R Help Desk

Package Management

Uwe Ligges

Preface

A huge number of packages dedicated to wide range of applications is available for R. Writing this article, there are more than 300 packages: base and recommended packages, CRAN[1] packages, packages from the Omegahat[2] and BioConductor[3] projects, as well as a couple of other packages. This shows that package management is really important. A recent summary of existing packages is published in the R FAQ by Hornik (2003).

Convenient tools for developing, checking, installing, updating and removing packages are available in R. Therefore, users and developers can easily handle packages. This is one of the main reasons for R's rapid growth.

In this article, the procedures to install, update, and manage packages in (more than) one library are described. In particular, this article summarizes and extends the existing documentation on package management.

Documentation on package management

Package management is described in several documents, because handling packages is fundamental for users of R. The "R Installation and Administration" manual (R Core, 2003a) is the main resource for users who want to install packages. Among other topics it explains the terminology. In particular, it is important to distinguish between a *package* and a *library*:

> "A package is loaded from a library by the function `library()`. Thus a library is a directory containing installed packages; ..."

"The R FAQ" (Hornik, 2003) devotes a section to package management. The complementary "R for Windows FAQ" (Ripley, 2003) explains a couple of specific Windows issues related to package management. For users of the Macintosh, the "RAqua FAQ" (Iacus, 2003) might be extended by a corresponding section in a future versions.

The manual "Writing R Extensions" (R Core, 2003b) is devoted to the development of packages, hence of minor interest for the user who tries to install or update a package.

[1]http://cran.r-project.org
[2]http://www.omegahat.org
[3]http://www.bioconductor.org

Libraries

If you have installed a released version of R, the base and recommended packages can be found in the main library: `R_HOME/library`, where `R_HOME` denotes the path to your version of R, e.g. `/usr/local/lib/R` or `c:\Programs\rw1081`.

By default, further packages will be installed into `R_HOME/library` as well.

It may be sensible to have more than one library, depending on the purpose of the system R has been installed on.

Let's consider the following situation: R has been installed on a server, and the regular user does not have any write access to R's main library. There is no need to ask the administrator to install a package from CRAN. The user can easily install the package into another library to which he has write access, e.g.: `/home/user/myRstuff/library`. Moreover, it might be convenient to have a third library containing packages the user is maintaining himself, e.g.: `/home/user/myRstuff/mylibrary`.

In the example given above, Unix file paths have been used, but the idea is applicable to arbitrary operating systems. As an example, consider our department's Windows system: R is installed on a network share, say `s:\R` (no write access for the regular user). If a user has write access to, say, `d:\user`, then `d:\user\myRstuff\library` might be a good choice for a library containing private packages.

Since R does not know about any further libraries than the default one, the environment variable `R_LIBS` must be set appropriately. This can be done in the usual way for setting environment variables on your operating system, or even better by specifying it in one of the environment files R processes on its startup, see `?Startup` for details. For example, in order to tell R that a second library `/home/user/myRstuff/mylibrary` is available, you might want to add the line

```
R_LIBS=/home/user/myRstuff/mylibrary
```

to the file '.Renviron' (see `?Startup`). More libraries can be specified as a semicolon separated list.

During R's startup, the library search path is initialized by calling the function `.libPaths()` on the directories defined in `R_LIBS` (the main library is always included). If `.libPaths()` is called without any arguments, the current library search path is returned.

Source vs. binary packages

Packages may be distributed in source or binary form.

Source packages are platform independent, i.e. independent of hardware (CPU, ...) and operating system, as long as the package author does not implement platform specific details — and, of course, as long as R is available on the particular platform. Installing source packages requires a couple of tools to be installed (e.g. Perl, and depending on the package: C compiler, Fortran compiler, ...). In order to provide packages to a large group of users, developers must submit their packages in source form to CRAN[4]

For users working on Unix-like systems (including Linux, Solaris, ...), it is quite easy to install packages from source, because their systems generally have (almost) all of the required tools installed.

Binary packages are platform specific. They may also depend on the R version — in particular, if compiled code has been linked against R. To install a binary package, no special tools are required, because the shared object files (as known as DLL[5] under Windows), help pages (HTML, text, ...), etc. already have been pre-compiled in a binary package.

In order to make package installation as convenient as possible for the user, binary versions of CRAN packages are provided for some platforms (in particular these are currently MacOS X, SuSE Linux, and Windows) and the most recent versions of R. For example, binary versions of packages for Windows regularly appear within two days after the corresponding source package on CRAN — given there is no special dependence or need for manual configuration.

By convention, source packages regularly have the file extension '.tar.gz', binary packages for Windows '.zip', and those for Linux '.deb' or '.rpm'. Both extensions '.tar.gz' and '.zip' indicate archives files (an archive contains files in compressed form) packed by different tools. Hence, it's a fast but also unsafe way to decide by looking at file extensions: Omegahat source packages still end in '.zip', MacOS X binary packages end in '.tar.gz'. You can always check whether a package contains folders `help` and `html`. If it does, it is almost certainly a binary package.

Installing and updating source packages

If you have more than one library, you need to specify the library to/in/from which a package is to be installed, updated, removed, etc. The syntax to install source packages on Unix using the command line is

```
$ R CMD INSTALL -l /path/to/library package
```

The part used to specify the library, `-l /path/to/library`, can be omitted. In that case the first library in the environment variable `R_LIBS` is used if set, otherwise the main library. Note that file '.Renviron' is not read by `R CMD`. A Section describing the syntax for Windows can be found below.

