Using ReproZip for Reproducibility and Library Services

Vicky Steeves[1], Rémi Rampin[2], and Fernando Chirigati[3 4]

Abstract

Achieving research reproducibility is challenging in many ways: there are social and cultural obstacles as well as a constantly changing technical landscape that makes replicating and reproducing research difficult. Users face challenges in reproducing research across different operating systems, in using different versions of software across long projects and among collaborations, and in using publicly available work. The dependencies required to reproduce research can be exceptionally hard to track – in many cases, these dependencies are hidden or nested too deeply to discover, and thus impossible to install on a new machine, which means the percentage of reproducible research remains low.

In this paper, we present ReproZip[5], an open source tool to help overcome the technical difficulties involved in preserving and replicating research, applications, databases, software, and more. We will examine the current use cases of ReproZip6, ranging from digital humanities to data science. We also explore potential library use cases for ReproZip, particularly in digital libraries and archives, liaison librarianship, and other library services. We believe that libraries and archives can leverage ReproZip to deliver more robust reproducibility services, repository services, as well as enhanced discoverability and preservation of research materials, applications, software, and computational environments.

Keywords

Reproducibility, data management, repository management, digital libraries, digital archiving, software preservation

1 Introduction

Reproducibility is at the core of the research process: it is not only essential for verification and authentication of results, but also for driving a field forward. If a work is reproducible, newcomers to the field can easily learn methods and experienced researchers can build upon it. Despite the widespread attention drawn to the subject following the *Reproducibility Project: Psychology*, carried out by the Center for Open Science (Open Science Collaboration, 2015), reproducibility still remains an elusive target for many researchers (Goodman, Fanelli, and Ioannidis 2016). While sharing research and materials should be easily achieved by using institutional or domain-specific repositories, this does not guarantee reproducibility.

As research has become increasingly reliant on digital tools, the challenges in reproducibility have become more difficult. This is where the concept of computational reproducibility becomes important. Before, researchers in the field captured their environments through observation, drawings, photographs, and videos; now, researchers and librarians who endeavor to preserve their work must begin to capture digital environments to achieve reproducibility (Stodden et al. 2016). However, preserving digital environments is exceedingly difficult to do manually. For example, Gronenschild et. al (2012) discussed how the results of data analyses in neuroscience performed with the same application differed based on the operating system:

> We investigated the effects of data processing variables such as FreeSurfer version (v4.3.1, v4.5.0, and v5.0.0), workstation (Macintosh and Hewlett-Packard), and Macintosh operating system version (OS X 10.5 and OS X 10.6). Significant differences were revealed between FreeSurfer version v5.0.0 and the two earlier versions. [...] About a factor two

> smaller differences were detected between Macintosh and Hewlett-Packard workstations and between Mac OS X 10.5 and Mac OS X 10.6. The observed differences are similar in magnitude as effect sizes reported in accuracy evaluations and neurodegenerative studies.

In addition, there may be many unforeseen dependencies for each software or tool, of which different versions from the original configuration may give totally disparate results or not even run. To manually address these problems, collectively known as 'dependency hell,' researchers enter into an error-prone and resource-heavy process. They would have to create a file that encapsulates metadata about their computational environment, including the operating system, hardware architecture, and software library dependencies. Tracking these dependencies is challenging – there are many layers of hardware and software which the average user has no skill or time to examine (Marwick, 2015).

There are a few classes of tools that can be used to capture digital environments. Workflow systems, such as VisTrails[7] and Kepler[8], represent research as an executable diagram, depicting the flow of the different steps and processes of the research (Davidson and Freire, 2008). Workflow systems help researchers conceptualize and manage the analysis process, support researchers by allowing the creation and reuse of analysis tasks, and (more recently) systematically record provenance information for later use. However, workflow systems have a steep learning curve and often represents too much of an 'ask' for users to adopt. In addition, software dependencies are rarely captured by these systems. Virtual machines (VMs) are an emulation of a computer system and provide the functionality of a physical computer. Common implementations involve specialized hardware, software, or a combination of both (Smith and Ravi Nair, 2005). The resulting files, called 'images,' are often large (gigabytes in size) since they encapsulate the entire operating system. While lightweight solutions exist for VMs, such as Vagrant and Docker, the user is again tasked with building these images, which is time-consuming. Another potential solution are configuration management systems (Feller, 2013), where users write 'recipes' that take the form of a requirements file that can be use to automate the installation of dependencies. Again, these files must be created manually by the user and updated as their work takes new shape.

Porting existing research to these current solutions takes too much time and work to be of use in researchers' day-to-day workflow. A class of tools that can automatically capture all the dependencies in the original environment and automatically set them up in another environment is needed to span this gap. There are a few tools that address parts of that gap, such as Provenance-To-Use (Pham et al., 2013), which uses CDE (Code, Data, and Environment packaging for Linux) (Guo, 2012) to create a package that contains the dependencies necessary to reproduce the research. However, users can only reproduce these packages on Linux, otherwise they would need to manually create a virtualization environment, which is less user-friendly and can be cumbersome for complex research.

ReproZip was created to fully bridge this gap in computational reproducibility. It is an open source, freely available program that allows users to make their work completely reproducible, down to the operating system level. ReproZip can be used to reproduce a plethora of applications, including data analysis tools, scripts and software written in any language, graphical tools, interactive tools, client-server applications (including databases), and Jupyter notebooks[9].

