Building Packages

Chao-Jen Wong, Nishant Gopalakrishnan, Marc Carson

Fred Hutchinson Cancer Research Center

17-18 May, 2011

Package Concept

R packages

A collection of source code allows the user to attach to R session when calling `library()` or `require()`.

Why write a package?

- Better way to organize your code.
- Ability to share software as R packages.
- Provide reliable access.
- Provide communication channel between the users and authors.

Package Source

- Sperical files
  - Essential: `DESCRIPTION` and `NAMESPACE`.
  - Others: `configure`, `LICENSE`, `COPYING` and `NEWS`.
- Subdirectories

Directory	**Content**
`R`	source files (.R)
`data`	files of data objects to be loaded by `data()`
`inst`	content copied to the installed packages' directory `doc` – Sweave document (.Rnw) `extdata` – misc. data objects (ASCII) `unitTest` – unit testing functions
`man`	`Rd` documentation
`src`	source code in `C`, `FORTRAN` or `C++`
`tests`	test code in *R*

DESCRIPTION file

```
Package: StudentGWAS
Type: Package
Title: Basic tools for manipulating GWAS data
Description: This package contains basic tools for facilitating the
        manipulation and processing of GWAS data (i.e. data from a Genome
        association study). It is a pedagogical package and therefore its
        content is voluntarily limited to the material teached during the
        'Advanced R Programming' course offered in Seattle in February 20
Version: 0.0.1
Author: You
Maintainer: Chao-Jen Wong <cwon2@fhcrc.org>
Imports: DBI, RSQLite
Suggests: org.Hs.eg.db
License: Artistic-2.0
LazyLoad: yes
```

Creating a Package

- Using `package.skeleton()`

```
> ## objects to be included in the package
> area_rec <- function(width, length) width*length
> area_circle <- function(r) pi*r^2
> obj <- c("area_rec", "area_circle")
> package.skeleton("studentSWAS", list=obj,
+                  namespace=TRUE)
```

- Manually create a directory and subdirectories containing source codes, documentaiton and data objects.

Package Tools

R shell tools

- Used to manage packages (build, check and etc.).
- Can be accessed from a command shell.

Shell commends

Take the form: `R CMD` *operation*

Useful tools:

```
$R CMD build package
$R CMD build --binary package
$R CMD check package
$R CMD check --help
$R CMD INSTALL package
```

Documentation Types

Manual pages

- Reference pages for *R* objects (functions, data sets, etc.)
- Written in "R documentation" (Rd) format
- Thoroughly checked during R CMD check
- Templates created by `prompt*` family of functions

Vignettes

- A task-oriented description of package functionality
- Contain simple "HowTo"s that are built with the package
- Written in Sweave (.Rnw) format, which integrates R code into LaTeX documents
- Required component of a Bioconductor package

Details provided in *Writing R Extensions* manual.

Manual Pages (A Simple Example)

```
\name{name}
\alias{alternate name}
\title{name of manual page}
\description{Brief description of what this does.}
\usage{
  myfun(arg1, arg2 = FALSE)
}
\arguments{
  \item{arg1}{\code{arg1} is required}
  \item{arg2}{\code{arg2} is optional}
}
\details{Important details on how it does it.}
\value{Return type}
\seealso{\code{\link[pkg:pkgfun]{pkgfun}}}
\author{Your name here}
\examples{## R code to demo this}
\keyword{names from KEYWORDS file in R doc dir}
```

Manual Pages (Tips)

- Flip through "Writing R documentation files" chapter of *Writing R Extensions* manual.
- Change manual page when when underlying *R* object changes.
- Make examples run fast and be robust to changes in annotations and web resources.
- Run R CMD check (or R CMD Rd2dvi –pdf) on modified packages.

Check Rd files

R CMD Rdconv –type=html –output=foo.html StudentGWAS-package.Rd R CMD Rd2dvi –pdf *.Rd

S4 Documentation

S4!!!

More about Description File

collate field

COLLATE!!! show DESCRIPTION file here

makevars

makevars!!!

