HowTo set up a simple R repository

Jeff Gentry

February 10, 2006

1 Overview

This article demonstrates how you can make use of the *reposTools* package available in the Bioconductor project to quickly and easily create a R repository for distribution of your files to other R users. To do this, you will need the *reposTools* package. This package must be installed for your version of R and when you start R you must load with the `library` command.

```
> library(reposTools)
```

For the purposes of this vignette, we will be using a sample directory that comes with *reposTools* to create a small repository.

2 Initial setup of repository structure

The first step to create a repository is to have a directory on the local system that contains the files you want to distribute. This directory should reside in a world readable file system (e.g. web space), or it will have to be moved to such a location later on. The specific method of accomplishing this depends on your OS, etc.

3 Creation of a vanilla repository

For most situations, a repository can be created with a single command. With the directory you want to use as a repository as your R working directory:

```
> genRepos("Test Repository", "http://biowww.dfci.harvard.edu/~jgentry/",
+     "newRepos")

Loading required package: XML
$repName
[1] "Test Repository"

$repType
```

```
[1] "package"

$repaddrBase
[1] "http://biowww.dfci.harvard.edu/~jgentry/"

$repaddrPath
[1] "newRepos"

$repRelLevel
[1] "release"

$repDir
[1] "."
```

The actual signature of the function (with explanation of the parameters) is: genRepos(reposName, urlBase, urlPath, dir=".")

```
reposName: Identifying name of the repository ("Test Repository")
urlBase:   A base URL for your repository (e.g. "http://cran.r-project.org").
           This can also be a symbolic name (e.g. "CRAN"), which if that
           symbol exists in the client's "repositories" option, the URL
           that maps to the symbol will be used instead.
urlPath:   The file path from the base URL.  ("newRepos") Pasted on to
           the urlBase. (http://biowww.dfci.harvard.edu/~jgentry/newRepos)
           Mainly useful if a symbolic name is used in urlBase. (If
           using the symbol CRAN for instance, to promote mirroring,
           urlPath would have the path from the user's CRAN option).
dir:       The directory to use to construct a repository.  The default is
           the current directory.
```

At this point, there will be two new files constructed in your repository directory:

```
> dir()

[1] "PACKAGES"                "PACKAGES.gz"
[3] "goodbyeWorld_1.0.tar.gz" "helloWorld_1.0.tar.gz"
[5] "html"                    "repThemes.rda"
[7] "repdatadesc.rda"         "replisting"
```

The `repdatadesc.rda` file is a database file which contains all the detailed information about all your distributable files. The 'replisting' file contains information about your repository itself. At this point, others can now access your repository using the client side repository tools if you provide them your URL.

4 Repository themes

Repository themes are a concept which allows repository maintainers to specify groupings of packages together which can be downloaded as a single entity. This

allows for maintainers of repositories with many packages to specify smaller clusters of packages as being meaningfully connected, and in a sense provides subrepositories (For a description of actual subrepositories, see the next section).

A repository theme is defined as one or more packages that are contained within a repository which are grouped together under a particular theme name. They are defined by the repository maintainer with an XML file `reposThemes.xml`, which is then read in by the `genRepos` function and the resulting R structures are stored in the control file `reposThemes.rda` which is used by the client functions to determine what themes are available and what they contain. Repository maintainers can choose to author the XML file themselves, or use the function `writeThemesXML`.

The XML structure of the `reposThemes.xml` file is straightforward. The `repositoryThemes` tag defines the primary XML block that is specifying the themes for a particular repository. Since every `repThemes.xml` file corresponds to a particular repository, there should only be one of these blocks per file, and it should contain all other XML information.

Every individual theme is contained in a `theme` tag block. Every theme for this particular repository needs to be defined as a `theme` block. Within every theme are two main subblocks: The `themeName` tag block specifies the name of the theme and the `themePackages` tag block specifies what packages are contained within that theme. Inside of the `themePackages` tag block are a series of `package` tag blocks, one for every package in the theme. The `package` blocks contain one or two tags each. Every `package` block has a `packageName` tag block specifying the name of the package, and there is an optional `packageVersion` tag which allows the maintainer to specify a particular version of that package. If no version is specified, the highest available version of that package in the repository is used.

