ILLUSTRATION BY THE PROJECT TWINS

A TRICK TO SIMPLIFY SCIENTIFIC COMPUTING

Computational environments and the tools to manage them can help researchers to deliver code that is reproducible, documented and shareable. **By Jeffrey M. Perkel**

Research software is like the tower-building game Jenga – tools atop tools atop tools. When developers tweak their individual pieces, this can change the function of the software that depends on them, potentially altering results – or causing the software to fail.

Version 3.6.0 of the R programming language, for instance, introduced a replacement algorithm for generating random numbers. This and the older algorithm both work, but not in the same way. "If you ran the same code with an older version of R and a newer version of R and it was using any function that needed to generate a random number, you would end up getting different results," says Tiffany Timbers, a data scientist at the University of British Columbia in Vancouver, Canada.

Among other things, that kind of variability can complicate collaboration (see 'Environmental testing'). In 2020, Mine Çetinkaya-Rundel was working with another author on a statistics textbook, using R and a formatting language called R Markdown to calculate numbers, create figures and format the final document. "We wanted to make sure that we were using the same versions," says Çetinkaya-Rundel, a statistician at Duke University in Durham, North Carolina, "and also that when we re-render the book, we're rendering it with a given version of the packages." If not, the two authors could have generated slightly different manuscripts.

To address that challenge, they turned to the R package renv, one of a small group of tools that help developers and researchers to manage their computational environments; other options include venv and virtualenv for Python, and conda, a language-agnostic tool. Most are command-line utilities, although renv is tightly integrated with the RStudio Desktop graphical programming environment. All can help researchers to ensure that their code is reproducible, reusable, documented and shareable.

Sleight of hand

C. Titus Brown, a bioinformatician at the University of California, Davis, has 187 conda environments on his laptop. Most are one-offs, used to test new tools or to illustrate a point during lectures. His day-to-day work mostly takes place in a development environment that includes a specific version of Python and other programming tools.

Some tasks, however, require a change of computational scenery. For instance, Brown writes blog posts in Markdown, which he renders into HTML, the standard markup language for web pages. But the code that performs that step doesn't work well with newer versions of a crucial software library, and older versions conflict with his development tools. To isolate

the problem, Brown created a separate environment. "I just fixed the version to something really old that still works, and I run [the rendering software] there," he says.

A conda environment is a computational sleight-of-hand, says Johannes Köster, a computer scientist at the University of Duisburg-Essen in Germany, who founded a bioinformatics-focused software repository called Bioconda. "Basically, it's just modifying your system path — the place where your system searches for executable [applications]." You might have multiple versions of a tool installed, but when conda activates a particular environment, your computer can only see the one you want.

Computational environments offer several benefits, says Timbers. One is reproducibility — the ability to analyse the same data with the same software on the same computing infrastructure to get the same results.

"It can be very frustrating, tracing down the differences between outputs across different computers," says Ben Marwick, an archaeologist at the University of Washington in Seattle. Some research projects take years to complete, he notes. And although Marwick prefers the newest libraries, his colleagues don't always upgrade at the same pace. Renv ensures that he and his collaborators always run their project codes in the same way. The resulting environment-description file can be version-controlled and shared on GitHub. Collaborators can recreate the environment using the command renv::restore()

Conda is a command-line tool that both creates environments and installs software into them. To create a new environment called my_env pinned to a specific version of Python, for instance, use conda create --name my_env python=3.9

Both R and conda allow users to install their own tools rather than having to ask system administrators to do it for them. "You don't need root privileges," says Rob Patro, a computational biologist at the University of Maryland in College Park. This is a useful feature when working on shared computing resources.

Environment managers also make software installation easier. Scientific software is often released as source code, which might need to be compiled, configured and installed in a specific location. It might have a network of dependencies, written in multiple programming languages, that must be installed in a particular order. Sometimes, says bioinformatician Fredrik Boulund at the Karolinska Institute in Stockholm, the process can be beyond users' skills. "That completely changed when solutions like conda entered the scene," he says. "Installing a complex set of dependencies is simply reduced to asking conda to create an environment according to an environment specification file."

