior to joining LHNCBC, a research arm of the U. S. National Library of Medicine (NLM). At NLM he became an early and active exponent of the World Wide Web, creating and running NLM's web services for the first two years of their existence. He has participated in a number of IETF working groups, and served as a founding member of the International World Wide Web Conference Committee and founding chair of the NSF/NCSA World Wide Web Federal Consortium.

Ziying Sherwin (sherwin@nlm.nih.gov) received a B.S. in Computing & Engineering from Zhejiang University in 1996, and a M.S. in Computer & Information Science from the University of Delaware in 1999. She has worked Bell for Atlantic, and joined the Emerging Technologies Group at LHNCBC in 2000.

References

[1] Manheimer, K., B. Warsaw, S. N. Clark, and W. Rowe, "The Depot: A Framework for Sharing Software Installation Across Organizational and UNIX Platform Boundaries," *LISA IV*, http://www.forwiss.uni-passau.de/archive/marchiv/systemverwaltung.html, 17-19 October, 1990.

[2] Rouillard, J. P., and R. B. Martin, "Depot-Lite: A Mechanism for Managing Software," http://www.usenix.org/publications/library/proceedings/lisa94/martin.html, *LISA VIII,* 1994.

[3] Glickstein, B., "GNU Stow," http://www.gnu.ai.mit.edu/software/stow/ , http://www.gnu.ai.mit.edu/software/stow/manual.html .

[4] Colyer, W., and W. Wong, "Depot: A Tool for Managing Software Environments," *LISA VI,* http://andrew2.andrew.cmu.edu/depot/depot-lisaVI-paper.html, 1992.

[5] "The Depot Configuration Management Project," Carnegie Mellon University, http://andrew2.andrew.cmu.edu/ANDREWII/depot.html, http://asg.web.cmu.edu/depot/depot.html.

[6] "opt_depot," ARL, University of Texas at Austin, http://www.arlut.utexas.edu/csd/opt_depot/opt_depot.html .

[7] Abbey, J., "The Group Administration Shell and the GASH Network Computing Environment," *LISA VIII*, http://www.arlut.utexas.edu/csd/gash_docs/lisa_paper/paper.html, September, 1994.

[8] Bakken, S. S., A. Christensen, T. Egge, and A. H. Juul, "STORE," Norwegian University of Science and Technology, http://www.pvv.unit.no/˜arnej/store/storedoc.html .

[9] Defert, P., S. Gouache, A. Peyrat, and I. Reguero, "ASIS User's and Reference Guide, Version 3.95," http://consult.cern.ch/writeups/asis/node1.html, CERN, 1997.

[10] Bailey, E. C., *Maximum RPM,* SAMS Publishing http://www.rpm.org/max-rpm/index.html; http://www.rpmdp.org/rpmbook, 1997.

[11] Bégnis, C., G. Cottenceau, G. Lee, and T. Vignaud, *Mandrake RPM HOWTO, vol. 1.1*, http://www.linux-mandrake.com/en/howtos/mdk-rpm/ .

[12] Ximian, "Red Carpet," http://www.ximian.com/products/ximian_red_carpet/ .

[13] Debian, "Package Management System," http://www.debian.org/doc/FAQ/ch-pkg_basics.html, http://www.debian.org/doc/packaging-manuals/developers-reference/ .

[14] Caldera International, "Volution," http://www.caldera.com/products/volution/ .

[15] Sleepycat Software Inc., *Berkeley DB*, New Riders Publishing, Indianapolis, 2001.

[16] Garfinkel, S., *PGP: Pretty Good Privacy*, First Edition, December, 1994.