Dependencies in DESCRIPTION

- Declare what packages are required to run your package.
- Clarify the relationship between your package and other packages.
- Give clear and reliable definition of the package's behavior (namespaces).

Dependencies in DESCRIPTION

Depends

Packages expected to be attached

Imports

- Only a few functions or objects are used by this package.
- Not necessarily needed to be attached.
- Avoid the cost in time and space of accessing the unused functions.

Suggests

- Used in examples or vignettes.
- Introduce special functionality.

Namespace

Why give your package a namespace

- Avoid conflicts and confusion from multiple functions (from attached packages) with the same names.
- Control what are public (exported) and private.

Namespace

- Declare in the `NAMESPACE` file
- Required being explicit about what is exported and imported
- 'import' – entire package or specific objects, classes and methods

```
import(Biobase)
```

or

```
importFrom(Biobase, openVignettes)
```

- 'export' – explicit list of objects, methods and classes

```
exportPattern("^[^\\.]")
export(...)
exportClass(...)
exportMethods(...)
```

- Sealed once the package is installed Non-exported functions can be addressed by the `:::` operator

Useful Tool: codetoolsBioC

```
> library(codetoolsBioC)
> ls(2)
```

```
[1] "findExternalDeps"
[2] "getRdFileNames"
[3] "writeNamespaceImports"
[4] "writeRUnitRunner"
```

writeNamespaceImports

Writes imports statements that can be included in a package's `NAMESPACE` file

```
> library(GenomicFeatures)
> writeNamespaceImports("GenomicFeatures")
```

What is a unit test?

A function myFun

```
library(RUnit)

myFun <- function(a) {
  # input checking
  if(!is.numeric(a))
    stop("'a' should be of
         type 'numeric(1)'")
  if(length(a) != 1)
    stop("'a' should be of
           length 1")

  # calc factorial
  factorial(a)
}
```

Unit test for myFun

```
test_myFun <- function() {
  target <- 6
  current <- myFun(3)
  checkIdentical(target,current)

  checkException(myFun("A"))

  checkException(myFun(1:8))
}
```

Why Unit tests ?

- Interface specification
- Ensures code correctness, e.g., when R changes
- Allows refactoring without breaking existing code
- Encourages writing simple, working code chunks that can be integrated into larger components
- Encourages collaboration – tests describe what is supposed to happen
- Helps describe bugs – 'this test fails'
- Documentation for developer – what code is intended to do

The *RUnit* package

- Framework for test case execution
  - create a series of test functions
  - define a test suite (defineTestSuite)
  - run the tests (runTestSuite)
  - summarize results (printTextProtocol, printHTMLProtocol)
- Hint: use `writeRUnitRunner` from the *codetoolsBioC* package

Adding Unit tests to your package

- Create test functions
  - save in inst/unitTests folder of your package
- Function to create test suite, run tests, summarize results
  - use `writeRUnitRunner` to create the file containing the `.test` function
  - save in R folder of your package
- Function to call the `.test` function
  - save in the tests folder of your package
- Add *RUnit* to the Suggests field in DESCRIPTION

Running a unit tests

```
> library(nidemo)
> nidemo:::.test()
```

Vignette (Skeleton)

```
%\VignetteIndexEntry{Descriptive Title}
%\VignetteKeywords{words}
%\VignettePackage{Package Name}

\documentclass[11pt]{article}

\usepackagege{Sweave}

\newcommand{\Rfunction}[1]{{\texttt{#1}}}
\newcommand{\Robject}[1]{{\texttt{#1}}}
\newcommand{\Rpackage}[1]{{\textit{#1}}}
\newcommand{\Rclass}[1]{{\textit{#1}}}

\title{Descriptive Title}
\author{your name}


\begin{document}
\maketitle

\end{document}
```

Vignette (Body)

```
\begin{document}
\maketitle

\section{Introduction}
The \Rpackage{foo} is great.

\subsection{Getting Started}
First load the \Rpackage{foo} package and then execute
function \Rfunction{foo}.

 <<code-block>>=
 library(foo)
 foo()
 @

\end{document}
```

