Building an R Package

Seth Falcon

27 January, 2010

Contents

1 Introduction

Packages in R provide a reliable mechanism for bundling code, data, and documentation into a reusable and sharable form. While R packages are most commonly used to share general purpose functionality, they can also be used to organize data analysis projects. In this case, the raw data and the R scripts used to process and analyze the data are kept together in a way that makes it easy to reproduce an analysis as well as to update it with either new data or new analysis methods.

In this lab, you will create your own package that will contain raw data describing patients with ALL (acute lymphoblastic leukemia) along with functions for reading the raw data into a *data.frame* and subsetting the data. We are using a portion of the data found in the ALL package available from the Bioconductor project.

An R package examplePkg consists of a directory `examplePkg` on disk containing files and subdirectories with standard names. This directory is the *package*

source. You use `R CMD build examplePkg` to create a package archive from the source directory (resulting in the archive `examplePkg_0.0.1.tar.gz`). The package archive can be shared with others and installed using `install.packages` or `R CMD INSTALL examplePkg_0.0.1.tar.gz`.

2 Package Structure By Example

We begin by examining the source for the `day1` package.

Exercise 1
Obtain the source package archive for the day1 package and unpack it. You can unpack the archive from within R using the `untar`.

You should now have a directory `day1` containing subdirectories `inst` and `man` as well as a file called `DESCRIPTION`.

It is important to distinguish between package sources and installed packages.

Exercise 2
If you haven't already installed the day1 package, do so now. If you have the day1 archive, you can install like this:

```
> p <- "~/path/to/day1_0.0.1.tar.gz"
> install.packages(p, type = "source", repos = NULL)
```

Now use `system.file` and `dir` to list the files associated with the installed version of day1.

```
> dir(system.file(package = "day1"))
```

```
[1] "DESCRIPTION"   "INDEX"         "Meta"
[4] "_pkg_solution" "doc"           "extdata"
[7] "help"          "html"
```

Observe that there are additional files in the installed version. When creating your own package, you should never modify the installed version. Always edit the package source and install it using R.

The `inst` subdirectory in the source package contains files that will be copied to the top-level of the installed package.

Exercise 3
Verify that the files in `day1/inst` have been copied to the top-level of the installed location for the day1 package (use `system.file` to find this location and `dir` to list files in a directory on disk).

While the `inst` subdirectory is optional, the `DESCRIPTION` file is a required component of an R package. It describes the package.

Exercise 4
Open the `DESCRIPTION` file found in the package source directory for `day1`. What is the license for the package?

The definitive reference for writing R packages is the *Writing R Extensions* manual. It comes with R and can be accessed using the `RShowDoc` function:

```
> ## This opens the Writing R Extensions manual
> ## in your web browser
> RShow("R-exts")
```

3 ALLpheno Package Skeleton

The objective of this section is to create the skeleton of a simple package. We will refine the skeleton in subsequent sections to create a more complete package, containing a data set and functions to process it.

Exercise 5
Create a directory named `ALLpheno`.

Exercise 6
Create a `DESCRIPTION` file inside the `ALLpheno` directory using a text editor and add the following lines, making appropriate edits for `Author` and `Maintainer`:

```
Package: ALLpheno
Title: Phenotype data and summary for the ALL dataset
Version: 0.0.1
Author: your name <you@mail.com>
Maintainer: your name <you@mail.com>
License: Artistic-2.0
Description: you can enter a multi-line description
  by indenting extra lines.  See the 'day1' package
  DESCRIPTION file for an example of how to do this.
```

You should now have a skeleton R package that you can install by following the instructions below[1] As you continue to develop this package, you will repeat these steps to install the source package so that you can test your package.

[1]The instructions that follow are designed to work without the installation of additional development tools. Normally a package archive (.tar.gz) is created using `R CMD build ALLpheno`. To be able to run this command on Windows, you must install Rtools (http://www.murdoch-sutherland.com/Rtools/). To run on OS X you need to install Xcode (http://developer.apple.com/technology/xcode.html). You can complete this lab without installing this extra software.

3.1 Installation from a running R session

Start R and issue the following commands, substituting the correct file path for where your copy of `ALLpheno` is located on your system.

```
> # note the forward slashes even on Windows!
> forWin <- "z:/course/ALLpheno"
> forMac <- "~/course/ALLpheno"
> install.packages(forWin, type = "source", repos = NULL)
```

4 Adding the Raw Phenotype Data

The day1 package contains a file called `ALL_pdata.txt` under the `extdata` directory.