An exapmle of a simple repository theme is provided, with three packages and where two of the packages have specified versions and one does not.

```
<?xml version="1.0"?>
<repositoryThemes>
   <theme>
     <themeName>Testing</themeName>
     <themePackages>
       <package>
         <packageName>Biobase</packageName>
 <packageVersion>1.4.2</packageVersion>
       </package>
       <package>
         <packageName>ROC</packageName>
       </package>
       <package>
         <packageName>Ruuid</packageName>
         <packageVersion>1.3.3</packageVersion>
       </package>
```

```
        </themePackages>
      </theme>
    </repositoryThemes>
```

The `writeThemesXML` function takes in a list, where each element's name corresponds to the name of a theme. Each element in turn contains a list where every element is either a package name or a `pkgInfo` object. The latter case is used for situations where the maintainer wishes to specify a particular version for a package.

```
> themeList <- list(Theme1 = list("Biobase", "Ruuid", "ROC", "geneplotter"),
+     Theme2 = list(buildPkgInfo("annotate", "1.3.0"), "genefilter",
+         buildPkgInfo("affy", "1.3.0")))
> outFile <- tempfile()
> writeThemesXML(themeList, outFile)
```

Here we have used a non-default output file due to specifics of vignette creation, so in most normal cases one would only use the first parameter. We are also showing here the contents of the generated XML file.

5 Linking in subrepositories

If you have other repositories that are logically connected , or if you would simply like to link in other repositories to your own (recall that the client software will search through any of the server's known repositories if it fails to find something in the server repository itself), you can do this by editing the 'replisting' file.

This file is in DCF format (see below), with the following structure for each known repository (all these fields match up with the corresponding argument to genRepos() above:

```
repname: <repository name>
reptype: <repository type>
repaddrBase: <URL base>
repaddrPath: <path from URL base>
```

A blank line is used as a separater between two entries. You'll notice that there is already one entry with the information you entered in `genRepos`. Leave this as the first entry, if you want to link in other repositories, append their entry to the listing - you'll have to edit this file by hand though as there are not currently any tools for doing this automatically.

6 DCF Format

As mentioned above, these files use a format known as DCF. To provide you with some background on this format, the following is provided from `help("read.dcf")`:

```
DCF is a simple format for storing databases in plain text files
that can easily be directly read and written by humans. DCF is
used in various places to store R system information, like
descriptions and contents of packages.

The DCF rules as implemented in R are:

  1.  A database consists of one or more records, each with one
      or more named fields. Not every record must contain each
      field.

  2.  Regular lines start with a non-whitespace character.

  3.  Regular lines are of form `tag:value', i.e., have a name
      tag and a value for the field, seperated by `:' (only the
      first `:' counts). The value can be empty (=whitespace
      only).

  4.  Lines starting with whitespace are continuation lines (to
      the preceding field) if at least one character in the line
      is non-whitespace.

  5.  Records are seperated by one or more empty (=whitespace
      only) lines.
```

R provides a good amount of functionality for handling DCF files. To get started, see the functions `read.dcf`.

7 Notes On Repository Clients

This section will provide a brief overview of how the client repository tools interact with your repository server. A client will have a set of repository locations as well as a set of objects to obtain. Progressing down the list of repositories, they will download each server's `repdatadesc.rda` file and examine it to see if that repository contains what they are looking for. They optionally will then look at any subrepositories known by the server before moving on to the next repository on their list.

If the client finds their desired object, it will obtain the proper file URL from the `repdatadesc.rda` file, download it and use it to install/update/etc according to their wishes. The client uses automatic library management, and will keep track of what packages the user has installed, the versions of those packages, etc.

For more information on the repository client functionality, please see the vignette `HowTo Access A File Repository`.