DataDeps.jl: Repeatable Data Setup for Reproducible Data Science

Lyndon White lyndon.white@research.uwa.edu.au
Roberto Togneri roberto.togneri@uwa.edu.au
Wei Liu wei.liu@uwa.edu.au
Mohammed Bennamoun mohammed.bennamoun@uwa.edu.au
35 Stirling Hwy, Crawley 6009 The University of Western Australia

Editor:

Abstract

We present DataDeps.jl: a julia package for the reproducible handling of static datasets to enhance the repeatability of scripts used in the data and computational sciences. It is used to automate the data setup part of running software which accompanies a paper to replicate a result. This step is commonly done manually, which expends time and allows for confusion. This functionality is also useful for other packages which require data to function (e.g. a trained machine learning based model). DataDeps.jl simplifies extending research software by automatically managing the dependencies and makes it easier to run another author's code, thus enhancing the reproducibility of data science research.

Keywords: data management; reproducible science; continuous integration, JuliaLang, software practices,

1. Introduction

In the movement for reproducible sciences there have been two key requests upon authors: **1.** Make your research code public, **2.** Make your data public (Goodman et al., 2014). In practice this alone is not enough to ensure that results can be replicated. To get another author's code running on a your own computing environment is often non-trivial. One aspect of this is data setup: how to acquire the data, and how to connect it to the code.

DataDeps.jl simplifies the data setup step for software written in Julia (Bezanson et al., 2014). DataDeps.jl follows the unix philosophy of doing one job well. It allows the code to depend on data, and have that data automatically downloaded as required. It increases replicability of any scientific code that uses static data (e.g. benchmark datasets). It provides simple methods to orchestrate the data setup: making it easy to create software that works on a new system without any user effort. While it has been argued that the direct replicability of executing the author's code is a poor substitute for independent reproduction (Drummond, 2009), we maintain that being able to run the original code is important for checking, for understanding, for extension, and for future comparisons.

Vandewalle et al. (2009) distinguishes six degrees of replicability for scientific code. The two highest levels require that "The results can be easily reproduced by an independent researcher with at most 15 min of user effort". One can expend much of that time just on setting up the data. This involves reading the instructions, locating the download link,

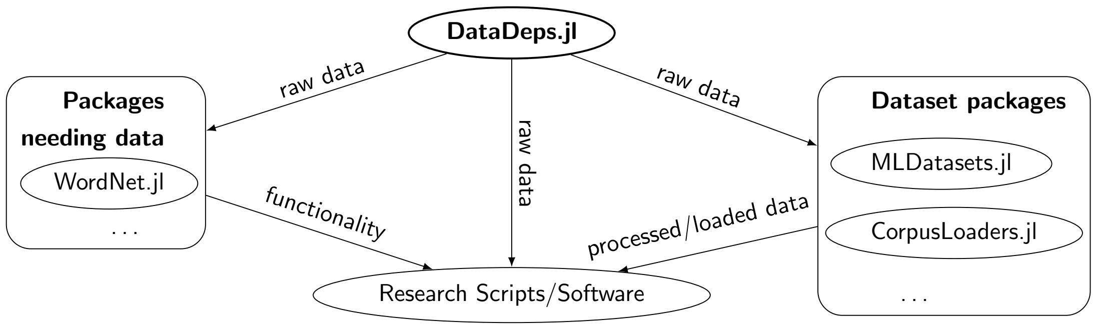

Figure 1: The current package ecosystem depending on DataDeps.jl.

transferring it to the right location, extracting an archive, and identifying how to inform the script as to where the data is located. These tasks are automatable and therefore should be automated, as per the practice "Let the computer do the work" (Wilson et al., 2014).

DataDeps.jl handles the data dependencies, while Pkg and BinDeps.jl[1] (etc.) handle the software dependencies. This makes automated testing possible, e.g., using services such as TravisCI or AppVeyor.[2] Automated testing is already ubiquitous amongst julia users, but rarely for parts where data is involved. A particular advantage over manual data setup, is that automation allow scheduled tests for URL decay (Wren, 2008). If the full deployment process can be automated, given resources, research can be fully and automatically replicated on a clean continuous integration environment.

1.1 Three common issues about research data

DataDeps.jl is designed around solving common issues researchers have with their file-based data. The three key problems that it is particularly intended to address are:

Storage location: Where do I put it? Should it be on the local disk (small) or the network file-store (slow)? If I move it, am I going to have to reconfigure things?