Exercise 7

1. *Use* `system.file` *to locate the* `ALL_pdata.txt` *file. Copy the file (using standard operating system commands or* `file.copy` *) to your ALLpheno source package under* `inst/extdata`*.*

2. *Reinstall your package and verify that the* `ALL_pdata.txt` *file has been installed by using* `system.file`

```
> dir(system.file("extdata", package="day1"))
> srcf <- system.file("extdata", "ALL_pdata.txt", package="day1")
> ## change this as appropriate for your setup
> mypkgdir <- "~/course/ALLpheno"
> dest_dir <- file.path(mypkgdir, "inst", "extdata")
> dir.create(dest_dir, recursive = TRUE)
> file.copy(srcf, dest_dir, overwrite = TRUE)
> install.packages(mypkgdir, type = "source", repos = NULL)
> dir(system.file("extdata", package="ALLpheno"))
```

5 Reading and Filtering Phenotype Description Data

Exercise 8

Use `read.table` *to read in the* `ALL_pdata.txt` *file from your installed copy of ALLpheno (use* `system.file` *to obtain the path to the text file. Hint: if you encounter errors, try opening* `ALL_pdata.txt` *to examine its format.*

```
> data_file <- system.file("extdata", "ALL_pdata.txt",
+                          package = "ALLpheno")
> df <- read.table(data_file, comment.char = "%")
> head(df[ , c("cod", "BT", "sex", "mol.biol")])
```

```
       cod BT sex mol.biol
01005 1005 B2   M  BCR/ABL
01010 1010 B2   M      NEG
03002 3002 B4   F  BCR/ABL
04006 4006 B1   M ALL1/AF4
04007 4007 B2   M      NEG
04008 4008 B1   M      NEG
```

Exercise 9
Subset the ALL phenotype data to select patients with B-cell ALL having either the BCR/ABL or no mutation (coded as NEG).

```
> ## You can adapt the approach taken in the
> ## Bioconductor Case Studies ALL chapter.
> ## Below is a slightly modified approach using
> ## logical vectors instead of integer index
> ## vectors.
> bcell <- grepl("^B", as.character(df$BT))
> types <- c("NEG", "BCR/ABL")
> moltyp <- as.character(df$mol.biol) %in% types
> df_sub = df[bcell & moltyp, ]
> dim(df_sub)
```

```
[1] 79 21
```

Exercise 10
Write a function that encapsulates the previous two exercises. Your function should take no arguments. It should load the phenotype data from the installed ALLpheno *package location and perform the necessary filtering. It should return a data.frame containing the desired subset of the data. Give your function the name* `load_pheno`.

```
> load_pheno <- function()
+ {
+     data_file <- system.file("extdata", "ALL_pdata.txt",
+                              package = "ALLpheno")
+     df <- read.table(data_file, comment.char = "%")
```

```
+     bcell <- grepl("^B", as.character(df$BT))
+     types <- c("NEG", "BCR/ABL")
+     moltyp <- as.character(df$mol.biol) %in% types
+     df[bcell & moltyp, ]
+ }
```

Exercise 11
Add your function to the ALLpheno *package. To do this, create a directory* `R` *inside your* `ALLpheno` *source package directory. (It is important to call the directory* `R`*, rather than* `r`*, because R works on operating systems that treat file and directory names in a case sensitive way, even if your operating system does not). Then add your function code to* `R/code.R`*.*

Exercise 12
Reinstall your ALLpheno *package. Start a new R session and verify that you can load the* ALLpheno *package and run the* `load_pheno` *function to obtain the filtered phenotype data.*

1. *Does it matter how you name the the* `.R` *file?*
2. *Can you have more than one* `.R` *file in the* `R` *directory?*
3. *How can you control the order in which* `.R` *files are loaded if there is more than one?*

```
> library("ALLpheno")
> df2 <- load_pheno()
> head(df2[ , c("cod", "BT", "sex", "mol.biol")])

       cod BT sex mol.biol
01005 1005 B2   M  BCR/ABL
01010 1010 B2   M      NEG
03002 3002 B4   F  BCR/ABL
04007 4007 B2   M      NEG
04008 4008 B1   M      NEG
04010 4010 B1   F      NEG
```

You can name your `.R` files however you want as long as you stick to names with no spaces or special character. There are many ways to organize the R code in a package. You can have a a single file for all code, separate file for each function, or a few files that group related functionality.