Vignette (Code Blocks)

```
<<UnevaluatedCode, eval=FALSE>>=
longRunningFunction(bigDataObject)
@
<<UnseenCodeAndOutput, echo=FALSE>>=
options(width = 60)
@
<<UnseenMessages, results=hide>>=
library(Biobase)
@
<<IncludeGraphic, fig=TRUE>>=
plot(1:10)
@
<<KeepMyFormat, keep.source=TRUE>>=
loveMyFormat(arg1 = "first",
             arg2 = "second")
@
```

Sweave and Stangle Commands

`Sweave` – creates a post-(code block)-processed LaTeX file

`Stangle` – creates an R script from code blocks

R commands

```
> library(tools)
> Sweave("foo.Rnw")
> texi2dvi("foo.tex", pdf=TRUE, clean=TRUE)
> Stangle("foo.Rnw")
```

Shell commands

```
R CMD Sweave foo.Rnw
R CMD texi2dvi --pdf --clean foo.tex
R CMD Stangle foo.Rnw
```

Need for Version Control

Problems

- Projects consist of multiple files
- We add/remove/change content
- Multiple people editing same file -> merge changes
- Multiple machines/operating systems -> merge changes
- Go back to a previous snapshot

The wrong way

- proj1.R, proj2.R, proj3.R
- User managed backups

Version control software

- svn
- Mercurial
- git

Bioconductor svn

- Devel Branch
  - `https://hedgehog.fhcrc.org/bioconductor/trunk/madman/Rpacks`
- 2.6 Release Branch
  - `https://hedgehog.fhcrc.org/bioconductor/branches/RELEASE_2_6/madman/Rpacks`
- username:readonly password:readonly

Reference Book: *Version Control with Subversion*
`http://svnbook.red-bean.com/`

Useful svn commands: svn checkout

svn co
https://hedgehog.fhcrc.org/bioconductor/trunk/madman/Rpacks/
BiocCaseStudies/ –username readonly –password readonly

Useful svn commands: svn log

svn log NAMESPACE | more

- Logs are useful only if useful commit messages are provided.
- Commit once conceptual change at a time.

Useful svn commands

- svn checkout
- svn add
- svn checkin
- svn update
- svn status
- svn log -v
- ...

Efficient Editing

Editing from the command line

- history()
- savehistory("foo")
- loadhistory("foo")
- reverse search on unix using `Ctrl-R`

Code editors

- Eclipse(StatET) `http://www.walware.de/goto/statet`
- Emacs(ESS) `http://ess.r-project.org/`
- vim(Vim-R-plugin2) `http://www.vim.org/scripts/script.php?script_id=2628`
- Tinn-R
- Notepad++(NppToR) `http://sourceforge.net/projects/npptor/`

Advantages

- syntax highlighting
- auto indent code
- send code/functions to R console

Efficient work flows

Editing without building documentation or configure

```
R CMD check --no-vigenttes --no-examples pkgs
R CMD INTSALL --no-docs pkgs
R CMD INSTALL --no-configure pkgs
R CMD INSTALL --help
```

.Rprofile

```
.First <- function() {cat("\n Hello! \n\n")}

if (interactive()) {
    tryCatch({
        source("http://bioconductor.org/biocLite.R")
    }, error=function(e) invisible(NULL),
       warning=function(w) message("Not connected to the net"))
}

reload_pkg <- function(p) {
    detach(paste("package", p, sep = ":"), unload = TRUE,
           character.only = TRUE)
    library(p, character.only = TRUE)
}
```

Resources

- John Chambers. *Software for Data Analysis*. Springer, New York, 2008.
- *Writing R Extensions* manual, `http://cran.r-project.org/doc/manuals/R-exts.html`
- *Version Control with Subversion*, `http://svnbook.red-bean.com/`
- *Sweave User Manual*, `http://www.stat.uni-muenchen.de/~leisch/Sweave`