

CDE: A Tool for Creating Portable Experimental Software Packages

One technical barrier to reproducible computational science is that it's hard to distribute scientific code in a form that other researchers can easily execute on their own computers. To help eliminate this barrier, the CDE tool packages all software dependencies required to rerun Linux-based computational experiments on other computers.

Although there are many social, cultural, and political barriers that hinder reproducible computational science research,¹ one technical barrier to reproducibility is that it's hard to distribute scientific code in a form that other researchers can easily execute on their own computers. Before your colleagues can run your computational experiments, they must first obtain, install, and configure compatible versions of the appropriate software and the myriad dependent libraries, which is often a frustrating and error-prone process. If even one portion of one dependency can't be fulfilled, then your experiment won't be re-executable.

To eliminate this technical barrier to reproducibility, I created a tool called CDE—which stands for Code, Data, and Environment packaging—that automatically packages all of the software dependencies required to run your computational experiments on another computer. CDE is easy to use: all you need to do is execute the commands for your experiment under its supervision, and CDE automatically packages all of the code, data, and environment that your commands accessed.

When you send that self-contained package to your colleagues, they can rerun those exact commands on their computers without first installing or configuring anything. Moreover, they can even adjust the parameters in your code and rerun it to explore related hypotheses or run your code on their own datasets to see how well your techniques generalize.

By using CDE to package your experimental code, data, and environment when you publish a paper, you can ensure that both you and your colleagues can reproduce the paper's results in the future. CDE currently works on 32- and 64-bit x86-Linux operating systems. In short, if you can run the original experiment on your own Linux computer, then your colleagues can run and modify it on their Linux computers without any setup effort.

CDE is free, open source software; you can download it and view its documentation at www.pgbovine.net/cde.html. In this article, I provide a high-level overview of how CDE can help computational scientists. Other articles^{2,3} provide details on the design, implementation, and formal evaluation of CDE.

CDE Usage Example: Alice and Bob

The best way to get a sense of how CDE works is through an example. Let's say that Alice is a climate scientist who's running weather simulations

1521-9615/12/\$31.00 © 2012 IEEE
COPUBLISHED BY THE IEEE CS AND THE AIP

PHILIP J. GUO
Stanford University

for her research. Her experiment consists of a single script written in the Python programming language (`weather_sim.py`) and a data file representing Tokyo weather data (`tokyo.dat`) located in her `/home/alice/cool-experiment/` directory. She normally runs the experiment by typing the following Linux shell command:

```
python weather_sim.py tokyo.dat
```

When that command is executed, the shell finds the `python` executable within `/usr/bin/` and invokes it with `weather_sim.py` and `tokyo.dat` as its arguments. Figure 1 shows all the files involved in running this command: first, the `python` executable (underlined in red) loads the standard C library (`libc-2.10.so`) and the `weather_sim.py` script file. Then, `weather_sim.py` loads the `tokyo.dat` data file and the `py-weather.so` library, which contains optimized weather simulation subroutines.

Note that `py-weather.so` is an example of a third-party Python extension library that doesn't come preinstalled on Alice's computer. Before she could run her experiments, Alice (or her system administrator) had to first install this library and configure her version of Python to be able to find and use it. This process might have taken hours or days of frustration, and she likely didn't document the installation steps for someone else to repeat at a later time.

Now, let's say that Alice's colleague Bob wants to reproduce her weather simulation experiment and modify it to test some related hypotheses. Bob simply asks Alice to zip up and email her entire `cool-experiment/` directory to him. He unzips the directory on his computer, navigates into it, and then tries to run her script in the same way that she originally did:

```
python weather_sim.py tokyo.dat
```

Bob thinks that he should have no problems running Alice's script, because Python came pre-installed on his Linux computer. However, when he tries to run her script, it crashes with an error because the `py-weather.so` library can't be found (see Figure 2). He must now go through the trouble of installing `py-weather.so` and configuring his computer's Python interpreter to be able to find and use it.

This example is actually oversimplified. In real life, Bob might have to install and configure several software libraries, which themselves might depend on even more libraries or conflict with

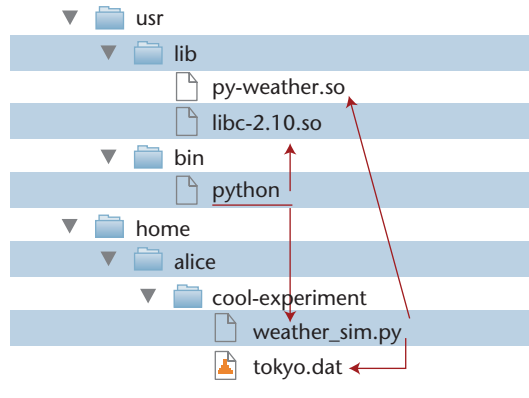


Figure 1. Alice runs her Python-based weather simulation experiment. The Python executable (underlined in red) loads a Python script, a data file, and two shared libraries.

