R Packages

November 23, 2003

1 Introduction

All the information you need is here `http://cran.r-project.org/doc/manuals/R-exts.pdf`.

Today we will:

- Give examples of why/when make an R package.
- Go through a simple example of making an R package in a Unix environment.
- Describe why it's a bit different using Microsoft Windows.
- Briefly describe some complicated situations.

One can say that all functions in R come from a package (sometime people call them libraries) Most functions you use are in the base package.

Remember functions are objects like any other. To see where R searches for objects type

```
> search()

 [1] ".GlobalEnv"      "package:tools"   "package:methods" "package:ctest"
 [5] "package:mva"     "package:modreg"  "package:nls"     "package:ts"
 [9] "Autoloads"       "package:base"
```

you will see a list of *environments* where R searches for objects. The last one is the base defined by the base package.

There are some packages that are automatically loaded in R because functions there are very commonly used.

You can attach other packages with more functions. For example by typing `library(splines)`

```
> try(args(ns))
> library(splines)
> args(ns)

function (x, df = NULL, knots = NULL, intercept = FALSE, Boundary.knots = range(x))
NULL
```

Notice, `ns` doesn't exist until we load the package. `splines` comes with R but many other commonly used packages you need to download from `http://cran.r-project.org`. You can use the function `install.packages` to automatically download and install (you need to have a web connection for this).

What if you have your own set of functions not available in CRAN? You can have them all in a file and source them in, but you can also create a package. It's very easy.

Reason for creating a package:

- Provides a way to organize and document your functions.
- Makes it easier for you to share your software.
- Makes it very easy to load binaries (C or Fortran).
- You can also share data through packages.

The first step in creating a package is to write the *source code*. To see some examples of package source code you can look (on athena) at `/users/faculty/ririzarr/madman/Rpacks` for all the Bioconductor source code. You can also download source code for any CRAN package.

You can do it in windows but it is much easier on Unix. So we will go through an example assuming you are working on a Unix system.

2 The parts of a package

Start with a name (avoid special characters and remember windows is not case sensitive).

Create a directory with that name

In that directory you can create directories called: `R, man, data, demo, exec, inst, src, tests`

But only the first two are necessary.

Here is are short descriptions (see How to CRAN document for more)

R: R code. I try to have one file per important function and always use the `.R` extension.

man: documentation goes here.

data: you guessed it! The data should be saved as `.rda` files. Once the package is loaded you can use `data` to load data.

demo: R code that shows a demo.

exec: code for Perl and other system tools.

inst: folder with files and folders that get installed with the package. Can contain anything, but the most common is doc which usually has a *vignette*.

src: contains C code that gets compiled on installation (or on windows on package creation)

tests: R code that runs on R check (we will explain)

You also need an `DESCRIPTION` file. Here is an example.

```
Package: affycomp
Version: 1.2.1
Date: 2003-11-1
Title: Graphics Toolbox for Assessment of Affymetrix Expression Measures
Author: Rafael A. Irizarry <rafa@jhu.edu>
with contributions from
Simon Cawley <simon_cawley@affymtrix.com>,
Zhijin Wu <zwu@jhsph.edu>
Maintainer: Rafael A. Irizarry <rafa@jhu.edu>
Depends: R (>= 1.6.1), modreg, Biobase (>= 1.1.0)
Description: The package contains functions that can be used to compare expression measures for Affyme
License: GPL version 2 or newer
```

3 Simple example

Lets make a quick package. Just one function and some data. If you want your collaborator to see a picture and be able to change it. Will call the function and the package `doit()`.

The done package will be put up on the class web page: `http://www.biostat.jhsph.edu/~bcaffo/statcomp/index.html`

The R code is usually already made. The hard part is the documentation. The function `prompt` is very useful!

R has a special feature to make sure you did things right: `R CMD check`

To create the package you can use: `R CMD build` This creates a source code archive (tared and zipped) that R on a unix system will, in general, install. This includes compilation of C or Fortran code.

But what about windows? Most windows systems don't have C compilers. Two recommendations:

- If your package has no C code you can untar and unzip the package archive and then use winzip (`http://www.winzip.com/`) to compress it. It will be ready for install on msft windows.
- If you have C code, then you'll need to compile your code on windows. To do this you will need some unix tools running on your windows machine. cygwin doesn't work for this. Instead, follow Professor Brian Ripley's instructions step by step: `http://www.stats.ox.ac.uk/pub/R/rw-FAQ.html`

4 More complicated stuff

If your library needs to define things when loading you can use the function `.First.lib`. You usually load up C binaries here. Tradition is that one calls the file that does this `zzz.R`.

If you need to look for system library location, etc... you can use configure scripts.

If you are using the S4 Classes and Methods, there are some issues you need to learn about.

All this is and more is described in detail in the "Writing R Extensions" document.