ReproZip works in two steps: automatically tracing the execution of work and then packaging all dependencies in a single, distributable package, a .rpz file (Chirigati et al., 2016). The ease of use extends to the unpacking step as well. For others to reproduce research that was packed using ReproZip, they use ReproUnzip, which works by providing different methods of unpacking from which the second user can choose. ReproUnzip then automatically sets up the environment by extracting metadata captured during the initial tracing process and building the new environment automatically. From there, the user only needs to rerun the work or application. Packaging and reproducing can be

achieved in four simple steps — two steps to pack, and two to unpack — and makes reproducibility easier to achieve across research domains as compared to the options referenced above.

Users can heavily customize the way they use ReproZip as well. The original user packing their work has a lot of control in what is included in a .rpz file. ReproZip will automatically identify input files, dependencies, and output files, but users have the option of editing the configuration file to remove or add files, to label items more meaningfully, and more. ReproZip packages are also highly portable, in that it automatically creates a virtual machine for the user – no extra work required beyond one click or command – allowing research to be reproduced across different operating systems. ReproZip also uses an extensible architecture for unpacking, which means anyone can add an unpacker to reproduce research while maintaining compatibility with existing ReproZip packages. For example, if Docker ceased to exist, then another unpacker could be written and added to ReproZip, and the packages would maintain forward compatibility.

While ReproZip has primarily been used in research, in this paper, we explore the many ways in which librarians can use ReproZip, from helping user populations create well-managed, reproducible research, to preserving computational environments, and to building library infrastructure.

2 Technical Infrastructure

ReproZip has two stages, one for packing and another for unpacking. ReproZip creates a small, self-contained package (.rpz file) by automatically identifying, tracking, and capturing all required dependencies of research, applications, databases, and indeed, most digital work (Chirigati et al., 2016). This package can easily be shared, as it is usually quite small, with reviewers, collaborators, or released to the general public (see Figure 1).

Secondary users can unpack the .rpz using ReproUnzip, and reproduce the work on their machine, regardless of operating system (see Figure 2). ReproUnzip then automatically sets up the environment for that user by extracting metadata captured during the initial tracing process and building the environment automatically. ReproUnzip's functionality is not limited to simple reproduction: it also allows users to modify the original work for reuse for new purposes, with very little effort.

2.1 Packing

Currently, users can pack their work using ReproZip on Linux operating systems via the command line. Future work includes adding in a GUI for packing and support for non-Linux environments (see Future Development Work[10]). ReproZip first needs to trace system calls, which identifies all the dependencies necessary to reproduce the environment and the research within. This is accomplished when the user prepends the command ‘reprozip trace’ to the execution of their research. For instance, if a user runs a Python script to analyze data by typing ‘python analysis.py’ in the command line, to trace with ReproZip, the user simply runs ‘reprozip trace python analysis.py’ instead. ReproZip collects information including command-line arguments, environment variables, files read, and files written, and stores everything in a SQLite database.

To detect detailed information about the dependencies, ReproZip keeps strict provenance data. For instance, by observing which files were read and by using the software manager in the operating system, ReproZip can identify the software libraries on which the research depends. ReproZip uses rules to identify the role of files in the execution of the research, such as input files – those files that were only read and do not belong in a software package. All of this information is written to a human-readable YAML configuration file.

The user can then customize the package via the configuration file, which contains all of the required information for reproducing the work including commands, arguments, working directory,

environment variables, operating system information, input files, output files, and software packages. This generated file is enough to create a reproducible package, however users can choose to modify it, e.g. to remove sensitive or proprietary information, give meaningful labels to input and output files, or include additional files to support more reproduction settings.

Lastly, users generate a package using the command 'reprozip pack <package-name> ' which compresses all the dependencies, files, and environmental information into a small .rpz file. This is usually small enough to send to reviewers of papers, deposit into an institutional repository, or simply be saved for archival purposes.

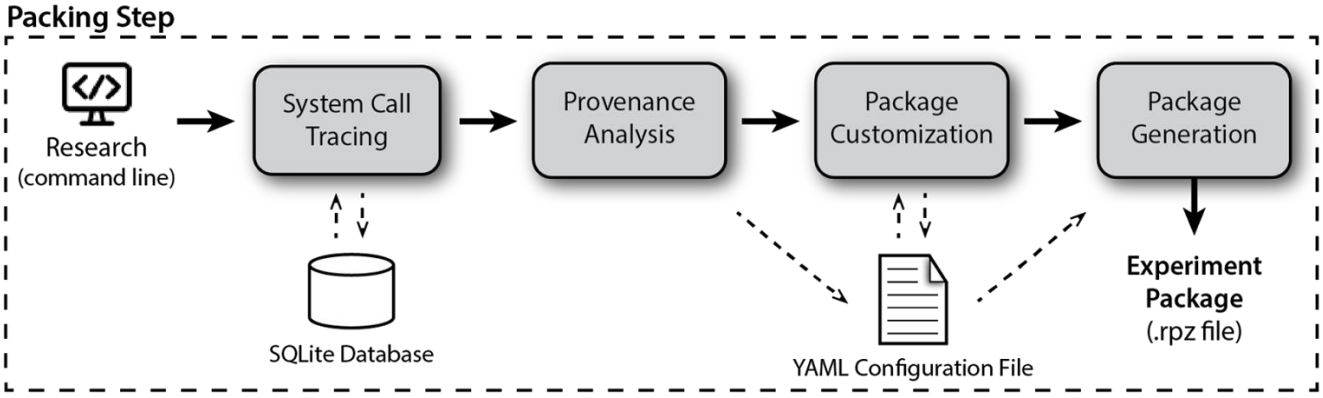

Figure 1. Packing step on ReproZip.