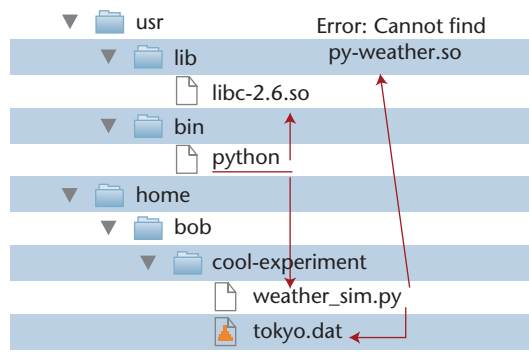


Figure 2. Bob tries to run Alice's experiment but encounters an error. Although he has Python installed, he does not have the custom `py-weather.so` library installed, which Alice's script requires.

those already installed on his computer. It could take him hours or days of frustration before he finishes setting up the proper dependencies to run Alice's script, and he could inadvertently break other programs on his computer in the process (for example, because of conflicting library versions or misconfigurations). Let's see how CDE can eliminate all of these frustrations.

Creating a CDE Package

After Alice downloads CDE to her computer, she can create a self-contained package for her experiment by simply prepending its original command with the `cde` executable:

```
cde python weather_sim.py tokyo.dat
```

CDE executes her script and uses the Linux `ptrace` interception mechanism to monitor all of the files that it accesses. CDE creates a `cde-package/` subdirectory and copies all of those accessed files there, mirroring the original

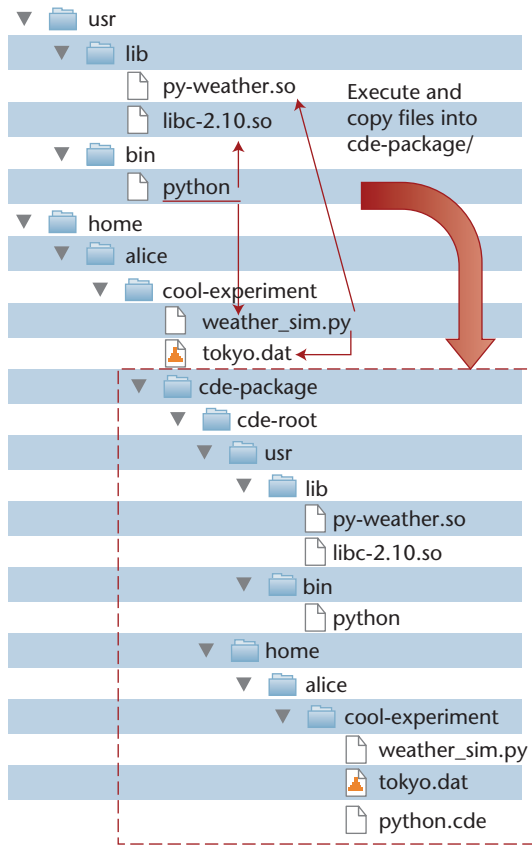


Figure 3. Alice creates a CDE package that contains all of the code, data, and environment that her Python script accessed when she ran her experiment. CDE mirrors the directory and file structure of all files that her experiment accessed (illustrated within the dotted red box).

directory structure (see Figure 3). CDE also creates a `python.cde` wrapper program in the package, which is a portable version of Alice's original `python` executable.

After Alice's script finishes executing, the `cde-package/` subdirectory (the dotted red box in Figure 3) now contains all the files required to run her script on another Linux computer. CDE has packaged her code (`weather_sim.py`), data (`tokyo.dat`), and environment (the standard C library, Python interpreter, and `py-weather.so` extension library). A package can range from several megabytes to several hundred megabytes in size, depending on its payload.

So, creating a CDE package is as simple as running the original program under its supervision.

Executing a CDE Package

Alice can now transfer her entire `cde-package/` directory to Bob (via email or file upload). Bob can run Alice's script by changing into the `cool-experiment/` subdirectory within the package and

