Guaranteeing reproducible software environments by combining Homebrew, continuous integration, and container images

Problem statement

Reproducibility is a critical component of the scientific process, as it permits future research to confidently build on past results. When reporting results, scientists are generally expected to document the steps taken in sufficient detail to permit others to replicate their findings. However, recent work documenting the "reproducibility crisis" has raised doubts about the reliability of published results (Baker 2016). As a partial solution, the reproducible software movement has begun to develop tools, methods, and best practices around computational analyses (Sandve et al. 2013), leveraging the inherent determinism in computer programs to increase the confidence in generated results, by ensuring that others can obtain the same results with the same data input and analyses.

Despite the promise of reproducible software, results from computational analyses in the scientific literature are often not reproducible, due to differences in how software is configured, packaged, and installed among users (Garijo et al. 2013; Collberg, Proebsting, and Warren 2015). This undesirable heterogeneity in the computing environment has increased the popularity of tools such as `packrat` and `virtualenv`, which keep track of R or Python packages used in a particular project and make reproducing the same environment straightforward. However, similar approaches do not currently exist for native, compiled software tools such as RAxML, SAMtools, or MAFFT.

Here I propose to extend a popular package manager, Homebrew, in two areas that will improve software reproducibility.

1. Ensure that all users of Homebrew can build their own software using the Homebrew build system, even if that software is too esoteric to be officially supported by the Homebrew team.
2. Create a way to track, save, and restore Homebrew-provided software in the researcher's computing environment.

These two steps will allow anything packaged using the battle-tested Homebrew build system, whether supported officially or by a third-party, to easily have a pre-built binary "bottles" associated with them; and will allow researchers to publish the software environment that generated their original results, so that future replication attempts can restore the exact same environment. Homebrew is an excellent system to advance research reproducibility, as it is easy for new users to contribute building software by putting together a simple "formula", a domain-specific language written in Ruby.

Proposed activities & outcomes

Aim 1. Ensure researchers can easily deploy continuous-integration (CI) infrastructure for testing and building binaries.

Homebrew currently runs on all major desktop computing environments: macOS, Linux (via Linuxbrew), and Windows (via Bash on Ubuntu on Windows with Linuxbrew). However, due to time and resource constraints, the official Homebrew organization does not aim to provide either binary bottles or support for source code builds for every single piece of software.[1] Therefore, many software packages are hosted

[1] https://docs.brew.sh/Acceptable-Formulae.html, "We will reject formulae that seem too obscure, partly because they won't get maintained and partly because we have to draw the line somewhere."

in third-party repositories ("taps"), where they do not benefit from the Homebrew continuous integration infrastructure for testing and building binary bottles, and thus the usability and reliability of software maintained this way is much worse compared to officially-supported software.

This activity will improve the documentation and tooling for building binary bottles, to ensure that any tap maintainer can have access to free test and build infrastructure similar to official Homebrew. The template to create a new tap includes a configuration file for Travis, a popular continuous integration service; a user merely has to sign up for Travis and enable tests for that repository. However, this procedure is poorly documented, hurting discoverability for new tap maintainers, and does not currently integrate with services such as Bintray, which Homebrew uses to distribute its pre-built binary bottles.

Proposed outcomes. Creation and improvement of documentation and tools for setting up continuous integration infrastructure for Travis, Circle, Amazon Web Services, and self-hosted solutions. The setup of infrastructure for at least two third-party repositories ("taps"), one of which is not maintained by the Co-PI, that successfully use CI for testing and building binary bottles.

Impact on research community. This would permit any software writer, regardless of how large or small their audience size, to leverage the well-tested Homebrew build infrastructure to consistently compile and distribute their own software. It would also permit any user of such software to reliably receive pre-built binary bottles, rather than the difficulty setting up their own compilation environment, increasing the accessibility of these research tools.

Aim 2. Tracking, saving, and restoring snapshots of the Homebrew-provided computing environment using Docker and other tools

Major differences how software is configured, packaged, and installed among users, collectively termed the "computing environment", has raised major roadblocks to the goal of reproducible software (Garijo et al. 2013). One emerging technology, containerization, allows users to run applications and their dependencies within isolated "containers". A "container image" generated using a tool such as Docker should have the exact same software installed on it regardless of where it is deployed. Containers are therefore one potential solution to the problem of heterogenous computing environments where scientific analyses are performed.

