Journal of Statistical Software

November 2017, Volume 82, Issue 1. *doi: 10.18637/jss.v082.i01*

Enhancing Reproducibility and Collaboration via Management of R Package Cohorts

Gabriel Becker
Genentech Research and
Early Development

Cory Barr
Anticlockwork Arts

Robert Gentleman
Computational Biology, 23andMe Inc.

Michael Lawrence
Genentech Research and
Early Development

Abstract

Science depends on collaboration, result reproduction, and the development of supporting software tools. Each of these requires careful management of software versions. We present a unified model for installing, managing, and publishing software contexts in R. It introduces the package manifest as a central data structure for representing version-specific, decentralized package cohorts. The manifest points to package sources on arbitrary hosts and in various forms, including tarballs and directories under version control. We provide a high-level interface for creating and switching between side-by-side package libraries derived from manifests. Finally, we extend package installation to support the retrieval of exact package versions as indicated by manifests, and to maintain provenance for installed packages. The provenance information enables the user to publish libraries or sessions as manifests, hence completing the loop between publication and deployment. We have implemented this model across three software packages, **switchr**, **switchrGist** and **GRANBase**, and have released the source code under the Artistic 2.0 license.

Keywords: reproducibility, collaboration, software distribution, package development.

1. Introduction

Every data analysis rests on four pillars: the data, the analysis code, the statistical methods, and the software which implements the methods. Changes to any one of these pillars will affect analysis results. We focus our attention on the often overlooked fourth pillar: the exact set of software – including the specific versions thereof – used to perform an analysis. We call this the *software context* of the analysis.

Management of software contexts is relevant to many activities in scientific computing. Collaborators often need to synchronize package versions to guarantee comparability of their results (Gentleman and Temple Lang 2004; Ushey, McPherson, Cheng, Atkins, and Allaire 2016; FitzJohn, Pennell, Zanne, and Cornwell 2014). A package author might switch between contexts when maintaining multiple software branches, or when alternating between development and analysis work (Ooms 2013; Gentleman *et al.* 2004). Large collaborative or enterprise organizations might formally support this synchronization by automating the testing and publication of a canonical cohort of package versions (Revolution Analytics 2014). Finally, analysts attempting to reproduce published computational results often need to approximate the software context of the original authors (Ushey *et al.* 2016).

The above examples suggest three general software requirements for managing software contexts. First, users need to locally manage software contexts, including the ability to create, populate, and conveniently switch between them. Second, specific package versions must be directly installable, including from non-repository sources (Wickham and Chang 2017) and historical releases (Revolution Analytics 2014). Finally, organizations and individuals should be able to define, test, and publish specialized, version-specific package cohorts (Ushey *et al.* 2016; Revolution Analytics 2014; de Vries 2017). Users should be able to install packages from such a cohort through standard mechanisms, or use the cohort to seed a new software context.

Ushey *et al.* (2016)'s **packrat** package focuses on packages installed and used by a particular analysis project. It provides the ability to maintain separate package libraries for different projects, and to bundle the package tarballs themselves with a project's analysis script(s) for preservation and manual distribution, resulting in what Gentleman and Temple Lang (2004) call *compendiums*. Revolution Analytics (2014)'s approach with **MRAN**, on the other hand, focuses on preserving the state of the Comprehensive R Archive Network (CRAN) in its entirety at specific time-points. These *snapshots* are preserved as live repositories which can be installed from at a later date as necessary. Other programming languages have similar tools, including Perl (Miyagawa 2011), (Trout 2007) and Python (Bicking 2007), though they typically focus on development environments rather than collaboration or analysis reproducibility. Eddelbuettel (2017)'s **drat** R allows users to bypass the question of installing non-repository sources by providing tooling to create light-weight package repositories on top of GitHub user accounts.

We present a framework for managing and distributing software contexts for the R statistical computing language. The framework provides two separate but integrated functionalities. First, we provide tools for locally installing and managing software contexts, including the retrieval of exact, specified package versions from a variety of sources. Secondly, we provide tools for publishing (optionally validated) version-specific R package cohorts which can be deployed locally as self-contained software contexts.

2. A brief example

Suppose we are embarking on a large-scale collaboration with a group of other researchers. Recognizing the need for comparable results, we all agree to standardize our software contexts on the package versions available from CRAN and Bioconductor on that date.

After installing or updating the relevant packages, we can create a package manifest which describes our set of currently installed packages – i.e., our current *package library* – or the subset of those currently loaded into our R session. We have included such a manifest in the file "manforpaper.rman" (see supplementary files).

```
R> library("switchr")
R> pkg_man <- loadManifest("manforpaper.rman")
R> pkg_man

A seeding manifest (SessionManifest object)

Describes a cohort of 6 package versions.
6 packages are listed in the underlying package manifest

Package versions:
          name version
1 randomForest  4.6-10
2         nlme 3.1-120
3        git2r  0.10.1
4 latticeExtra  0.6-26
5      lattice 0.20-31
6 RColorBrewer   1.1-2
```

This manifest will act as a generalized repository, allowing collaborators to consistently install the same package versions across time and physical distance. We could also construct a manifest directly from the cohort of packages available from one or more package repositories – e.g., CRAN and Bioconductor – or from a 'SessionInfo' object. We discuss the details of how package manifests provide the cornerstone of our framework in Section 2.

