DataDeps.jl: Repeatable Data Setup for Reproducible Data Science

Editor:

Abstract

We present a framework DataDeps.jl for the reproducible handling of static datasets to enhance the repeatability of software scripts used in the data and computational sciences. DataDeps.jl is a library for the Julia programming language. 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 via traditional means of a software dependency, as the extension does not have to worry about ensuring the data is setup for its dependency. DataDeps.jl makes it easier to rerun another authors code, thus enhancing the reproducibility of data science research.

Keywords: Julia; data; data management; data dependencies; reproducible science; downloading; computational environment setup; continuous integration; software practices.

1. Introduction

In the movement for reproducible data and computational sciences there have been two key additional requests from authors: 1. Make your research code public, 2. Make your data public (Goodman et al., 2014). In practice however, this alone is not enough to ensure that even the purely computational results can be replicated. To get another author's code running on a new machine is often non-trivial. One is more likely to see errors preventing the code from running, than in the results. One aspect of this is the data setup, how to acquire the data, and how to connect it to the code.

DataDeps.jl simplifies the automation of the data setup step for Julia (Bezanson et al., 2014) packages and research software. It allows the code to depend on data, and have that data automatically downloaded as required. DataDeps.jl is a compact tool with a single job, following the unix philosophy of doing one job well. It increases repeatability of any scientific code that uses data. It does this by decreasing the effort to get such a program working on a new system, while also decreasing the effort to write the code in the first place.

Vandewalle et al. (2009) distinguishes 6 degrees of reproducibility for scientific code. To achieve either of the 2 highest levels, requires that "The results can be easily reproduced by an independent researcher with at most 15 min of user effort". It is our experience that one can often expend much of that time just on setting up the data. This involves reading the instructions, locating the download link, 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 therefore should be automated; to save user time, and remove the

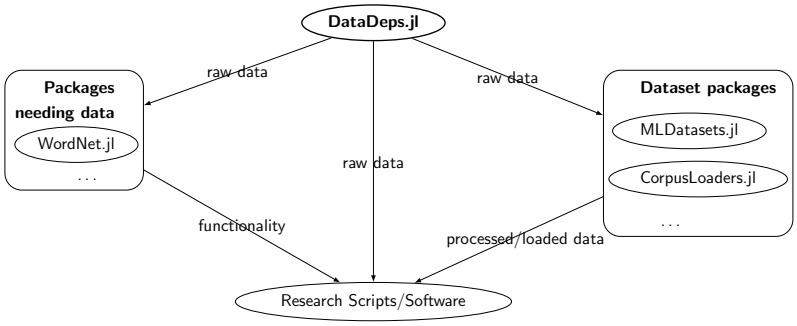

Figure 1: The package ecosystem using DataDeps.jl. DataDeps.jl can be used directly by research code to produce data; or it can be used indirectly by making use of a package which uses DataDeps.jl to manage its data dependencies.

opportunity for mistakes, as per the key practice identified by Wilson et al. (2014) "let the computer do the work".

Automating the data setup is part of achieving full automation of the setup in a new environment. DataDeps.jl handles the setup of data dependencies in Julia, while BinDeps.jl (and other packages) handle the binary software dependency. The automation of the full setup is required to make automated testing possible, for example using Continuous Integration testing services, such as TravisCI or AppVeyor. Automated testing is already ubiquitous in use amongst researchers and developers using Julia, but rarely for parts where data is involved. If the full setup can be automated, such that it can run on a clean CI environment, it is almost certain that any human trying to reproduce your work using your code will be able to do so with little effort.

2. DataDeps.jl

2.1 Ecosystem

As shown in Figure 1, DataDeps.jl exists as a higher level dependency. Some research code employing commonly used datasets will not directly use DataDeps.jl at all; rather it will depend on a dataset loading package such as MLDatasets.jl, or CorpusLoaders.jl, which in turn use DataDeps.jl as their back-end to actually manage their data. These dataset packages provide a more friendly access to data that has been cleanly loaded into Julia data structures. For more obscure, or self-created data sets the research code would employ DataDeps.jl directly, and handle the loading of the data itself. Finally, DataDeps.jl is also used by packages which require data to perform their functionality. Examples include WordNet.jl which provides functions for non-trivial lookups on the WordNet (Miller, 1995) database, or any packages which require a trained machine-learning based model – the data in this case being the serialised model. Packages and research code alike depend on data, and DataDeps.jl exists to fill that need.