By default, the `.R` files are loaded by R in the order determined by sorting the file names. If you need to control the order in which source files are loaded, you should add a `Collate` field to your `DESCRIPTION` file. Read the *Writing R Extensions* manual for details.

6 A Reusable Subsetting Function

The objective of this section is to create a function for extracting the desired subset of the ALL phenotype data that is flexible and reusable. In Exercise 10 you created the `load_pheno` function that reads the raw phenotype data and returns the desired B-cell BCR/ABL vs NEG subset. In the following exercises you will extract the subsetting code into a separate function (`subset_ALL`) and make it more general by allowing a user of the function to specify a cell type (B or T) as well as the desired mutations (e.g. BCR/ABL, NEG).

Exercise 13
Create a function `subset_ALL` that performs subsetting of the ALL phenotype data based on the cell type and mutation specified by the user. Here is an outline for your new function:

```
> subset_ALL <- function(x, cell.type, mol.types)
+ {
+     ## your code here
+ }
```

Here is one way to code the reusable subsetting function:

```
> subset_ALL <- function(x, cell.type = c("B", "T"),
+                        mol.types = c("BCR/ABL", "NEG"))
+ {
+     cell.type <- match.arg(cell.type)
+     bcell <- grepl(paste("^", cell.type, sep=""),
+                    as.character(x[["BT"]]))
+     moltype <- as.character(x[["mol.biol"]]) %in% types
+     if (!any(moltype))
+         warning("No patients matched mol.types of:",
+                 paste(mol.types, collapse=", "))
+     s <- x[bcell & moltype, ]
+     ## recode factors to drop levels
+     ## that are no longer present in the data
+     s[["BT"]] <- factor(s[["BT"]])
+     s[["mol.biol"]] <- factor(s[["mol.biol"]])
+     s
+ }
```

Exercise 14

1. *Add `subset_ALL` to your ALLpheno package.*
2. *Update your `load_pheno` function to call `subset_ALL` to avoid having a near repeat of the subsetting code in your package.*

The BT and mol.biol columns of the data are *factor* variables, an R class used to represent categorical data. If you summarize either of these columns in your subset, for example using `summary(df[["BT"]])`, you will see that there are extra categories or *levels* that no longer make sense in the context of the subset.

Exercise 15

1. *Recreate the factor variables for `BT` and `mol.biol` following this pattern:*

   ```
   > newfactor <- factor(oldfactor)
   ```

   Observe that this removes unused levels.

2. *Add the recoding of the `BT` and `mol.biol` factors to your `subset_ALL` function so that the returned subset does not contain factor variables with unused levels.*

7 Adding Some Documentation

As a final step with the ALLpheno package, we will add some documentation that will integrate with R's help system. Package documentation is written in `.Rd` files placed in the `man` subdirectory of a package.

Exercise 16
Create a `man` directory inside your `ALLpheno` source directory.

Exercise 17
In a new R session, load the `ALLpheno` package and use the `promptPackage` function to create a template package documentation file. By default `promptPackage` will write an `.Rd` file to your current working directory. Copy this file to `ALLpheno/man/`.

```
> promptPackage("ALLpheno")
```

Exercise 18
Edit `ALLpheno/man/ALLpheno-package.Rd` to replace the filler text with actual documentation. Reinstall your `ALLpheno` package, load it, and then view the package help page with `package ? ALLpheno`.

Exercise 19
Use `prompt` to create template documentation files for the `load_pheno` and `subset_ALL` functions. Copy the files into `ALLpheno/man` and edit them. Then reinstall the package and view the help for your functions.

```
> library("ALLpheno")
> prompt(load_pheno)
> prompt(subset_ALL)
```

8 Session information

- R version 2.10.1 Patched (2009-12-14 r50736), `i386-apple-darwin10.2.0`
- Locale: `C/C/C/C/C/en_US.utf-8`
- Base packages: base, datasets, grDevices, graphics, methods, stats, tools, utils
- Other packages: ALL 1.4.7, ALLpheno 0.0.1, AnnotationDbi 1.8.1, Biobase 2.6.1, DBI 0.2-5, RSQLite 0.8-0, genefilter 1.28.2, hgu95av2.db 2.3.5, lattice 0.17-26, org.Hs.eg.db 2.3.6
- Loaded via a namespace (and not attached): annotate 1.24.1, grid 2.10.1, splines 2.10.1, survival 2.35-7, xtable 1.5-6