To ease our collaborators' use of the manifest, we can use the **switchrGist** package (Becker 2017c) to publish the manifest as a GitHub gist:

```
R> library("switchrGist")
R> gisturl <- publishManifest(pkg_man, Gist())
```

Now suppose a collaborator wishes to install the chosen package versions using our manifest. She can avoid overwriting currently installed packages with those chosen for our collaboration by *switching to* a new package library and *seeding* it with the package manifest (installation output from initial seeding omitted for brevity).

```
R> switchTo("CollabEnv", seed = pkg_man)

Found existing switchr context. Ignoring seed value
Switched to the 'CollabEnv' computing environment.
33 packages are currently available.
Packages installed in your site library are suppressed.
To switch back to your previous environment type switchBack()
```

Switching to a package library has three primary effects. First, it unloads any currently loaded packages from the R session. Next, it resolves the specified name into a library, creating a new one if no library with that name exists. If a new library is being created and a *seed* is provided, the packages listed in the seed are automatically installed during the creation process; seeds are ignored in the case of a pre-existing library. Finally, it configures the current R session to use the specified library. After switching, the collaborator can use the specified packages without affecting her default – or any other – library.

The collaborator would then proceed to work on the project by using packages within the "CollabEnv" library. To work on something else within the same R session, she can *switch back* to the previous library, which was unaffected by both the seeding and any subsequent package installation:

```
R> switchBack()

Reverted to the 'original' computing environment.
147 packages are currently available.
To switch back to your previous environment type switchBack()
```

When returning to working on the collaboration – either in the same R session or within a different one – the collaborator simply switches to her existing, specialized library:

```
R> switchTo("CollabEnv")
```

Alternatively, our collaboration might decide that a shared and (potentially) evolving set of package versions serves our purpose better. In this case, we can use package **GRANBase** (Becker, Barr, and Kulkarni 2017) to create a traditional, validated package repository from a package manifest:

```
R> library("GRANBase")
R> repo <- makeRepo(pkg_man, basedir = "./collabrepos",
+    repo_name = "CollabPkgs", check_test = FALSE, install_test = FALSE)
R> tail(available.packages(repo)[, c("Package", "Version")])

             Package        Version
git2r        "git2r"        "0.10.1"
latticeExtra "latticeExtra" "0.6-26"
lattice      "lattice"      "0.20-31"
nlme         "nlme"         "3.1-120"
randomForest "randomForest" "4.6-10"
switchr      "switchr"      "0.12.6"
```

2.1. Representing package cohorts via generalized repositories

Every R workflow involves specific cohorts of packages. Installed packages are collected into package libraries, which dictate the set of packages loadable within R. Loaded and attached packages control the set of functionality available to the user within his or her R session.

The cohorts of packages which pass check together define the contents of the CRAN and Bioconductor package repositories, determining which packages – and versions thereof – users can install via standard machinery.

Package repositories provide a natural, existing mechanism for representing and publishing cohorts of packages beyond their narrow – but indispensable – current role of defining the R software ecosystem at large. A package repository is essentially a mapping between a set of package names and one or more pre-built archives of the package source code or binaries. Using a package repository, we can define and publish arbitrary package cohorts, which end-users can install via the standard package installation machinery in R. For example, a cohort might correspond to the set of package versions used to generate the results of a single publication. Broader application of package repositories would enhance reproducibility, package development and collaboration.

We generalize the concept of package repositories via *package manifests*. Package manifests define package cohorts – including the information necessary to retrieve and install the packages' source code – decoupled from the pre-built tarballs stored in standard repositories. Instead, package sources can reside in virtually any form, including directories under public (GitHub, Bitbucket) or private source control, a package within a standard repository or the CRAN archive, or more generally a Web accessible directory or tarball. This decoupling allows users to create, install from, and publish package cohorts – including those that contain packages or versions thereof which are not available in standard repositories – without specialized hosting.

Via the manifest abstraction, we can operate at the level of entire package cohorts (as approximations of software contexts), rather than individual packages. Our framework allows users to bi-directionally and reversibly transform package cohorts between three forms: abstract manifests, installed package libraries, and standard package repositories, as pictured in Figure 1. This allows users to describe, share, publish, locally recreate, and use software contexts directly (R version, OS, and external programs not withstanding).

A *seeding manifest* represents a filtered subset of a package manifest. Filtering allows users to define and install a subset of a larger package manifest, such as one representing a community repository. Currently supported filters are package, indicating inclusion in the subset, and version, indicating an exact version of the associated package. We use seeding manifests to represent both package libraries – the set of installed package versions – and the set of packages loaded within the current R session, along with the information necessary to reinstall that set of packages elsewhere.

2.2. An abstraction for managing package libraries

Our *switching* abstraction, presented briefly in Section 2, combines all activities necessary for the user to begin using the specified library – whether or not it exists prior to the switch. These include: altering the library path (the location on disk where R installs and loads packages), unloading currently loaded packages, and, if necessary, creating the library and *seeding* it with a set of packages.