2.2 Three common questions about research data

DataDeps.jl is designed around solving common issues researchers have with their file-based data. The three questions 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, I'm going to have to reconfigure things

Redistribution Is it ok to include it with my work? I don't own copyright on this data.
- Am I allowed to redistribute this data?
- What if people getting the data from me as part of my experiments don't realize the original creator?

Replication How can I be sure someone running my code has the same data?
- It is possible for later users to download the wrong data
- It is possible for data to be corrupted, or modified (even maliciously).

These three issues are solved by DataDeps.jl.

2.3 Functionality

When declaring a data dependency the developer needs only to declare a data registration block. For a normal data dependency this is a small piece of code consisting of:

Name used to refer to the dependency in code, as in the described `datadep"NAME"`. This solves the **Storage Location** issues, as will be discussed below data is referred to by name, not by path. The data can be moved to any where on the load path, without changing the code. The load path can be configured using an environment variable, allowing it to differ across environments, by default it includes a very wide range of common locations.

Remote path a path or a list of paths (normally URLs) to remote copies of the data. By fetching the data from a remote location, the issues of **Redistribution** are avoided, as you are not distributing data merely using existing links. This also avoids issues with storing data in version control, which causes problems for many version control systems.

Message the message to be displayed to the user before any download occurs, it is followed by requiring user acceptance to continue. This allows the user to be informed of the data's true origin, and display other key info like papers about the data to be cited. Together with fetching from a remote path, this solves the **Redistribution** issue.

Checksum a checksum (or list of checksums) for the files being fetch. Via external packages a wide variety of checksums can be supported, including MD5 and all versions of SHA. This, together with the automation of the whole process, solves the **Replication** issue, ensuring the data the user receives is as it was when the software was developed. If the checksum is not provided, then one is calculated, together with a warning that this line should be added to the registration block; allowing the researcher to avoid having to use unfamiliar tools to calculate the checksum.

Post fetch method a function (or list of functions) to call on the fetched files. Most commonly used to unpack an archive. This completes the automation of data setup, contribution towards the **Replication** issue.

The registration block is effectively metadata, instructing on how to obtain the data.

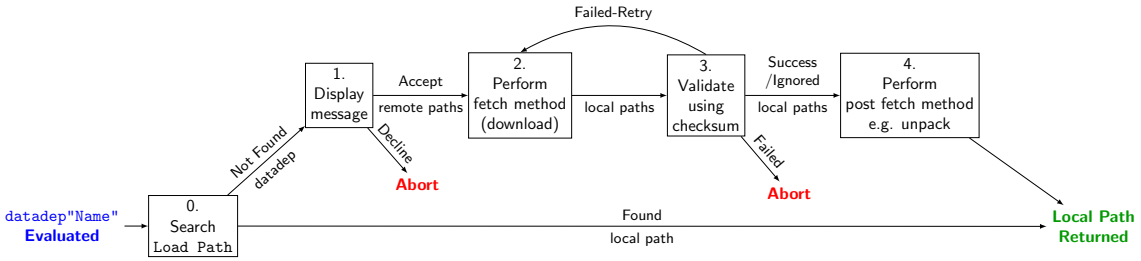

Figure 2: The core process for resolving a data dependency. Each step in the top path is linked to one of the properties declared in the registration block.

Once the data dependency has been declared using the registration block, it can be referred to by name using a datadep string, written `datadep"Name"`. It can be treated just like it were an absolute path to the data, however it is actually a a string macro (related to LISP's reader macro). At compile time it is replaced with a block of code which performs the operation shown in Figure 2. It always resolves to a being a string to an absolute path to the data, even if that means it must download the data first.

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; the registration block here would only include a name and a message. This allows users to manage the data themselves by putting a folder somewhere on the load path. They can still refer to it using the `datadep"Name"`. If it is not found, the message is displayed which should include instructions on fetching it manually. Before that research is submitted for publications, the user can upload their data to a repository such as DataDryad, FigShare or another archival repository, and update the registration.

The use of archival repositories as the remotepath host should combat URL decay (Wren, 2008). Though it remains that URLs in DataDeps.jl are just as vulnerable to URL decay as those in manual instructions. Unlike manual instructions however, periodic automated tests can easily be use to confirm if the URLs remain valid.

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 setting up for good practices now for the still young Julia programming language, better scientific code can be written in the future.

4. Availability and Requirements

DataDeps.jl is verified to work on Windows 7+, Linux, Mac OS X, with Julia 0.6. It depends on HTTP.jl, Reexport.jl, and SHA.jl, which are automatically installed when the package is installed through the normal Julia package manager. The source code, documentation, and issue tracker can be found at `https://github.com/oxinabox/DataDeps.jl`.

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.

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. URL https://doi.org/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.

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.