Making an R package

R.M. Ripley

Department of Statistics
University of Oxford

2012/13

Why or why not?

Pro:

Altruism You have benefited from others' work: give something back

Convenience Simple way for others to use your functions

Extra testing Others may find a few bugs

Con:

Tedium Your package will be checked and must comply with the standards

Convenience It can be slightly more difficult to adapt a package than the pieces

How?

- Use function `package.skeleton` to create the package
- You will need:

package name	consult list on CRAN to avoid clashes
functions	
data objects	possibly needed for examples

- For portability, be careful with the names of objects and files. No special characters and do not rely on case to make them unique.
- The functions should be in source files, (with filenames ending in ".R"), or in an R workspace, but not a mixture.
- `package.skeleton` will create
  - a package directory structure
  - DESCRIPTION file
  - outline help files for editing.
  - "Read-and-delete-me": a file of helpful instructions

Details of `package.skeleton`

```
package.skeleton(name = "anRpackage", list,
    environment = .GlobalEnv,
    path = ".", force = FALSE, namespace = FALSE,
    code_files = character())
```

- Define the objects using one of:

list	character vector of names of objects
environment	name of an environment e.g. `.GlobalEnv`
code_files	character vector of names of source files

- Other arguments

name	required
path	where to create the package directories
force	whether to overwrite an existing set of directories
namespace	no longer used, and hence deprecated.

What next?

`Read-and-delete-me` contains:

* Edit the help file skeletons in **`man`**, possibly combining help files for multiple functions.
* Edit the exports in **`NAMESPACE`**, and add necessary imports.
* Put any C/C++/Fortran code in **`src`**.
* If you have compiled code, add a **`useDynLib()`** directive to **`NAMESPACE`**.
* Run **`R CMD build`** to build the package tarball.
* Run **`R CMD check`** to check the package tarball.

Read “Writing R Extensions” for more information.

We will consider each in turn. but first consider the necessary editing of the **`DESCRIPTION`** file.

DESCRIPTION File

The skeleton DESCRIPTION file contains:

```
Package: mypkg
Type: Package
Title: What the package does (short line)
Version: 1.0
Date: 2008-12-19
Author: Who wrote it
Maintainer: Who to complain to
                  <yourfault@somewhere.net>
Description: More about what it does (maybe more
                           than one line)
License: What license is it under?
```

DESCRIPTION file: straightforward items

NB: any continuation lines in this file must start with a space or tab.

Package The name you give the package

Type `Package`. Other types are very different objects

Title To be inserted. A single line

Description To be inserted. A single paragraph

Author To be inserted. Can be several names

Maintainer To be inserted: one name, plus a valid email address

Version should relate to the current release

Date Optional, should relate to the current release

DESCRIPTION file: further items

License Required. One of:

- A string such as "GPL-2" referring to a file in the `share\licenses` subdirectory of the R package
- The name of an entry in the file `share\licenses\license.db`
- the words "File LICENSE" or "File LICENCE" with an appropriately named file in the top directory of the package
- the string "Unlimited", meaning no restrictions apply

LazyData If you have data objects, add this entry and set to yes. This will remove the need to use `data()` before using the dataset.

DESCRIPTION file: further important items

Depends Comma-separated list of packages which are required to be attached in order to run your package. May include details of the version required: e.g.

```
Depends:  R (>= 2.7.0), tcltk
```

Suggests Similar to `Depends`, but only necessary for examples and vignettes, so use of the package would be restricted but not impossible if these packages are unavailable.

Imports Lists packages whose name spaces are imported from using the `import` directive in the `NAMESPACE` file (or by using `"::"` or `":::"` within the package), but which do not need to be attached. (`":::"` allows access to hidden objects.)

There are other optional fields: for details see the manual *Writing R Extensions*

Help Files

- Written in R documentation format, stored in `.Rd` files
- The format resembles LaTeX.
- Emacs is `Rd` aware, if you have `ESS`
- Converted to HTML, LaTeX, and text versions when you create the package.
- Slightly different items for packages, functions or datasets.
- `prompt` will generate a skeleton page for an individual object.
- `promptPackage` will generate a skeleton page for the complete package.
- Much more information on the format in the manual *Writing R Extensions*

Help file format: 1. Functions

\name{name}	Name of help file
\alias{topic}	Often multiple entries: one for each word which, when used after `"?"`, should lead to this help page. Usually includes name.
\title{Title}	Short description of the topic. No special characters. Starts with capital letter, no full stop at the end.
\description{...}	A few lines describing the topic
\usage{fun(arg1, arg2, ...)}	The exact function call syntax, showing the arguments. Verbatim-type command.
\arguments{...}	Description of each argument.
\details{...}	Detailed and if possible precise description of what the function does.

Help file format: 1. Functions, continued

\value{...}	Description of the return value. If this is a list, document each item individually.
\references{...}	References to the literature. Use `\url{}` for web addresses.
\author{...}	Author of the help file. Use `\email{}` for email addresses, `\url{}` for web addresses.
\note{...}	Anything you wish to point out especially.
\seealso{...}	Pointers to related R objects. Specify as `\code{\link{...}}`.
\examples{...}	Examples of how to use the function. They should run successfully and not take too long.
\keyword{key}	May be multiple such entries. Choose from a list in the /doc directory of R. In R, use `file.show(file.path(R.home("doc"), "KEYWORDS"))` to display the list of keywords.