Redistribution: I don't own this data, am I allowed to redistribute it? How will I give credit, and ensure the users know who the original creator was?

Replication: How can I be sure that someone running my code has the same data? What if they download the wrong data, or extract it incorrectly? What if it gets corrupted or has been modified and I am unaware?

2. DataDeps.jl

2.1 Ecosystem

DataDeps.jl is part of a package ecosystem as shown in Figure 1. It can be used directly by research software, to access the data they depend upon for e.g. evaluations. Packages such as MLDatasets.jl[3] provide more convenient accesses with suitable preprocessing for commonly used datasets. These packages currently use DataDeps.jl as a back-end. Research

1. `https://docs.julialang.org/en/stable/stdlib/pkg/`, `https://github.com/JuliaLang/BinDeps.jl`
2. `https://travis-ci.org/`, `https://ci.appveyor.com/`
3. `https://github.com/JuliaML/MLDatasets.jl`

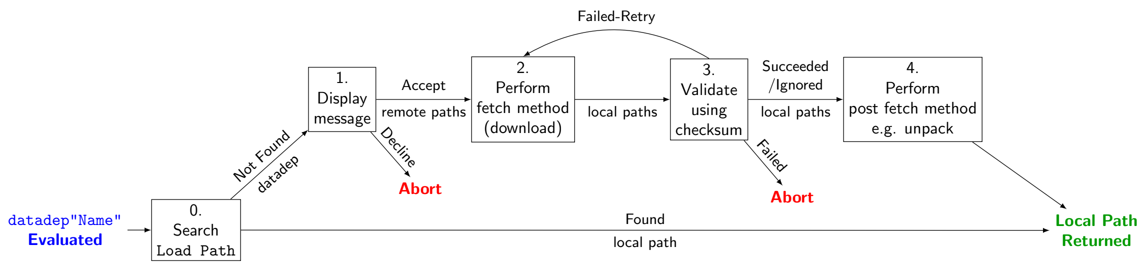

Figure 2: The process that is executed when a data dependency is accessed by name.

code also might use DataDeps.jl indirectly by making use of packages, such as WordNet.jl[4] which currently uses DataDeps.jl to ensure it has the data it depends on to function (see Section 2.4). Packages and research code alike depend on data, and DataDeps.jl exists to fill that need.

2.2 Functionality

Once the dependency is declared, data can accessed by name using a datadep string written `datadep"Name"`. This can treated just like a filepath string, however it is actually a string macro. At compile time it is replaced with a block of code which performs the operation shown in Figure 2. This operation always returns an absolute path string to the data, even that means the data must be download and placed at that path first.

DataDeps.jl solves the issues in Section 1.1 as follows:

Storage location: A data dependency is referred to by name, which is resolved to a path on disk by searching a number of locations. The locations search is configurable.

Redistribution: DataDeps.jl downloads the package from its original source so it is not redistributed. A prompt is shown to the user before download, which can be set to display information such as the orignal author and any papers to cite etc.

Replication: when a dependency is declared, the creator specified the URL to fetch from and post fetch processing to be done (e.g. extraction). This removed the chance for human error. To ensure the data is exactly as it was originally checksum is used.

DataDeps.jl is primarily focused on public, static data. For researchers who are using private data, or collecting that data while developing the scripts, a manual option is provided; which only includes the **Storage Location** functionality. They can still refer to it using the `datadep"Name"`, but it will not be automatically downloaded. During publication the researcher can upload their data to an archival repository and update the registration.

2.3 Similar Tools

Package managers and build tools can be used to create adhoc solutions, but these solution will often be harder to use and fail to address one or more of the concerns in Section 1.1. Data warehousing tools, and live data APIs; work well with continuous streams of data; but they are not suitable for simple static datasets that available as a collection of files.

4. `https://github.com/JuliaText/WordNet.jl`

Quilt[5] is a more similar tool. In contrast to DataDeps.jl, Quilt uses one centralised data-store, to which users upload the data, and they can then download and use the data as a software package. It does not directly attempt to handle any **Storage Location**, or **Redistribution** issues. Quilt does offer some advantages over DataDeps.jl: excellent convenience methods for some (currently only tabular) file formats, and also handling data versioning. At present DataDeps.jl does not handle versioning, being focused on static data.