Seeding a package library, e.g., with a seeding manifest, automatically installs the selected package versions into the library during creation. This provides a convenient mechanism for recreating the R package portion of a published software context. In principle, we can seed libraries with any object which specifies a set of package versions along with their locations,

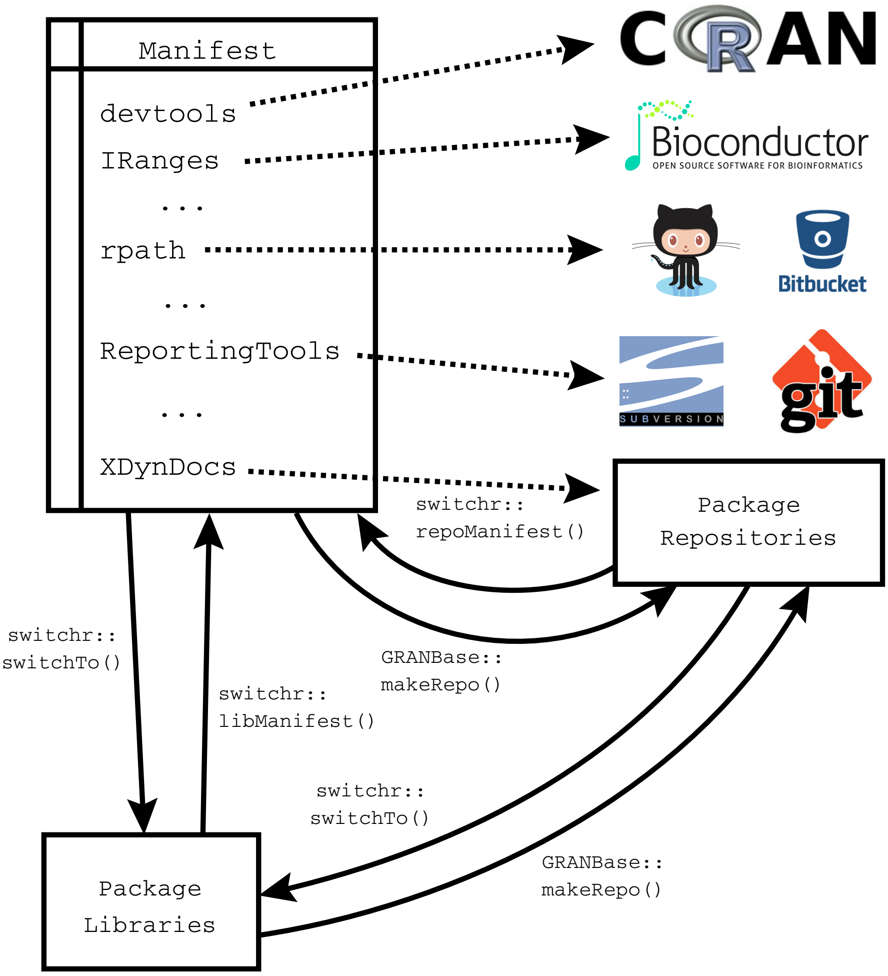

Figure 1: Our framework centers around representing package cohorts as manifests. These manifests can be transformed into useful forms, such as package libraries, and validated repositories.

though typically a package or seeding manifest is used.

Alternatively, the user can derive a library from an existing one by either *branching* or *inheritance*. If a library inherits from another, the packages installed in the derived library override those in the parent, while other packages in the parent remain available. Branching, on the other hand, copies the installed contents of an existing library into the new one but otherwise preserves no relationship between them.

Updating package versions within a library mid-project can be costly, potentially involving the regeneration of results or the modification of analysis or package code (Ooms 2013). Freezing package versions over the course of long-running projects, however, introduces a risk of generating incorrect results *which would have been correct if generated using up-to-date software*.

Weighing the costs and benefits of updating against those of using currently installed versions is an important aspect of library management. The optimal strategy will vary greatly depending on the needs of specific projects or users. We facilitate these decisions by introducing the concept of an *update risk report* which summarizes the available updates and, to the extent possible, their potential impacts.

2.3. Extending package libraries with provenance

We saw in Section 2 that a user can export a session or library to a package manifest. Traditional package libraries, however, do not contain provenance information regarding how – and from where – the package was installed. Without this information, construction of an accurate manifest is difficult.

Our extension of the base R installation mechanism, which is discussed more generally in Section 2.5, automatically records provenance information for packages as they are installed, generalizing the approach taken by Wickham and Chang (2017) in **devtools**'s `install_github()` function. With package provenance recorded at installation time – supplemented via heuristics for packages installed via other methods – we can transform a set of installed packages directly into a seeding (or package) manifest. This allows full, bi-directional conversion between the package library and package manifest representations of a given cohort.

The user can publish an exported package manifest to the Web, e.g., as a GitHub gist or a simple Web-hosted file, allowing others to immediately recreate or install from the described cohort. This allows users to effectively publish package libraries, ensuring others, whether they be current collaborators or future researchers attempting to reproduce our results, can recreate them. Package cohorts can also be published in the form of a traditional, validated package repository which we discuss in more detail in Section 2.6.