2.2 Unpacking

ReproUnzip allows users to unpack a .rpz package on any operating system, using the command line or the GUI (Figure 3). Different unpackers are provided to reproduce the corresponding work. ReproUnzip was designed to allow anyone to create new plugins to unpack .rpz files, which will support every package created in the past. This model allows both the core maintainers of ReproUnzip and its users/contributors to plan for current unpackers (Vagrant and Docker) to become deprecated, or for future unpackers to be developed. This way, users are not locked into specific technologies, and so can use their ReproZip packages to the fullest.

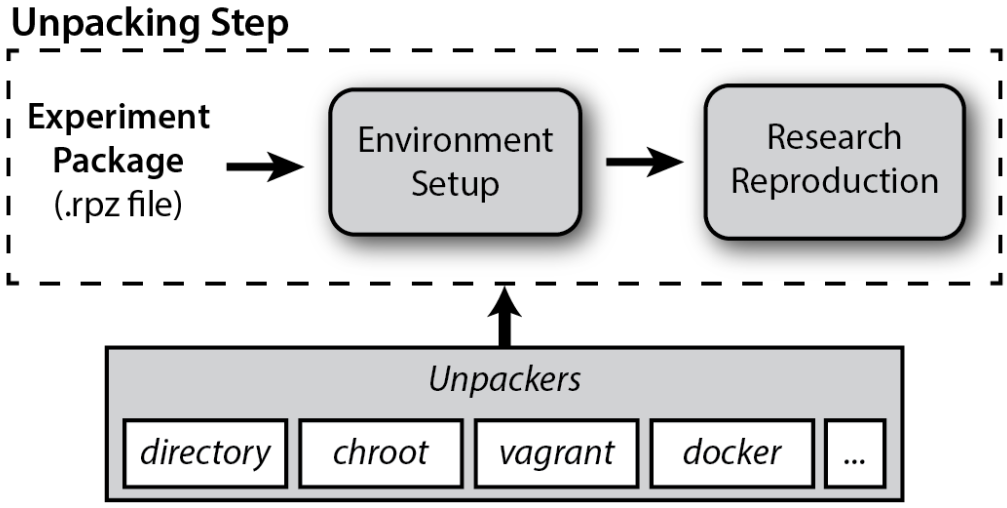

Figure 2. Unpacking step on ReproZip.

Unpacking works in two steps. The first is setting up the environment, which consists of extracting information from the .rpz, and can be accomplished by using the `reprounzip setup <package-name>`

command, or by double-clicking the .rpz file to use with the GUI. This step varies based on the unpacker chosen. For reproducing research across different operating systems, users can choose between Vagrant and Docker unpackers. For reproducing the research on Linux, users can additionally use the `directory` or the `chroot` unpackers. For the latter, setting up the .rpz file means copying its contents into a single directory. For the former, a Docker container or Vagrant virtual machine is initialized.

To re-execute the research, users can either use the `reprozip run <path-to-unpacked-work>` on the command line or click the 'Run Experiment' button in the GUI. If the user is unpacking using Vagrant or Docker, ReproUnzip will automatically create and start the virtual machine or Docker container. The secondary user needs no knowledge of these programs – they simply need to use the setup and run commands, and ReproUnzip will take care of the rest. The `directory` and `chroot` options install all the required dependencies in the secondary users' native system, and reruns the research inside a directory. These two unpackers are not recommended because they are not as reliable as Vagrant or Docker, which provide an isolated execution environment.

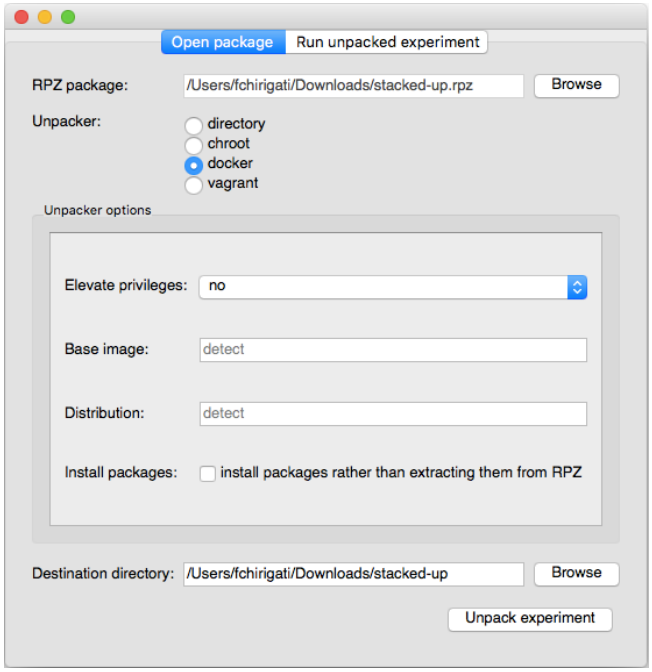

Figure 3. The graphical user interface for unpacking research.