For example, a source package `mypackage_0.0-1.tar.gz` can be installed to the library `/home/user/myRstuff/mylibrary` using the command

```
$ R CMD INSTALL -l /home/user/myRstuff/mylibrary
    mypackage_0.0-1.tar.gz
```

In most cases an alternative approach is much more convenient: Packages can be downloaded and installed from within R using the function `install.packages()`. In order to install a package **package** from CRAN into the library `/path/to/library` from an appropriate CRAN mirror[6] (which might not be the US mirror given below), the call looks like:

```
R> options(CRAN="http://cran.us.r-project.org/")
R> install.packages("package",
R+     lib = "/path/to/library")
```

Analogously to the command line version, the specification of a library may be omitted when the package is to be installed into the first library of the library search path. More specifically, `install.packages()` looks for available source packages on CRAN, downloads the latest version of the requested source package, and installs it via `R CMD INSTALL`.

The function `update.packages()` helps to keep packages on a system up to date. According to R Core (2003a), `update.packages()`

> "... downloads the list of available packages and their current versions, compares it with those installed and offers to fetch and install any that have later versions on CRAN."

Note that, instead of the current package management tools, `packageStatus()` (see its help page `?packageStatus` for details) is intended to become the default package manager for future versions of R. Both `packageStatus()` and the package **reposTools** from the BioConductor project are expected to become great tools for modern package management in R.

On Windows, `install.packages()` and `update.packages()` are used to install binary versions of packages rather than source packages (see below).

See R Core (2003a) for an explanation how to remove packages.

[4]Note that submissions of any binary packages to CRAN will not be accepted.
[5]dynamic link libraries
[6]For a list of mirrors see http://cran.r-project.org/mirrors.html

Package management on the Macintosh

Recent versions of R only support MacOS X. On this operating system's command line, R behaves as under Unix.

For users of RAqua, the GUI has menus for the installation and update of source and binary packages. Also, there are separate menus for CRAN packages and packages from the BioConductor project, as well as for packages from local directories.

Functions like `install.packages()` for managing source packages behave as on Unix, but there are also functions for managing binary packages available in RAqua. Instead of ending on `.packages`, names of these functions end on `.binaries`. Hence, `install.binaries()` installs binary packages, etc. A corresponding binary repository for RAqua is available on CRAN at `yourCRANmirror/bin/macosx/MajVer/` where MajVer is, e.g., '1.8' for R-1.8.1.

Package management on Windows

Package management on Windows is special in a way: On the typical Windows system many of the required tools to install packages from source are missing. Moreover, the operating system's command shell is quite different from typical Unix command shells.

Therefore, R's command line tools are slightly different on Windows. For example, the command to install the source package called **mypackage** in file 'mypackage_0.0-1.tar.gz' to the library path `c:\myRstuff\mylibrary` using the command line is:

```
c:\> Rcmd INSTALL -l c:\myRstuff\mylibrary
       mypackage_0.0-1.tar.gz
```

How to collect the required tools and how to set up the system in order to be able to install packages from source is described in the file `R_HOME/src/gnuwin32/readme.packages` within the R sources. It is highly recommended to read that file very carefully line by line, because almost every line is important to get `Rcmd INSTALL` working. Those who have already compiled their version of R from source also have set up their system and tools correctly to install source packages.

The R functions `install.packages()`, `update.packages()` and friends work differently on Windows than on Unix (with the same syntax, though). On Windows, these functions look at the list of *binary* packages (rather than of source packages) and download the latest version of the requested binary package.

The repository in which these functions are looking for binary packages is located at `yourCRANmirror/bin/windows/contrib/MajVer/` where MajVer is, e.g., '1.8' for R-1.8.1. The ReadMe files in these directories (and also the one in `yourCRANmirror/bin/windows/contrib/ReadMe`) contain important information on availability of packages and how special circumstances (e.g. what happens when packages do not pass the quality control, `Rcmd CHECK`) are handled.

If you are using R's GUI ('RGui.exe') on Windows, you will find a menu called "Packages" that provides a GUI interface to the functions `install.packages()`, `update.packages()` and `library()`. Besides the CRAN repository, the BioConductor repository is also accessible via the menu, as well as the installation of local files.

Note that it is not possible to install to another library than that one in the first place of the library search path (`.libPaths()[1]`) using the menu. Consider using `install.packages()` directly in order to install to a different library. For example, in order to install a binary version of the package **mypackage**, available as a local zip file `c:\myRstuff\mypackage_0.0-1.zip`, into the library `c:\myRstuff\mylibrary`, we can use

```
R> install.packages(
R+   "c:/myRstuff/mypackage_0.0-1.zip",
R+   lib = "c:/myRstuff/mylibrary", CRAN = NULL)
```

Note that you have to escape the backslash, or use normal slashes as above, to specify a path within R for Windows.

Bibliography

Hornik, K. (2003): *The R FAQ*, Version 1.8-28. http://www.ci.tuwien.ac.at/~hornik/R/ ISBN 3-900051-01-1. 37

Iacus, S. (2003): *RAqua FAQ/DOC*, Version for R 1.8.x. http://cran.r-project.org/bin/macosx/RAqua-FAQ.html. 37

R Development Core Team (2003a): *R Installation and Administration*. URL http://CRAN.R-project.org/manuals.html. ISBN 3-900051-02-X. 37, 38

R Development Core Team (2003b): *Writing R Extensions*. URL http://CRAN.R-project.org/manuals.html. ISBN 3-900051-04-6. 37

Ripley, B.D. (2003): *R for Windows FAQ*, Version for rw1081. http://cran.r-project.org/bin/windows/rw-FAQ.html. 37

Uwe Ligges
Fachbereich Statistik, Universität Dortmund, Germany
ligges@statistik.uni-dortmund.de