2.4. Just-in-time repositories

Many package versions reside outside the package ecosystem defined by a given set of repositories. For example, development package versions might reside on GitHub, while superseded historical versions might be found in the CRAN Archive or Bioconductor SVN, etc. Installing these can be challenging, particularly in the face of dependencies between such packages. We introduce *just-in-time* (JIT) package repositories to mitigate these challenges and allow direct, dependency-aware package installation in the absence of a pre-existing repository.

JIT repositories are transient, local package repositories, which are constructed at installation time and populated with only the requested packages and their (potentially non-repository-available) dependencies. Once the JIT repository is created, actual installation can be delegated to R's established, core installation machinery, as we do in the `install_packages()` function in our **switchr** package (Becker 2017b).

The JIT repository mechanism does not provide any CRAN- or Bioconductor-like guarantees that a package will install on a particular OS, or that a cohort of package versions are compatible. It does, however, allow users to conveniently install cohorts of inter-dependent packages as-is from a combination of repository and non-repository sources.

2.5. Extending R's package installation mechanism

Leveraging JIT repositories, **switchr** provides an extension to R's package installation mechanism with three key features:

1. Installation of packages from package manifests (including dependency support);

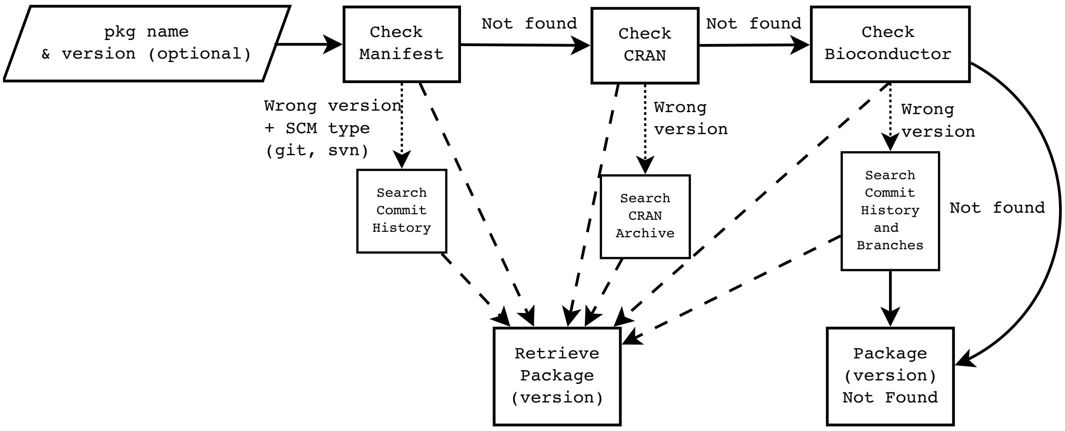

Figure 2: Package versions are searched for in three stages. First the manifest is checked, including SCM history for git and SVN package locations. Next, CRAN and the CRAN archive are searched. Finally, Bioconductor repositories and SVN histories are searched.

2. Installation of specific package versions – including non-current ones – from most types of package locations; and
3. Automatic recording of package provenance during installation.

We provide a mechanism which supports installing packages – or specified versions thereof – from package manifests, but which utilizes R's core, existing installation machinery (i.e., `install.packages()`) under the hood. This maintains a single code-path for installation logic internally while expanding the options of end-users.

When installing specific package versions, **switchr** automatically searches through a number of different locations. These include current and historical CRAN releases, the package repositories for all Bioconductor releases, and the commit history of SCM systems, including the Bioconductor SVN and any listed in the relevant entry in the package manifest.

The specific search algorithm happens in three stages: searching the manifest, searching CRAN, and searching Bioconductor, as pictured in Figure 2.

When retrieving historical releases from SCM commit history, many commits can define the same package version. To remove this ambiguity, we define the commit associated with a package version number to be the *earliest* commit with that version listed in the package's DESCRIPTION file. This models the process whereby developers might make changes without changing the version number, but those changes are not (ever) reflected in that version of the package.

2.6. Building and testing package cohorts into validated repositories

Packages or package versions which appear together in a package or seeding manifest are not, in general, guaranteed to be compatible with each other. This is particularly true when the manifest points to the SCM locations of development versions of multiple packages. For important cohorts of packages, it is beneficial that they be built and tested together – a service that traditional, validated repositories such as CRAN and Bioconductor provide.

We can construct validated package repositories from package manifests. These repositories are tested *incrementally*, meaning that when testing a cohort of packages, only those packages which have been updated since they last passed – or have dependencies which have – are checked. A detailed build report is automatically generated and placed within the `contrib` directory of the new repository. These features facilitate frequent, automated testing via continuous integration systems such as Jenkins (Kawaguchi, Bayer, and Croy 2014) or Travis (Travis CI GmbH 2014). We note here that while our framework supports constructing an actual repository, a manifest which simply points directly to the pre-built tarballs which passed the integration testing would serve a similar purpose and is also supported.

3. Results

Here we present some example applications of our framework. These include the recreation of historically published results, the retrieval and installation of packages and their dependencies from GitHub, and the publication of package manifests for use by the wider community.

