

# Building Packages

Chao-Jen Wong, Nishant Gopalakrishnan, Marc Carlson

Fred Hutchinson Cancer Research Center

17-18 May, 2011

## R Packages

- Package Concept

- Creating a Package

- Package Tools

## Package Documentation - part 1

- Help Pages

- Sweave

- Package Vignettes

## Package Dependency and Namespaces

## Unit Testing

## Tools of the trade

- Version Control

- Efficient Work flows

## Resources

# 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.
- ▶ *R* tools support quality control checks.

# Package Source

- ▶ Special files
  - ▶ Essential: DESCRIPTION and NAMESPACE.
  - ▶ Others: configure, LICENSE, COPYING, NEWS and etc.
- ▶ 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	<i>R</i> documentation (.Rd)
src	source code in C, FORTRAN or C++
tests	test code in <i>R</i>

# The 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-wide association study). It is a pedagogical package and therefore its content is voluntarily limited to the material taught during the 'Advanced R Programming' course offered in Seattle in February 2011.

Version: 0.0.1

Author: You

Maintainer: You <youremail@email.com>

Imports: DBI, RSQLite

Suggests: org.Hs.eg.db

License: Artistic-2.0

LazyLoad: yes

Collate: utils.R SQLiteFunctions.R GWASdata-class.R fapply.R

# The NAMESPACE file

```
import(methods)
import(DBI)
import(RSQLite)

exportClasses(
  GWASdata
)

export(
  ## Ordinary functions (i.e. not generic):
  getSnp,
  getSubjects,
  getKEGGSnp,
  GWASdata
)

exportMethods(
  ## Methods associated with the generic functions
  datapath, dataconn,
  metadatapath, metadataconn,
  getCols,
  cld
)
```

# 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("StudentGWAS", list=obj,
+                  namespace=TRUE)
```
- ▶ Manually create a top-level directory and subdirectories and put files, source codes and documentation in appropriate places.

# Package Tools

## R shell tools

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

## Shell commands

Take the form: R CMD *operation*

Useful tools:

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



# Lab

Execises 1 to 5 in *Building Packages: Self-study Exercises*.

# Package documentation

## Help pages

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

Details provided in the *Writing R Extensions* manual.

# Package documentation

## Package 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  $\text{\LaTeX}$  documents

Details provided in the *Sweave User Guide* manual.

## Help 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}
```

# Help Pages (Tips)

- ▶ Flip through “Writing R documentation files” chapter of the *Writing R Extensions* manual.
- ▶ Change help 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.

# Sweave documentation

## What is Sweave

- ▶ Enables integration *R* code for data analysis into  $\text{\LaTeX}$  documents
- ▶ Produces dynamic, reproducible and transparent reports
- ▶ Offers full power of  $\text{\LaTeX}$  for high-quality typesetting

# Sweave

## How does it work

- ▶ Transfer both the *R* code and respective output into  $\text{\LaTeX}$ 
  - ▶ `.Rnw -> .tex`
  - ▶ Sweave
- ▶ Final document is created by running `latex` on the `.tex` file
  - ▶ `.tex -> .pdf`
  - ▶ `texi2dvi`: run `pdflatex` on the `.tex` file to create the PDF version

# Sweave files format (.Rnw)

```
\documentclass[a4paper]{article}
\usepackage{Sweave}
\begin{document}
```

In this example, we embed parts of the examples from the `\texttt{boxplot}` help page into a `\LaTeX{}` documentation.

```
<<get data>>=
mat <- cbind(Uni05 = (1:100)/21, Norm = rnorm(100),
              `5T` = rt(100, df = 5),
              Gam2 = rgamma(100, shape = 2))
head(mat)
@

<<plot_boxplot, fig=TRUE>>=
boxplot(as.data.frame(mat),
        main = "boxplot(as.data.frame(mat), main = ...)")
@
\end{document}
```



# The tex file

```
\documentclass[a4paper]{article}
\usepackage{Sweave}
```

```
\begin{document}
```

In this example, we embed parts of the examples from the  
`\texttt{boxplot}` help page into a `\LaTeX{}` documentation.

```
\begin{Schunk}
\begin{Sinput}
> mat <- cbind(Uni05 = (1:100)/21, Norm = rnorm(100), `5T` = rt(100,
+   df = 5), Gam2 = rgamma(100, shape = 2))
> head(mat)
\end{Sinput}
\begin{Soutput}
```

	Uni05	Norm	5T	Gam2
[1,]	0.04761905	0.2248667	0.39843424	2.1904914
[2,]	0.09523810	0.7875030	0.39367707	1.8562296
[3,]	0.14285714	0.1488066	-0.23620478	1.8673665
[4,]	0.19047619	-1.9921185	-2.20927228	6.1497068
[5,]	0.23809524	-0.4765474	-3.43556247	0.7488957
[6,]	0.28571429	-0.5603378	-0.08039217	0.2130001

```
\end{Soutput}
\end{Schunk}
```

```
\begin{Schunk}
\begin{Sinput}
> boxplot(as.data.frame(mat), main = "boxplot(as.data.frame(mat), main = ...)")
\end{Sinput}
\end{Schunk}
\includegraphics{example-plot boxplot}
\end{document}
```

# Vignette Skeleton

```
%\VignetteIndexEntry{An R package for ...}  
%\VignetteKeywords{kwd1}  
%\VignettePackage{Package Name}  
%\VignetteDepend{pkg1, pkg2, ...}  
  
\documentclass[11pt]{article}  
  
\usepackage{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}
```

# Sweave options (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  $\text{\LaTeX}$  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
```

# 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 to the *R* session.

## 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.

# 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))  
}
```

# Namespaces

- ▶ 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

Install the *codetoolsBioC*:

```
> source("http://bioconductor.org/course-package/courseInstall.R")
> courseInstall("codetoolsBioC")

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

## writeNamespaceImports

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

```
> library(Biobase)
> writeNamespaceImports("Biobase")
```

#Generated by codetoolsBioC version 0.0.16

#Timestamp: Thu Feb 17 14:12:08 2011

#Imports: methods, utils

```
importClassesFrom(methods, ANY, character, data.frame, environment,
                  "function", integer, list, logical, matrix, missing,
                  "NULL", numeric)
```

```
importMethodsFrom(methods, coerce, Compare, initialize, show)
```

```
importFrom(methods, "@<-", as, callGeneric, callNextMethod, extends,
            getClass, getSlots, is, isClass, isGeneric, isVirtualClass,
            new, setGeneric, setMethod, slot, "slot<-", slotNames,
            validObject)
```

```
importFrom(utils, menu, packageDescription, read.table, write.table)
```

# Lab

Execises 6 to 7 in *Building Packages: Self-study Exercises*.

# 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(StuendSWAS)  
> StudentGWAS:::.test()
```

# 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>

- ▶ 2.7 Release Branch

- ▶ [https://hedgehog.fhcrc.org/bioconductor/branches/RELEASE\\_2\\_7/madman/Rpacks](https://hedgehog.fhcrc.org/bioconductor/branches/RELEASE_2_7/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 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
```

# 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 Guide* manual,  
<http://www.stat.uni-muenchen.de/~leisch/Sweave>