Help file format: 2. Datasets

\name{name}	Name of data object, and help file
\docType{data}	Always `data`
\alias{topic}	As for functions.
\title{Title}	Short description of the data object.
\description{...}	A few lines describing the object,
\usage{name}	Or `data(name)` if lazy data is not in effect. Note the skeleton has `data(name)`.
\format{...}	Description of object. If this is a list or data frame, document each item individually.
\source{...}	Details of the original source. Use `\url{}` for web addresses.
\references{...}	References to secondary sources.
\examples{...}	Examples of how to use the data object. Load or display or plot, possibly.
\keyword{datasets}	Always `datasets`

Help file format: 3. Packages

A good idea to document the package: the skeleton will contain a file named `mypkg-package.Rd`, with an `alias` entry for the package name. Other entries are similar to the `DESCRIPTION` file: try to keep both up-to-date!

\name{name}	Name of package help file
\alias{topic}	As for functions.
\docType{package}	Always `package`
\title{Title}	
\description{...}	
\author{...}	Author(s) and maintainer of the package.
\references{...}	References to secondary sources.
\keyword{package}	Always `package`
\seealso{...}	Pointers to other packages: Specify as `\code{\link[<pkg>:<pkg-package]{<pkg>}}`
\examples{...}	Simple examples of the most important functions.

Exporting objects from Name spaces

- R will automatically create a name space for the package.
- When a package is loaded using `library()`, only exported items are placed in the attached frame, although all are loaded.
- You should export any objects which you would like the user to see, and only those.
- The skeleton will export everything: not recommended, as you will be forced to write a help file for every object!
- To export items `a` and `b`, use
  ```
  export(a, b)
  ```
- To specify the items using patterns, use e.g.
  ```
  exportPattern("^[^\\.]")
  ```
  (everything except if it starts with a dot.)

Other entries in the NAMESPACE file

import

- If you wish to use an item from a name space in another package, you can `import` it.
- `import(pkg1, pkg2)` will import all items from packages `pkg1` and `pkg2`.
- `importFrom(pkg1, a, b)` will import just items `a` and `b` from package `pkg1`

Generic function

- If your package includes a function intended to be used as a generic function, this should be indicated in the NAMESPACE file using an `S3method` statement: e.g. `S3method(print, myclass)`

Using compiled code in your package

- Create a `src` directory alongside the `R` one, and put your source code files in it.
- Avoid having any .o files in there: if you change platforms they will not be appropriate!
- If you have a complicated directory structure below `src` you may need to tell R about it, using `Makevars`. Look up the details in the *Writing R Extensions* manual if necessary.
- Load the compiled object with either
  `useDynLib(mypkg)` or
  `useDynLib(mypkg, mycfn1, mycfn2=myfn2)`
  in the `NAMESPACE` file. The latter form creates objects with corresponding names (so they need to be renamed here if they duplicate R objects in the package). The functions can then be referred to e.g. in a `.C` call as `mycfn1` with no quotes.

Startup and close down functions

- Special functions used to perform processing when a package is loaded or unloaded are defined in a source file called `zzz.R`.
- It is possible to `load` the package without `attaching` it. Thus there are four functions with arguments:

.onLoad	`libname`	character string containing the path of the library where the package was found
	`pkgname`	
.onAttach	as for `.onLoad`, use for things only necessary if a user has loaded the library, e.g. a banner	
.onUnload	`libpath`	character string containing the complete path to the package
.Last.lib	as for `.onUnload`	

`.onLoad`, `.onAttach`, `.onUnload` should not be exported, `.Last.lib` does need to be exported.

Building the package

Phew! At last we have everything we need!

- If our package is in a directory called `mypkgdir`, and the `DESCRIPTION` file indicates the package is called `mypkg` and is at version 1.0, then
  `R CMD build mypkgdir`
  (at the command prompt) will create a file called `mypkg_1.0.tar.gz` which is known as a tarball
- `R CMD INSTALL mypkg_1.0.tar.gz`
  will install the library
- on Windows,
  `R CMD INSTALL --build mypkg_1.0.tar.gz`
  will install the library *and* create a zip file called `mypkg_1.0.zip` which can be used to install the package via the function `install.packages` or the RGui menu.
- It seems too easy! But there is one more hurdle:

Does our package pass the checks?

- Contributed packages are required to pass some checks before they can be distributed via CRAN.
- The test is to run

  ```
  R CMD check mypkgdir
  ```

  or

  ```
  R CMD check mypkg_1.0.tar.gz
  ```

- The following are the main checks made:

install	It must be possible to install the package
portability	All file names must be valid for all supported systems.
permissions	Unix only: checks that the files have the right permissions
binary files	Will look for and warn about binary files (unsafe!)
DESCRIPTION file	Check for completeness and partially for correctness
subdirectories	Check for suitable names and not empty

CHECK continued

R files	Checked for syntax
load	Checked to see if the package can be loaded
Code problems	Checks to see if library.dynam, .C etc. calls can be interpreted sensibly. Also checks R code for problems.
Rd	Format check
undocumented items	Is there a `.Rd` file corresponding to each exported object
documentation	checked for consistency with use of function and datasets
usage	in `.Rd` files checked against `arguments` in the same files
source	C, C++, Fortran source is checked for portable line endings (LF-only). Also `Makefile` or `Makevars` in the `src` directory, if any
examples	All are run

Exercises

Just write a package!

No, seriously, look at the project for the course.

Well done: you have completed the course!