3.1. Recreating Anders and Huber's DESeq paper four years later

We now apply **switchr** to reproduce a published result. Specifically, we recreate a subset of from Anders and Huber (2010)'s paper presenting methods implemented in the **DESeq** package[1]. The authors present many results in their paper. We focus here on one: the number of differentially expressed genes found within their fruit fly data.

The authors' original code will not run under modern versions of **DESeq** (Becker 2014) due to evolution of the API. A direct port of the code to the new API yields radically different results, with approximately half as many genes identified. Thus, we cannot assess the strict replicability of their results using modern versions of the package.

Fortunately, Anders and Huber provided code, data, and `sessionInfo()` output within their supplementary materials. With **switchr** – and access to the correct version of R – the information they provide is sufficient to reproduce their results.

Given a compatible version of R, our first step is to reproduce the authors' software context. **switchr** provides two ways to *switch to* a compatible context. The first is to seed a package library directly with the published `sessionInfo()` output text (console output omitted for brevity here and below):

```
R> sessInfo <- readLines("DESeqSession.txt")
R> switchTo("DESeqRepro", seed = sessInfo)
```

The exact package versions listed in `sessInfo` are retrieved and installed into the "DESeqRepro" library. This code, however, requires the work of locating, retrieving, and building the packages to be duplicated on each machine. This is inefficient if many users will be recreating the same context; we can perform these retrievals once and publish the resulting cohort as a package repository via either **GRANBase**'s `makeRepo()` function for possibly validated repositories, or **switchr**'s `lazyRepo()` function for unvalidated JIT-style repositories. In this case, the packages Huber and Anders used are not installable in R versions 3.1.0 or later,

[1] With generous permission from the authors.

which the current version of **GRANBase** requires; `makeRepo()` can still be used but validation must be turned off, as shown below, rendering it roughly equivalent to simply creating a JIT repository via `lazyRepo()`.

```
R> sessInfo <- parseSessionInfoString(readLines("DESeqSession.txt"))
R> sman <- makeSeedMan(sessInfo)
R> makeRepo(sman,
+    basedir = file.path(tempdir(), "10.1186/gb-2010-11-10-r106"),
+    repo_name = "DESeqRepo", check_test = FALSE, install_test = FALSE)
```

or instead of `makeRepo()` using

```
R> lazyRepo(sman)
```

We have uploaded the resulting repository to http://research-pub.gene.com/gran/10.1186/gb-2010-11-10-r106, and we presently switch to it:

```
R> library("switchr")
R> repo <- "http://research-pub.gene.com/gran/10.1186/gb-2010-11-10-r106"
R> switchTo("DESeqRepro", seed = repo)
```

As noted above, the above code must be run in a compatible version of R. We achieved this by using a virtual machine, specifically an Amazon Machine Image (AMI), containing R 2.12 with **switchr** installed. The specific AMI we used is freely available here: http://thecloudmarket.com/image/ami-8afc30e2--ubuntu-natty-r-2-12-1.

Having approximately recreated Anders and Huber's original software context, we are ready to reproduce the result[2]:

```
R> suppressMessages(library("DESeq"))
R> countsTableFly <- read.delim("fly_RNA_counts.tsv")
R> condsFly <- c("A", "A", "B", "B")
R> rownames(countsTableFly) <- paste("Gene", 1:nrow(countsTableFly),
+    sep = "_")
R> cdsFly <- newCountDataSet(countsTableFly, condsFly)
R> cdsFly <- estimateSizeFactors(cdsFly)
R> cdsFly <- estimateVarianceFunctions(cdsFly)
R> resFly <- nbinomTest(cdsFly, "A", "B")
R> length(which(resFly$padj < .1))
```

```
[1] 864
```

This agrees with the original results. Thus we have successfully replicated published results which modern software contexts will no longer produce (with good reason).

[2] Unlike other code outputs in this manuscript, the following result was inserted manually, as the manuscript was created in a version of R not compatible with Anders and Huber's original software context.

3.2. GitHub-based packages and package manifests

The **switchr** and **GRANBase** packages provide two very different mechanisms for interacting with packages and package versions not currently hosted in a repository: direct installation and the construction of validated repositories which contain such packages, respectively. We illustrate these approaches below in the context of GitHub-based packages.

General package manifests are created using the `Manifest()` constructor, but for typical GitHub use-cases, the specialized `GitHubManifest()` constructor is more convenient:

```
R> ghman <- GithubManifest("gmbecker/fastdigest", "duncantl/CodeDepends",
+    "gmbecker/RCacheSuite")
R> ghman

A package manifest (PkgManifest object)

Contains 3 packages and 5 dependency repositories

Packages:
         name type
1  fastdigest  git
2 CodeDepends  git
3 RCacheSuite  git
```

To install the development version of Becker (2017a)'s experimental **RCacheSuite** package, and of some of its dependencies – namely his **fastdigest** (Becker 2015) and Temple Lang, Peng, Nolan, and Becker (2017)'s **CodeDepends** – we use the `install_packages()` function. We pass our package manifest as the "repository" from which the packages will be installed (output omitted for brevity).

```
R> install_packages("RCacheSuite", ghman)
```

The above call generates a JIT repository containing the necessary packages and installs them using standard R machinery in a single step.

