

# HowTo set up a simple R repository

Jeff Gentry

January 23, 2004

## 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", "package", TRUE, TRUE)
```

```
$repName  
[1] "Test Repository"
```

```
$repType  
[1] "package"
```

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

```
$repaddrPath
[1] "newRepos"
```

```
$repDir
[1] "."
```

The actual signature of the function (with explanation of the parameters) is:  
genRepos(repoName, urlBase, urlPath, repType=c("package", "vignette", "data"),  
PACK=FALSE, Status=FALSE, dir=".", dataPath="", infoPath="")

repoName: 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).  
repType: The type of files being distributed. Can be either  
"package", "data", or "vignette".  
PACK: If FALSE, it is assumed that the user will supply a PACKAGES.in  
file (see section XXX). If TRUE, this information is created  
for the user.  
Status: If FALSE, it is assumed that the user will supply appropriate  
Status files (see section XXX). If TRUE, this information is  
created for the user.  
dir: The directory to use to construct a repository. The default is  
the current directory.  
dataPath: A relative path from 'dir' of a directory which contains  
the distributable material. Typically they are kept in the main  
repository directory, but this option allows people to provide a bit  
cleaner structure if they desire.  
infoPath: A relative path from 'dir' of a directory which contains  
the information files (see section XXX) if they are to be  
used.

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

```
> dir()
```

```
[1] "CVS"                "Ruuid_1.0.tar.gz"    "qvcalc_0.7.tar.gz"
[4] "repmatadesc.rda"   "replisting"         "xtable_1.1-2.tar.gz"
```

The `repmatadesc.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 Linking in subrepositories

If you have other repositories that are logically connected (e.g. a vignette repository or a data repository that go with your package repository, or if you have substructures), 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 separator 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.

## 5 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, separated 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 separated 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`.

## 6 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 `repmatadesc.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 `repmatadesc.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`.