2.2.1 Data Management

ReproZip identifies the role of files in the research (input and/or output) so users can take advantage of this in the unpacking step. After setting up and reproducing the research, users can download the resulting output files for their own inspection or reuse. Users may also upload their own input files – this is useful when a researcher wants to use the same method on data they have collected independently of the original user.

ReproZip also allows users to visualize and edit their workflow through an integration with VisTrails, a scientific workflow system (Freire and Silva, 2012). By working in VisTrails, users can examine, modify, and play around with their own workflow, which is useful for evaluating the efficiency of methodologies used in the original work. This is made easier by the fact that users can re-execute unpacked work directly inside of VisTrails.

3 Current Use Cases

ReproZip's predominant use cases are for archiving scholarly works, publishing and sharing reproducible research, and reviewing results in publications for academic journals. Within scholarly works, ReproZip is used as a means to reproduce work, but also as a useful tool for reproducibility in many different fields, most recently data journalism, geoscience, and neuroscience, for example: Boss and Broussard, 2016; Knoth and Nüst, 2017; Ghosh et al., 2017.

Since version 1.0 was released, ReproZip has had 271 users, and a subset of those users have allowed us to publicize their work on GitHub[11] and through a website, ReproZip-Examples[12]. Currently, there are four examples from digital humanities and social sciences, specifically from web publishing (including capturing client-server architecture and databases), history, and data journalism (one example has a playlist of demo videos for this example[13] on YouTube).

Additional examples include use cases in STEM, including a neuroscience example, which also demonstrates the way ReproZip can be used in tandem with the popular tool Jupyter Notebooks (there are also demo videos[14] on YouTube for this example); one interactive graphical application example, and many data science examples, specifically in machine learning, statistics (including simulations), and urban studies.

These examples range not only in discipline, but also in technologies used. We have examples using Java, R, Python, the Django web framework, PostgreSQL databases, and more. The largest ReproZip package in this gallery of examples is only 84 Mb, and the smallest is 19 Mb.

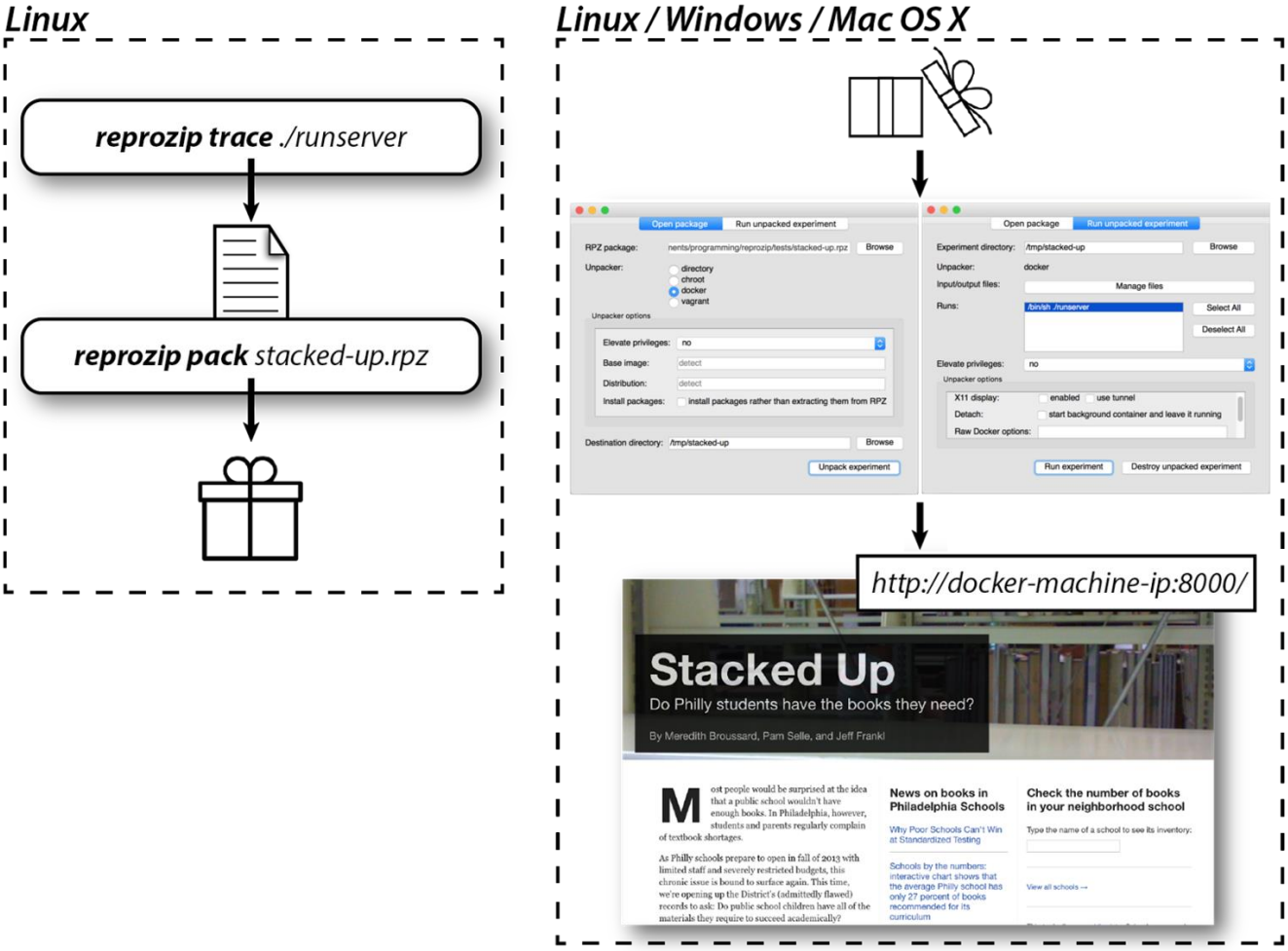

Figure 4. Steps in using ReproZip to package StackedUp, a web application.