In this case, the code does not appear substantially different from calling `install_github()` multiple times; manifests, however, can be shared, and can list many more packages than those needed for a particular install. With a sufficiently exhaustive, centralized manifest, only the `install_packages()` call is required.

We can install the packages in the manifest directly by *seeding* a new library with our manifest. By default, seeding installs all packages listed directly in the manifest, but we can restrict the installation to a specific set of packages. Furthermore, when specifying the packages to install directly, we can include packages which appear in the manifest's dependency repositories (Bioconductor and CRAN by default) but not in the manifest itself. The **XML** package (Temple Lang 2015) is such a package in the second expression below.

```
R> switchTo(name = "githubLib", seed = ghman)
R> switchTo(name = "githubLib2", seed = ghman,
+    pkgs = c("fastdigest", "XML"))
```

While **switchr** provides direct installation of package cohorts, the **GRANBase** package allows users or organizations to construct validated repositories from package manifests. This involves passing a package manifest or existing 'GRANRepository' object to the makeRepo() function, as shown below. We refer system administrators to the **GRANBase** documentation for a complete discussion of this process and the options available.

```
R> repo <- makeRepo(ghman, cores = 3, basedir = tempfile("testrepo"))
```

This process generates a standard, validated R package repository, which can be queried via available.packages() and installed from via install.packages() or also via **switchr**'s install_packages(). We prefer the latter, because it records the package provenance.

3.3. Publishing and distributing package manifests

The libManifest() function allows users to "reverse seed" a package library, generating a session or package manifest representing the packages installed therein.

```
R> mani <- libManifest()
```

Similarly, the makeSeedMan() function creates a session or package manifest representing only those packages loaded in a particular R session.

```
R> mani2 <- makeSeedMan()
R> mani2

A seeding manifest (SessionManifest object)

Describes a cohort of 58 package versions.
58 packages are listed in the underlying package manifest

Package versions:
    name         version
1   "RCurl"      "1.95-4.8"
2   "bitops"     "1.0-6"
3   "GRANBase"   "1.6.5"
4   "switchr"    "0.12.6"
... "..."        "..."
54  "magrittr"   "1.5"
55  "lazyeval"   "0.2.0"
56  "tibble"     "1.3.3"
57  "assertthat" "0.2.0"
58  "R6"         "2.2.2"
```

For packages installed using **switchr** – whether manually or while seeding a library – the annotations added to the installed DESCRIPTION files are used to (re-)construct the manifest entry. For those installed via other methods, **switchr** attempts to locate information about the package automatically. By default, package names are matched against existing CRAN and Bioconductor packages. Optionally, users can also provide an existing manifest to be searched as necessary.

We also support the serialization of a package or seeding manifest as a tab-delimited plain-text data file by passing a path to a local file (or connection) to `publishManifest()`.

```
R> fil <- publishManifest(mani2, tempfile("mani"))
R> head(readLines(fil))

[1] "# R manifest"
[2] "# Manifest type: session"
[3] "# Dependency repositories: 5"
[4] "# repo: https://cloud.r-project.org"
[5] "# repo: https://bioconductor.org/packages/3.5/bioc"
[6] "# repo: https://bioconductor.org/packages/3.5/data/annotation"
```

The package information is stored in tabular form within the body of the file, while the type of manifest and dependency repositories are listed within comments within the header. In the case of a seeding manifest, an additional column is added indicating version information.

Our **switchrGist** extension for **switchr** provides a mechanism, built upon the **gistr** package (Vaidyanathan, Ram, and Chamberlain 2017), which publishes manifests directly to GitHub gists when `publishManifest()` is passed a gist target object (constructed via `Gist()`).

```
R> publishManifest(mani, Gist())
```

The `loadManifest()` function reverses the publish operation, loading a saved manifest from the file or gist URL.

4. Discussion

4.1. Limitations

Our framework has some limitations. First, the internal representation of a particular class of R object may change over package versions. This limits our ability to directly compare results generated by running the same code under different software contexts within a single R session. The **BiocGenerics** package (Huber *et al.* 2015) provides the `updateObject()` generic for updating object representations. However, relying on object conversion decreases the probability of a valid comparison.

Secondly, many old package versions are not installable within modern versions of R, and older R versions can be difficult to install themselves. One way to mitigate this issue is to use virtual machines which provide various historical R versions, as we did in Section 3.1. These images would be bare-bones, potentially providing only R and **switchr**, and would allow users to re-use images, rather than build a complete virtual machine for each analysis, as proposed by Howe (2012). We could also take advantage of the Bioconductor AMIs, which correspond to all minor and most micro releases of R (i.e., all Bioconductor releases) since R 2.8 (Bioconductor Team 2014). Another approach is to use **switchr** within the context of a more stringent reproducibility framework, such as *rocker* (Boettiger and Eddelbuettel 2014), which seeks to build R within Docker (Docker Inc. 2013) images.

Finally, installation from a source code repository or archive assumes the ability to build packages from source. In cases where this is difficult or otherwise infeasible, the offending packages can be pre-compiled and placed in a package repository.