Proposed outcomes. A new Homebrew command, `brew docker-create`, that will use Docker to create a clean Homebrew environment inside a Docker container where software and data can be installed, or optionally copied from an existing installation. Another command, `brew docker-deploy`, that will use Docker to deploy and execute a pre-existing container image. Similar support for Vagrant, a tool to manage virtual machines, for researchers that desire more control over isolation and consistency. Documentation and tutorials, with extensive examples, of how to use this functionality to ensure reproducible software environments.

Impact on research community. Researchers would be able to use the new `brew docker-create` command to package all of their software, scripts, and data files into a single container image for archiving in a data repository like Dryad or figshare. Results in a published manuscript could readily be checked by a community member by downloading the container image and running `brew docker-deploy` and verifying the outputs.

Tentative timeline

Task	Winter	Spring	Summer	Fall
Integrate CI (Travis, Circle, AWS, self-hosted)	x	x		
Integrate storage solutions (Bintray, AWS)	x	x		
Integrate environment reproduction with Docker		x	x	
Integrate environment reproduction with Vagrant		x	x	x
Upstream code to Homebrew and Linuxbrew projects	x	x	x	x
Create and improve documentation and tutorials	x	x	x	x
Present at conference and write manuscript				x

Personnel

Jonathan Chang (UCLA) will serve as the rOpenSci Fellow and perform all of the tasks outlined in this proposal. He is a maintainer and user of Homebrew-science and Linuxbrew. **Michael Alfaro** (UCLA) will supervise the project as the PI and postdoctoral mentor, and is a current user of Homebrew. **Mike McQuaid** (GitHub) is the lead maintainer for Homebrew and would provide guidance and direction for contributing to the core Homebrew codebase and review any code submitted to Homebrew. **Shaun Jackman** (University of British Columbia) is a maintainer of Homebrew-science and Linuxbrew and would review code submitted to Linuxbrew. **All Homebrew maintainers, users, and community members** will also be welcome to suggest improvements or conduct reviews on an *ad hoc* basis, as all development will be on GitHub.

Budget

Funds are requested to support Co-PI Jonathan Chang for one year as a postdoctoral researcher. The UCLA minimum yearly stipend for a postdoc is $48,216, with an additional $15,477 for fringe at the rate of 32.1%, for a total of $63,693. The Co-PI will commit 12 months at 100% effort for this project. Travel funds of $1,040 are requested for a one-person/four-day trip to attend an academic conference to present and promote the work products from this project. Projected costs include: airfare $400, lodging $600, ground transportation $50; registration $390. These are anticipated based on Google Maps and historical data. The total requested budget is therefore $64,733.

References

Baker, Monya. 2016. "1,500 Scientists Lift the Lid on Reproducibility." *Nature* 533 (May): 452–54. doi:10.1038/533452a.

Collberg, Christian, Todd Proebsting, and Alex M Warren. 2015. "Repeatability and Benefaction in Computer Systems Research: A Study and a Modest Proposal." TR 14-04. University of Arizona. http://reproducibility.cs.arizona.edu/v2/RepeatabilityTR.pdf.

Garijo, Daniel, Sarah Kinnings, Li Xie, Lei Xie, Yinliang Zhang, Philip E. Bourne, and Yolanda Gil. 2013. "Quantifying Reproducibility in Computational Biology: The Case of the Tuberculosis Drugome." Edited by Christos A. Ouzounis. *PLoS ONE* 8 (11). Public Library of Science (PLoS): e80278. doi:10.1371/journal.pone.0080278.

Sandve, Geir Kjetil, Anton Nekrutenko, James Taylor, and Eivind Hovig. 2013. "Ten Simple Rules for Reproducible Computational Research." Edited by Philip E. Bourne. *PLoS Computational Biology* 9 (10): e1003285. doi:10.1371/journal.pcbi.1003285.