ReproZip was also recently integrated into the National Institute of Standards and Technology's CoRR (Cloud of Reproducible Records) web platform, designed as a gateway for simulation management tools to capture software executions. CoRR also allows users to store and view metadata associated with simulation records (Faical Yannick Palingwende Congo, 2015). The testing repository is on GitHub at CoRR-ReproZip[15].

ReproZip has also been integrated into publication workflows for scholarly journals and conference proceedings. The journal *Information Systems* recommends using ReproZip for its reproducibility section, which evaluates the reproducibility of submitted papers. ReproZip is also recommended by the Association of Computing Machinery (ACM) Artifact Evaluation Process guidelines, and in the ACM Special Interest Group on Management of Data (SIGMOD) reproducibility review for conference materials. There is continued work on expanding this to fields outside computing, given that ReproZip has a demonstrated usefulness in digital humanities, physical sciences, and social sciences.

4 ReproZip in Librarianship

Librarians involved with research, whether their own or others', can utilize ReproZip to help make it reproducible. Disciplines within librarianship can also use ReproZip to great advantage, such as software preservation, repository management, and generally digital libraries and archives infrastructure.

4.1 Digital Libraries

Using ReproZip, digital preservation work can be streamlined; users, data managers, and digital archivists/librarians can capture works, research, and most digital content at the environmental level to create a preservation-ready object. This file is a preservation-ready object for a number of reasons, but the most salient is the flexibility in unpacking technologies. One .rpz file can be rerun using many different unpackers in ReproUnzip. Due to ReproZip's plugin model, unpackers can be added and removed as systems become more or less popular or usable. If Vagrant no longer exists in ten years, then an unpacker for another virtual machine system can be written and the ReproZip packages remain usable.

However, if in the future there are no containers or virtual machines, then the archivist/librarian can still use the robust technical and administrative metadata in the configuration file that ReproZip provides. This config.yml is machine-readable, and lists every single piece of information about the computational environment and dependencies necessary to recreate the environment. This can be extracted during the 'setup' phase of unpacking.

As digital libraries and archives have begun collecting and preserving software alongside other materials they collect, this has presented many challenges in terms of long-term preservation, which ReproZip could help alleviate. There is a lot of work being done in this area by not only institutional libraries, but also in consortial groups such as the Software Preservation Network[16], who seek to "save software together," through community engagement, infrastructure support, and knowledge generation.

However, software preservation has a unique problem in that the boundaries for the preservation-ready objects are not clearly defined. This is due in large part to the problem of dependency hell described earlier. Some considerations include archiving simply the source code or executable, or the whole environment on which it runs (McDonough and Olendorf, 2011). In describing use cases for software preservation, Rios et. al. (2017) call out first the "capture [of] the research transparency, reproducibility, and reuse scenarios of software preservation. The goals are to enable other researchers to examine and understand the preserved software and enable its re-execution and reuse."

ReproZip, in capturing computing environments in which research takes place, could be used to preserve software down to the operating system on which it runs. Additionally, ReproZip allows for users not only to reproduce and modify the packages, but also to reuse them for their own purposes. Making collections accessible and usable are key components of libraries and archives, and ReproZip facilitates this goal for research, applications, and software. Users could leverage ReproZip at library terminals, through obtaining a .rpz file via the catalog, and simply reproduce the research for their own learning and enrichment, or build on the contents for their own work.

ReproZip allows digital librarians and archivists to capture scholarly works, software, and more at the computing environmental level, and have a preservation-ready object at the end of it: the .rpz.

4.2 Repository Management

Open and accessible infrastructure is key to advancing openness in research. Libraries have answered these challenges by either creating data repositories alongside their institutional repositories, or integrating deposit of data and other research output into their institutional repositories.

As more repositories look to include research materials, building in tools to enable reproducibility is key. ReproZip's implementations in CoRR is one example of how it can be integrated into a repository environment. CoRR uses ReproZip as an application that users registered to the platform could use to host and store their ReproZip 'records,' which are metadata records and .rpz files that are accessible, shareable, and published within CoRR. Users can add to the 'reprozip trace' command the configuration to automatically push .rpz files and metadata to the platform.

This represents a way to integrate ReproZip into the ingest process of repositories. While the ingest process in a library context would look different (a standard ingest process and descriptive metadata added), ReproZip provides extensive metadata extraction options available from the configuration files that would help in automating this workflow. From the package, which contains extremely detailed technical and administrative metadata, can be exported as a json file, which allows for extensible models of metadata – such as crosswalking to the Resource Description Framework (RDF) or Dublin Core (Digital Public Library of America, 2017). Aggregating metadata from diverse digital objects requires this level of extensibility, which ReproZip automatically supplies within its own options for reading and extracting metadata. Allowing ReproZip packages to be a citable object within repositories would also go a long way in creating cultural shifts towards rewarding reproducible practices.