4.2. Future directions

Generalizing the base R installation mechanism

From both the technical and conceptual standpoints, our manifest-based installation framework is similar to the one included with R. The PACKAGES(.tgz) file in a package repository is a combination of a centralized package manifest and a cache of package dependency information, tightly coupled with a cohort of pre-built packages.

The core installation mechanism of R could be extended to include some or all of the concepts discussed here – decentralized repositories, versioned installation, direct installation from SCM – with only narrow, targeted changes to existing code. For example, one could add a new field to the PACKAGES file that indicates the location of the package source code and avoid the assumption that the source tarball is present on the same host. This would enable anyone with access to GitHub or other web hosts to define package cohorts and would help democratize scientific software publication.

Mapping seeding manifests to DOIs

Modern publications are typically referenced by *digital object identifiers* (DOIs). With a DOI, one can lookup a publication or its metadata, such as its citations (PILA 2013). A simple, centralized repository mapping DOIs to recreatable software contexts would dramatically increase the strict reproducibility of published results for which the code and data are available. The authors might simply upload their shared software context as a Gist, or make the investment of deploying a centralized package repository, which streamlines installation. Recreating the software context for a particular publication, then, could be as easy as *switching to* a URL, e.g.,

```
R> switchTo("repro", seed = "http://Rlibraries.org/<doi>")
```

Dynamic documents and reproducibility

Gentleman and Temple Lang (2004) argue that for true reproducibility, a dynamic document should be distributed within a *compendium* that also includes the data and software required to run the document. With the ability of **switchr** to recreate historical contexts from package archives and SCM systems, encoding a `sessionInfo()` object, or a seeding manifest, within a dynamic document is essentially equivalent to explicitly including the package sources themselves (the approach pursued by Ushey *et al.* 2016 with their **packrat** system). Gehring and Becker have preliminary, unpublished work in this area which suggests that this encoding can be fully automated within the context of re-runnable dynamic documents.

We also imagine dynamic documents which contain code chunks intended to be run using different package libraries or even R versions. One example of this would be an automated report describing the differences in results when running the same code within different contexts, such as our **DESeq**-based example in Section 3.1. The **switchr** framework, possibly combined with virtual machines, such as the AMIs provided by the Bioconductor Team (2014), gives dynamic document processing systems the flexibility necessary to support this use-case automatically.

Approximating software contexts based on historical CRAN states

As we saw in Section 2, **switchr** enables us to proactively generate a virtual snapshot of the current state of a repository and to materialize the snapshot on a later date. We can also generate snapshots retroactively.

The **crandb** package and database (Csardi 2014) is a prototype system that automatically retains extensive, queryable information about the contents of CRAN. **switchr** accesses **crandb** to enable users to install cohorts of packages corresponding to historical dates, releases of R, or releases of specific packages. This can provide approximate recreation of R-based software contexts for publications which do not provide version information in the form of a seeding manifest or `sessionInfo()` output. Bioconductor could track and provide similar data, further extending the utility of this approach.

5. Availability

We have released our **switchr**, **switchrGist**, and **GRANBase** packages under the Artistic 2.0 open-source software license. Current, up-to-date development versions of their source-code is available on GitHub at `https://github.com/gmbecker/switchr`, `https://github.com/gmbecker/switchrGist`, and `https://github.com/gmbecker/gRAN`, respectively. Stable, release versions of the packages can be found on CRAN.

Acknowledgments

Gabriel Becker and Michael Lawrence are Bioinformatics Scientists, Dept. of Bioinformatics and Computational Biology, Genentech Research and Early Development, South San Francisco, CA; Cory Barr is Interactive Artist and Engineer, Anticlockwork Arts, San Francisco, CA; and Robert Gentleman is Vice President of Computational Biology, 23andMe Inc., Mountain View, CA.

References

Anders S, Huber W (2010). “Differential Expression Analysis for Sequence Count Data.” *Genome Biology*, **11**(10), R106. `doi:10.1186/gb-2010-11-10-r106`.

Becker G (2014). “GRAN and **switchr** Can’t Send You Back in Time, but They Can Send R (Sort of).” URL `http://blog.revolutionanalytics.com/2014/08/gran-and-switchr-cant-send-you-back-in-time-but-they-can-send-r-sort-of.html`.

Becker G (2015). "**fastdigest**: Fast, Low Memory Footprint Digests of R Objects." R package version 0.6-3, URL https://github.com/gmbecker/fastdigest.

Becker G (2017a). "**RCacheSuite**: State and Dependency Aware Caching." R package version 0.0-2, URL https://github.com/gmbecker/RCacheSuite.

Becker G (2017b). ***switchr**: Installing, Managing, and Switching Between Distinct Sets of Installed Packages.* R package version 0.12.6, URL https://CRAN.R-project.org/package=switchr.

Becker G (2017c). ***switchrGist**: Publish Package Manifests to GitHub Gists.* R package version 0.2.2, URL https://CRAN.R-project.org/package=switchrGist.

Becker G, Barr C, Kulkarni D (2017). ***GRANBase**: Creating Continuously Integrated Package Repositories from Manifests.* R package version 1.6.5, URL https://CRAN.R-project.org/package=GRANBase.

Bicking I (2007). "**virtualenv** Python Module." URL https://virtualenv.pypa.io.