For the Galaxy project — an open-source framework for reproducible data analysis — those features were a key reason for choosing conda as the project's software installation manager. Bioinformatician Björn Grüning, who runs the European Galaxy server at the University of Freiburg in Germany, says that the Galaxy community started searching for a cohesive tool-installation strategy in around 2015 because its existing, manual approach was unsustainable. "Conda ticked all our requirement boxes," Grüning says. It doesn't need root privileges; it is programming-language agnostic; and it uses human-readable package recipes that are easy to understand and maintain. Today, there are more than 9,000 bioinformatics tools available to Galaxy users through the Bioconda channel.

Early starts

Perhaps the biggest benefit to environments, however, is isolation: environments enable researchers to explore new or updated tools while knowing that their code will still run.

Elana Fertig, a statistician at Johns Hopkins University in Baltimore, Maryland, describes herself as "lax" when it comes to environments: "For me, everything goes in one environment." But larger environments are harder to use, because the environment manager has to resolve a larger network of dependencies to install new tools. (Conda is notorious for poor performance with large environments, but a drop-in resolver called mamba accelerates the process.) Instead, Fertig suggests that her students use one environment per project.

Indeed, most researchers contacted for this article recommend creating environments to accommodate specific workflows or projects — and to do so early on. "Start your project with a package-management solution in mind," says Joshua Shapiro, senior data scientist at the Childhood Cancer Data Lab for Alex's Lemonade Stand Foundation, based in Wynnewood, Pennsylvania. "It has the potential to save a lot of headaches down the line."

Tommy Tang, director of computational biology at Immunitas Therapeutics, a biotechnology company in Waltham, Massachusetts, uses dedicated environments for different computational tasks — processing data from RNA sequencing or working in Google Cloud, for instance.

Users of the Snakemake and Nextflow computational workflow managers can even direct those tools to execute each step in a separate conda environment, says Köster, who leads Snakemake development. "Make them as fine-grained and as single-purpose as possible," he advises. Besides being easier to maintain, he explains, small environments are also more transparent. "People who want to understand what the analysis actually did immediately see what software stack was used for which step."

Limitations

Still, environments can't do everything. Tools written in languages such as C, Perl and Fortran can be hard to encapsulate into environments, and dependency differences can make environments difficult to port across operating systems. In that case, users can try software containers, such as those from Docker and Singularity.

Containers, which essentially package a tool with its underlying operating system, are larger and more complicated than environments, but are more portable. They are also easier to share, because although an environment can hold thousands of files, a container has only one. On high-performance systems in which jobs can be run in parallel across hundreds of computing cores, transferring many small files can affect performance.

Computational environments, says Timbers, are "the forgotten child" of reproducibility. Journals increasingly ask for code and data alongside manuscripts, but full reproducibility requires knowing the environment in which they were run. "It's the elephant in the room," she says.

Jeffrey M. Perkel is technology editor at *Nature*.

ENVIRONMENTAL TESTING

An example of how variable computing environments can hinder collaboration.

Suppose you have the latest versions of R and Python installed, but your collaborator has been slower to upgrade. They want to share a Python script with you, and you have an R program you want them to use. Will the code work in each others' hands?

Between Python 2 and Python 3, the 'print' command that outputs text to the screen changed. The directive print "hello, world!" is valid in Python 2, but Python 3 requires parentheses — print ("hello, world!"). Similarly, before R 4.0, the function that creates spreadsheet-like data tables treated text as discrete 'factors' by default, whereas later versions do not.

To highlight these differences, we created scripts and environments for Python 2.7, Python 3.11, R 3.6 and R 4.2 (see go.nature.com/4tirjm7). Following the instructions (see go.nature.com/4tnd5ke), install conda. Then, open a terminal window, run the set-up script and execute run.sh. You should see the code working correctly in one environment but not in the other. For instance, although the R script behaves as intended in R 4.2 — it changes the gender of a study subject — it does something unexpected (and issues a warning) in R 3.6.