As repositories begin to build curation workflows around reproducibility, ReproZip can be built in to expand the creation of reproducible works. Additionally, as these services grow and expand models of ingest workflows to include a variety of digital objects, repository managers can leverage the metadata created within the ReproZip packages.

4.3 Academic Libraries

With the rise of data services departments (also called data management, research data services, or statistical services) in academic libraries, their involvement with researchers' data management has grown exponentially (Crowe and Crumpton, 2016). Currently, most support in academic libraries for reproducibility comes out of data management services. However, data management is a means towards reproducibility, and libraries must begin to develop holistic support for reproducible research practices, extending beyond simply data management.

4.3.1 Liaison Librarians/Subject Specalists

Liaison librarians play a central role in identifying and reacting to user needs. Liaison work includes assessment of services, collecting materials to satisfy patron needs, and locating useful resources to enhance scholarship. But perhaps most importantly, liaison work involves a focused and dedicated relationship with a subset of library users – the work is reliant on these relationships which requires two-way communication (Silver, 2015). In this way, many library workers rely on liaison librarians to

effectively communicate new initiatives, ideals, and services offered. Liaison librarians are the bridge between students and faculty and the library. This offers an excellent opportunity to encourage and disseminate information on reproducible practices, particularly with ReproZip.

Furthermore, liaison librarians also are research collaborators. A recent case study that exemplifies this idea is a paper co-authored by Katherine Boss, the liaison librarian at New York University for journalism and media studies, and Meredith Broussard, a faculty member in the Arthur L. Carter Journalism Institute at New York University. Their paper, 'Challenges facing the preservation of born-digital news applications,' delves into how two different domain-specific sets of expertise work in tandem to solve a set of problems evident in both fields – digital preservation and archiving of news apps (such as Dollars for Docs[17]). On the library side, there's the challenge of digital preservation and archiving software, but also conceptually of how to collect, disseminate, and make these news apps accessible (Boss and Broussard, 2016). On the journalism side, there is a serious concern that as funding falls away, so too will these important digital resources that organizations such as ProPublica provide. Working together, these research questions are advanced in both fields, and accessibility to vetted news applications is advanced. Boss and Broussard are also exploring ReproZip as a solution to news app preservation (see the StackedUp[18] example on Reprozip-Examples) as a direct result of collaboration with the ReproZip development team (one member is a librarian) which resulted in Boss communicating this tool to her department.

Generalizing from Boss and Broussard's partnership, there is space and motivation for liaison specialists to work not only with other library partners, but also researchers. This can include using ReproZip for mutual benefit -- the library gains an excellent way to preserve and build collections of diverse, preservation-ready research outputs, and users easily integrate reproducible practices into their workflows.

4.3.2 Data Services

As libraries add support for data services, which typically include data management training and software support for quantitative, geographic information systems (GIS), and qualitative research, there should be support for reproducible research practices. As demonstrated, ReproZip is an extremely user-friendly tool. In two steps, users can have reproducible research, and in another two steps, a secondary user can unpack the .rpz file and re-execute their work. This presents a low barrier not only to the research community, but also to those in data services departments involved with instruction and consultation. Support for reproducibility using ReproZip in research data services would draw in a lot of opportunities to cross-campus collaboration, as well as position data services departments and the library as a center for modern research practices.

By adding ReproZip to the curriculum of data services department workshops, and to the list of supported software, data services teams can become a center on campus for assisting the reproducibility of their user's works.

5 Future Development Work

ReproZip has been evolving since 2012 and has always been open source. Development work on the tool has been initiated through assessing user needs, ascertained through one-on-one interactions, workshops, GitHub issues, and/or the ReproZip users mailing list. The hope is to continue this work as a core team of maintainers, but also solicit contributions from the community of users.

5.1 macOS Packing

ReproZip currently only allows for packing on Linux OS. Across domains, this represents a minority of users, who tend to prefer macOS and Windows.

The team has chosen to begin with macOS, as packing on this operating system is technically possible with a low barrier, based on our previous work. Because macOS and Linux are both Unix-based operating systems, ReproZip could potentially use *dtrace* to capture dependencies on macOS in place of the *ptrace* that is available for Linux systems. The limitations of reproducing macOS environments are significant in terms of licensing; as set out in the macOS user agreement, users may only run macOS on Apple machines. Therefore, unpacking would only be 'allowed' on another macOS machine, which would be very limiting to the users.

5.2 ReproZip-Jupyter

Jupyter notebooks (also know as IPython Notebooks) are an interactive computational environment where users can combine executable code, text, math, plots, and rich media (Thomas et al., 2016). This tool is increasingly at the center of research initiatives across many domains of scholarship. However, Jupyter notebooks are not inherently reproducible: they do not capture provenance information, dependencies, or the native environment. To add this missing element, the developers created the Jupyter plugins for ReproZip, which allow users to pack a notebook from the Jupyter interface, and also to choose to start a notebook server and rerun when unpacking. Users can see the source[19] and watch a demo video[20] demonstrating packing a notebook.

5.3 Workflow Visualizations and Graphs

Currently, ReproZip allows users to visualize the provenance and execution of the contents of a .rpz package using GraphViz, an open source graph visualization software, used to represent structural information as diagrams of abstract graphs and networks (Gansner and North, 2000). This allows users to graphically represent the flow of their work, both for internal exploratory purposes and for tracking provenance.