Bioconductor Team (2014). "Bioconductor – Cloud AMI." URL http://www.bioconductor.org/help/bioconductor-cloud-ami/.

Boettiger C, Eddelbuettel D (2014). "rocker." URL https://github.com/rocker-org/rocker.

Csardi G (2014). "**crandb**: Query the Unofficial CRAN Metadata Database." R package version 1.0.0, URL https://github.com/metacran/crandb.

de Vries A (2017). "**miniCRAN**: Tools to Create an Internally Consistent, Mini Version of CRAN with Selected Packages Only." R package version 0.2.10, URL http://github.com/andire/miniCRAN.

Docker Inc (2013). "docker – Build, Ship, and Run Any App, Anywhere." URL https://www.docker.com/.

Eddelbuettel D (2017). "**drat**: Drat R Archive Template." R package version 0.1.3, URL http://CRAN.R-project.org/package=drat.

FitzJohn R, Pennell M, Zanne A, Cornwell W (2014). "Reproducible Research is Still a Challenge." URL http://ropensci.org/blog/2014/06/09/reproducibility/.

Gentleman R, Temple Lang D (2004). "Statistical Analyses and Reproducible Research." *Working Paper 2*, Bioconductor Project. URL http://biostats.bepress.com/bioconductor/paper2.

Gentleman RC, Carey VJ, Bates DM, Bolstad B, Dettling M, Dudoit S, Ellis B, Gautier L, Ge Y, Gentry J, Hornik K, Hothorn T, Huber W, Iacus S, Irizarry R, Leisch F, Li C, Maechler M, Rossini AJ, Sawitzki G, Smith C, Smyth G, Tierney L, Yang JY, Zhang J (2004). "Bioconductor: Open Software Development for Computational Biology and Bioinformatics." *Genome Biology*, **5**(10), R80. doi:10.1186/gb-2004-5-10-r80.

Howe B (2012). "Virtual Appliances, Cloud Computing, and Reproducible Research." *Computing in Science and Engineering*, **14**(4), 36–41. doi:10.1109/mcse.2012.62.

Huber W, Carey VJ, Gentleman R, Anders S, Carlson M, Carvalho BS, Bravo HC, Davis S, Gatto L, Girke T, Gottardo R, Hahne F, Hansen KD, Irizarry RA, Lawrence M, Love MI, MacDonald J, Obenchain V, Oleś AK, Pagès H, Reyes A, Shannon P, Smyth GK, Tenenbaum D, Waldron L, Morgan M (2015). "Orchestrating High-Throughput Genomic Analysis with Bioconductor." *Nature Methods*, **12**(2), 115–121. doi:10.1038/nmeth.3252.

Kawaguchi K, Bayer A, Croy RT (2014). "Welcome to Jenkins CI! – Jenkins CI." URL http://jenkins-ci.org/.

Miyagawa T (2011). "Carton Perl Module." URL http://search.cpan.org/~miyagawa/Carton-v1.0.12/lib/Carton.pm.

Ooms J (2013). "Possible Directions for Improving Dependency Versioning in R." *The R Journal*, **5**(1), 197–206. URL https://journal.R-project.org/archive/2013/RJ-2013-019/.

PILA (2013). "crossref.org." URL http://www.crossref.org/.

Revolution Analytics (2014). "**MRAN**: Managed R Archive Network (MRAN)." URL http://mran.revolutionanalytics.com/.

Temple Lang D (2015). ***XML**: Tools for Parsing and Generating XML within R and S-PLUS*. R package version 3.99-0, URL http://www.omegahat.net/RSXML.

Temple Lang D, Peng R, Nolan D, Becker G (2017). "**CodeDepends**: Analysis of R Code for Reproducible Research and Code Comprehension." R package version 0.5-3, URL https://github.com/duncantl/CodeDepends.

Travis CI GmbH (2014). "Travis CI: Free Hosted Continuous Integration Platform for the Open Source Community." URL https://travis-ci.org/.

Trout MS (2007). "local-lib Perl Module." URL https://github.com/Perl-Toolchain-Gang/local-lib.

Ushey K, McPherson J, Cheng J, Atkins A, Allaire JJ (2016). "**packrat**: Reproducible Package Management for R." R package version 0.4.8-1, URL http://rstudio.github.io/packrat/.

Vaidyanathan R, Ram K, Chamberlain S (2017). "**gistr**: Work with GitHub Gists." R package version 0.4.0, URL http://CRAN.R-project.org/package=gistr.

Wickham H, Chang W (2017). "**devtools**: Tools to Make Developing R Code Easier." R package version 1.13.3, URL http://CRAN.R-project.org/package=devtools.

Affiliation:

Gabriel Becker
Dept. of Bioinformatics and Computational Biology
Genentech Research and Early Development
1 DNA Way, South San Francisco, CA
94080 United States of America
E-mail: becker.gabriel@gene.com

Journal of Statistical Software http://www.jstatsoft.org/
published by the Foundation for Open Access Statistics http://www.foastat.org/
November 2017, Volume 82, Issue 1 *Submitted:* 2015-01-12
doi:10.18637/jss.v082.i01 *Accepted:* 2016-08-25