2.4 Case Studies

Research Paper: White et al. (2016) We criticize our own prior work here, so as to avoid casting aspersions on others. We consider it's limitations and how it would have been improved had it used DataDeps.jl. Two version of the script were provided [6] one with just the source code, and the other also including 3GB of data. It's license goes to pains to explain which files it covers and which it does not (the data), and to explain the ownership of the data. DataDeps.jl would avoid the need to include the data, and would make the ownership clear during setup. Further sharing the source code alone would have been enough, the data would have been downloaded when (and only if) it is required. The scripts themselves have relative paths hard-coded. If the data is moved (e.g. to a larger disk) they will break. Using DataDeps.jl to refer to the data by name would solve this.

Research Tool: WordNet.jl WordNet.jl is the Julia binding for the WordNet tool (Miller, 1995). As of PR #8[7] it now uses DataDeps.jl. It depends on having the WordNet database. Previously, after installing the software using the package manager, the user had to manually download and set this up. The WordNet.jl author previously had concerns about handling the data. Including it would inflate the repository size, and result in the data being installed to an unreasonable location. They were also worried that redistributing would violate the copyright. The manual instructions for downloading and extracting the data included multiple points of possible confusion. The gzipped tarball must be correctly extracted. The user must know to pass in the *grand-parent* directory of the database files. Using DataDeps.jl all these issues have now been solved.

3. Concluding Remarks

DataDeps.jl aims to help solve reproducibility issues in data driven research by automating the data setup step. It is hoped that by supporting good practices, with tools like DataDeps.jl, now for the still young Julia programming language better scientific code can be written in the future .

4. Availability and Requirements

DataDeps.jl works on Windows 7+, Linux, and Mac OS X, with Julia 0.6. It depends on HTTP.jl, Reexport.jl, and SHA.jl, which are automatically installed. The source code, documentation, and issue tracker are at `https://github.com/oxinabox/DataDeps.jl`.

5. `https://github.com/quiltdata/quilt`
6. Source code and data provided at `http://white.ucc.asn.au/publications/White2016BOWgen/`
7. `https://github.com/JuliaText/WordNet.jl/pull/8`

References

Jeff Bezanson, Alan Edelman, Stefan Karpinski, and Viral B. Shah. Julia: A fresh approach to numerical computing. 2014. URL `http://arxiv.org/abs/1411.1607`.

Chris Drummond. Replicability is not reproducibility: nor is it good science. *Proceedings of the Evaluation Methods for Machine Learning Workshop at the 26th ICML*, 2009. URL `https://nparc.nrc-cnrc.gc.ca/eng/view/accepted/?id=54187bb4-a8e2-4ce9-9067-9938dc403bea`.

Alyssa Goodman, Alberto Pepe, Alexander W. Blocker, Christine L. Borgman, Kyle Cranmer, Merce Crosas, Rosanne Di Stefano, Yolanda Gil, Paul Groth, Margaret Hedstrom, David W. Hogg, Vinay Kashyap, Ashish Mahabal, Aneta Siemiginowska, and Aleksandra Slavkovic. Ten simple rules for the care and feeding of scientific data. *PLOS Computational Biology*, 10(4):1–5, 04 2014. doi: 10.1371/journal.pcbi.1003542.

George A Miller. Wordnet: a lexical database for english. *Communications of the ACM*, 38(11):39–41, 1995. URL `http://mooo.cf/20140731/Reference/1995%20-%20Miller%20-%20WordNet%20a%20lexical%20database%20for%20English.pdf`.

P. Vandewalle, J. Kovacevic, and M. Vetterli. Reproducible research in signal processing. *IEEE Signal Processing Magazine*, 26(3):37–47, May 2009. ISSN 1053-5888. doi: 10.1109/MSP.2009.932122.

Lyndon White, Roberto Togneri, Wei Liu, and Mohammed Bennamoun. Generating bags of words from the sums of their word embeddings. In *17th International Conference on Intelligent Text Processing and Computational Linguistics (CICLing)*, 2016.

Greg Wilson, D. A. Aruliah, C. Titus Brown, Neil P. Chue Hong, Matt Davis, Richard T. Guy, Steven H. D. Haddock, Kathryn D. Huff, Ian M. Mitchell, Mark D. Plumbley, Ben Waugh, Ethan P. White, and Paul Wilson. Best practices for scientific computing. *PLOS Biology*, 12(1):1–7, 01 2014. doi: 10.1371/journal.pbio.1001745. URL `https://doi.org/10.1371/journal.pbio.1001745`.

Jonathan D Wren. Url decay in medline: a 4-year follow-up study. *Bioinformatics*, 24(11):1381–1385, 2008. URL `https://academic.oup.com/bioinformatics/article/24/11/1381/191025`.