There is room for improvement on this simple visualization by using the D3 Javascript library[21] to make the graph more appealing aesthetically, and to allow the user to interactively explore the provenance data which is often too much to be shown in full on a single graph. It will also be integrated into the GUI of ReproUnzip, so users could interact with the graph and choose to save snapshots as image files for sharing and publication from there. This would help users in communicating the execution and provenance of their research or application. A prototype of this has been created but not yet integrated into ReproZip.

5.4 MPI/HPC

Lastly, there is demand to add more support for researchers who use High Performance Computing (HPC) in their research. ReproZip runs on Linux and therefore on most HPC clusters. There has been success reproducing experiments spanning multiple machines with shared filesystems using the Message Passing Interface (MPI) framework, merging the traces into a working .rpz file. In the future, the team aims to add functionality that is aware of the common scheduling tools and setups of HPC facilities to automatically run ReproZip to trace all the processes without the user having to adapt his job submission routine. For unpacking, correctly rerunning an unpacked experiment through ReproUnzip for reproduction on a new cluster also requires extra functions. It would also be prudent to develop unpacker plugins specifically for HPC environment, where virtual machines are impractical and Docker is generally not yet available.

6 Conclusion

The library community can leverage ReproZip in instruction, consultation, repository services, digital archiving, and in their own research. It is extensible enough to be used for reproducibility across research domains as well as across library services. It is simple enough to use that it can be easily adopted to ensure reproducibility of scholarship, even if reproducibility is an afterthought. The user does not need to start their project with reproducibility in mind – but can, in the end, use ReproZip to

create a compendium of their work that's easily shareable, citable (if deposited in an institutional repository), and usable by themselves and the community at large. Ultimately, this improves research by lowering the barriers to reproducibility, and also improves the way in which the LIS community can preserve and make discoverable these research compendia.

7 Acknowledgements

We would like to acknowledge Dr. Juliana Freire, the Principal Investigator of the ReproZip project, for her support in continuing to build ReproZip. We'd also like to thank Dr. Nicholas Wolf, Beth Daniel Lindsay, Jonathan Petters, and Declan Fleming for comments on this paper. Lastly, we would like to acknowledge the support from the Gordon and Betty Moore Foundation as well as the Alfred P. Sloan Foundation. The Moore-Sloan Data Science Environment was vital to the development of ReproZip.

References

Boss, Katherine, and Meredith Broussard. "Challenges Facing the Preservation of Born-Digital News Applications," 2016.
http://blogs.sub.uni-hamburg.de/ifla-newsmedia/wp-content/uploads/2016/04/Boss-Broussard-Challenges-Facing-the-Preservation-of-Born-digital-News-Applications.pdf.

Chirigati, Fernando, Rémi Rampin, Dennis Shasha, and Juliana Freire. "ReproZip: Computational Reproducibility With Ease," 2085–88. ACM Press, 2016. doi:10.1145/2882903.2899401.

Crowe, Kathryn, and Michael Crumpton. "Defining the Libraries' Role in Research: A Needs Assessment; A Case Study," 2016. https://libres.uncg.edu/ir/listing.aspx?id=19091.

Davidson, Susan B., and Juliana Freire. "Provenance and Scientific Workflows: Challenges and Opportunities." In Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data, 1345–1350. SIGMOD '08. New York, NY, USA: ACM, 2008. doi:10.1145/1376616.1376772.

Digital Public Library of America. "An Introduction to the DPLA Metadata Model," January 6, 2017. https://dp.la/info/wp-content/uploads/2015/03/Intro_to_DPLA_metadata_model.pdf.

"Docker." Docker, 2017. https://www.docker.com/.

Faical Yannick Palingwende Congo. "Building a Cloud Service for Reproducible Simulation Management." In Proceedings of the 14th Python in Science Conference, edited by Kathryn Huff and James Bergstra, 195–201, 2015.
https://conference.scipy.org/proceedings/scipy2015/pdfs/yannick_congo.pdf.

Feller, M. "Configuration Management." IEEE Transactions on Engineering Management EM-16, no. 2 (February 7, 2013): 64–66. doi:10.1109/TEM.1969.6447051.

Fernando Rios, Nicole Contaxis, Bridget Almas, Paula Jabloner, and Heidi Kelly. "Exploring Curation-Ready Software: Use Cases." The Software Preservation Network, April 14, 2017.
http://www.softwarepreservationnetwork.org/exploring-curation-ready-software-use-cases/.

Freire, J., and C. T. Silva. "Making Computations and Publications Reproducible with VisTrails." Computing in Science Engineering 14, no. 4 (July 2012): 18–25. doi:10.1109/MCSE.2012.76.

Gansner, Emden R., and Stephen C. North. “An Open Graph Visualization System and Its Applications to Software Engineering.” Software: Practice and Experience 30, no. 11 (September 2000): 1203–33. doi:10.1002/1097-024X(200009)30:11<1203::AID-SPE338>3.0.CO;2-N.

Ghosh, Satrajit S., Jean-Baptiste Poline, David B. Keator, Yaroslav O. Halchenko, Adam G. Thomas, Daniel A. Kessler, and David N. Kennedy. “A Very Simple, Re-Executable Neuroimaging Publication.” F1000Research 6 (February 10, 2017): 124. doi:10.12688/f1000research.10783.1.

Goodman, Steven N., Daniele Fanelli, and John P. A. Ioannidis. “What Does Research Reproducibility Mean?” Science Translational Medicine 8, no. 341 (June 1, 2016): 341ps12-341ps12. doi:10.1126/scitranslmed.aaf5027.

Gronenschild, Ed H. B. M., Petra Habets, Heidi I. L. Jacobs, Ron Mengelers, Nico Rozendaal, Jim van Os, and Machteld Marcelis. “The Effects of FreeSurfer Version, Workstation Type, and Macintosh Operating System Version on Anatomical Volume and Cortical Thickness Measurements.” Edited by Satoru Hayasaka. PLoS ONE 7, no. 6 (June 1, 2012): e38234. doi:10.1371/journal.pone.0038234.

Guo, Philip. “CDE: A Tool for Creating Portable Experimental Software Packages.” Computing in Science & Engineering 14, no. 4 (July 2012): 32–35. doi:10.1109/MCSE.2012.36.

Knoth, Christian, and Daniel Nüst. “Reproducibility and Practical Adoption of GEOBIA with Open-Source Software in Docker Containers.” Remote Sensing 9, no. 3 (March 18, 2017): 290. doi:10.3390/rs9030290.

Marwick, Ben. “How Computers Broke Science – and What We Can Do to Fix It.” The Conversation, November 9, 2015. http://theconversation.com/how-computers-broke-science-and-what-we-can-do-to-fix-it-49938.

McDonough, Jerome, and Robert Olendorf. “Saving Second Life: Issues in Archiving a Complex, Multi-User Virtual World.” International Journal of Digital Curation 6, no. 2 (October 7, 2011): 89–108. doi:10.2218/ijdc.v6i2.192.

Open Science Collaboration. “Estimating the Reproducibility of Psychological Science.” Science 349, no. 6251 (August 28, 2015): aac4716-aac4716. doi:10.1126/science.aac4716.

Pham, Quan, Tanu Malik, and Ian Foster. “Using Provenance for Repeatability.” In Proceedings of the 5th USENIX Conference on Theory and Practice of Provenance, 2–2. TaPP’13. Berkeley, CA, USA: USENIX Association, 2013. http://dl.acm.org/citation.cfm?id=2482613.2482615.

Silver, Isabel D. “For Your Enrichment: Outreach Activities for Librarian Liaisons.” Reference & User Services Quarterly 54, no. 2 (January 26, 2015): 8–14.

Smith, J.E., and Ravi Nair. “The Architecture of Virtual Machines.” Computer 38, no. 5 (May 2005): 32–38. doi:10.1109/MC.2005.173.

Stodden, Victoria C. “The Scientific Method in Practice: Reproducibility in the Computational Sciences,” 2010. https://academiccommons.columbia.edu/catalog/ac:140117.

Stodden, Victoria, Marcia McNutt, David H. Bailey, Ewa Deelman, Yolanda Gil, Brooks Hanson, Michael A. Heroux, John P. A. Ioannidis, and Michela Taufer. “Enhancing Reproducibility for

Computational Methods." Science 354, no. 6317 (December 9, 2016): 1240–41. doi:10.1126/science.aah6168.

Thomas, Kluyver, Ragan-Kelley Benjamin, Pérez Fernando, Granger Brian, Bussonnier Matthias, Frederic Jonathan, Kelley Kyle, et al. "Jupyter Notebooks-- a Publishing Format for Reproducible Computational Workflows." Stand Alone, 2016, 87–90. doi:10.3233/978-1-61499-649-1-87.

Endnotes

[1]Vicky Steeves is Librarian for Research Data Management and Reproducibility, Division of Libraries & Center for Data Science, New York University, vicky.steeves@nyu.edu

[2]Rémi Rampin is PhD Candidate, Tandon School of Engineering, New York University, remi.rampin@nyu.edu

[3]Fernando Chirigati is Research Engineer, Center for Data Science, New York University, fchirigati@nyu.edu

[4]The authors presented this paper at the 2017 IASSIST conference. The presentation can be found here: https://vickysteeves.gitlab.io/2017-IASSIST-ReproZip/#/

[5]https://www.reprozip.org/

[6]https://examples.reprozip.org/

[7]https://www.vistrails.org/

[8]https://kepler-project.org/

[9]https://ipython.org/notebook.html

[10]#5 Future Development Work

[11]https://github.com/ViDA-NYU/reprozip-examples

[12]https://examples.reprozip.org/

[13]https://www.youtube.com/watch?v=SoE2nEJWylw&list=PLjgZ3v4gFxpXdPRBaFTh42w3HRMmX2WfD

[14]https://www.youtube.com/watch?v=gs4s2JH8Yw4&list=PLjgZ3v4gFxpW_lesrw1ZE2Ql1yXDsgzki

[15]https://github.com/usnistgov/corr-reprozip

[16]http://www.softwarepreservationnetwork.org/

[17]https://projects.propublica.org/docdollars/

[18]https://github.com/ViDA-NYU/reprozip-examples/tree/master/stacked-up

[19]https://github.com/ViDA-NYU/reprozip/tree/ipython

[20]https://www.youtube.com/watch?v=Y8YmGVYHhS8

[21]https://d3js.org