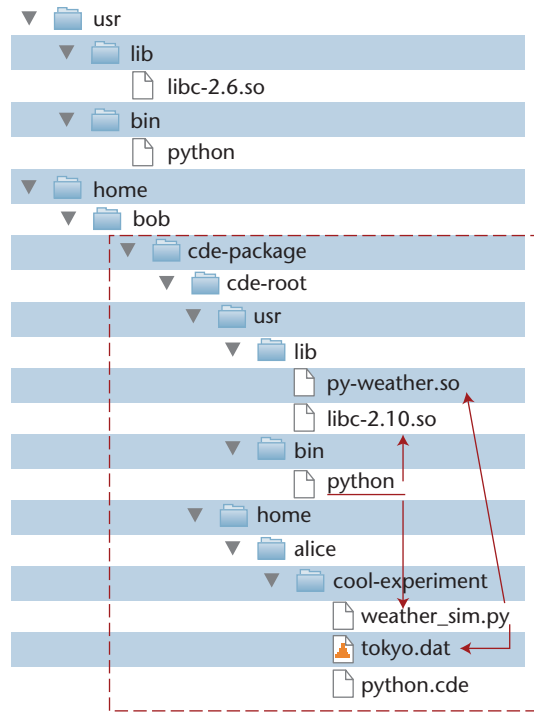


Figure 4. Bob runs Alice's experiment from within her CDE package. CDE creates a temporary sandbox by redirecting all file-access system calls within the subdirectory denoted by the dotted red box.

running the special `python.cde` wrapper program with the same arguments as Alice's original command:

```
./python.cde weather_sim.py tokyo.dat
```

Note that this command looks almost exactly like the command that Alice originally ran on her computer.

The `python.cde` wrapper first creates a *sandbox* within the package (the dotted red box in Figure 4) and then invokes Alice's version of Python (underlined in red). Alice's Python knows how to find the `py-weather.so` library, so her script runs properly, just like it ran on her own computer.

All of the file access arrows in Figure 4 remain within the sandbox. CDE uses Linux `ptrace` system call redirection to ensure that commands under its supervision can access only files within the sandbox, so they can't interfere with the rest of Bob's computer. Thus, even though Bob has Python and an older standard C library (`libc-2.6.so`) installed on his computer, CDE always accesses the versions from within Alice's package.

Programs executed from within CDE packages will run slightly slower because of the system call

redirection overhead. In my experiments, slowdowns ranged from negligible to 30 percent.²

In essence, CDE lets Bob transfer a “slice” of Alice’s computer onto his computer, so that he can safely run and modify her experimental code. Bob doesn’t have to install any software dependencies before running Alice’s weather simulation script. In fact, he doesn’t even need to have root (administrator) access, so he can run her CDE package on, say, a shared university computer cluster. In addition to reproducing Alice’s script run, Bob can also modify `weather_sim.py` to explore alternative hypotheses, test other datasets, or write new scripts that build off Alice’s script.

CDE isn’t limited to Python; it works on arbitrary Linux programs written in any language. If Alice can run a command on her computer, then CDE enables her colleagues to run that same command on theirs.

CDE Package Portability

Alice’s CDE package can execute on any Linux computer with an architecture and kernel version that are compatible with its constituent binaries. CDE currently works on 32- and 64-bit variants of the ubiquitous x86 architecture. Users have been able to create CDE packages on modern x86-Linux computers and run them on versions of Linux that are up to five years old. However, CDE doesn’t emulate software licenses or custom hardware, so those are additional limits to portability.


Users can combine CDE with a virtual machine to achieve greater portability. For example, if Alice wants her colleagues who run Windows, Mac OS, or an antiquated Linux to reproduce her experiments, she can put her CDE package within a Linux virtual machine (VM) and distribute the entire VM image. However, the price to pay for such portability is increased file size: A VM image file can be 10 to 100 times larger than a CDE package because it contains the entire operating system.

Finally, unlike language-based portability technologies (such as Java or Python `virtualenv`), CDE works on Linux programs written in any language or mix of languages.

Here, I focused on how scientists can use CDE to instantly make their Linux-based computational experiments portable across a wide range of Linux distributions. However, others have

found many creative uses for CDE beyond experiment reproducibility:

- Researchers, designers, and hobbyists have used CDE to distribute their prototype software in a portable format so that users can instantly run their software without the hassles of installation.
- Scientists have used CDE to deploy “embarrassingly parallel” computations to clusters and cloud computing (such as Amazon EC2) without needing root access or installing dependencies on the remote machines.
- Web developers have used CDE to deploy custom software stacks to their hosting providers’ Web servers without needing root access.
- Students have used CDE to collaborate on class programming assignments without requiring each teammate to go through a laborious software installation procedure.
- People have used CDE to run software that’s hard to install on their preferred Linux distribution due to library incompatibilities. They first install the desired software on a compatible Linux distribution (often within a VM), package it using CDE, and then transfer that package to their own computer to execute.

Because CDE is a research project, I’m still actively recruiting new users to evaluate its effectiveness in real-world use cases. Visit www.pgbovine.net/cde.html to learn more and try it out. 

References

1. V. Stodden, “The Scientific Method in Practice: Reproducibility in the Computational Sciences,” *MIT Sloan Research Paper No. 4773-10*, 2010; <http://dx.doi.org/10.2139/ssrn.1550193>.
2. P.J. Guo, “CDE: Run Any Linux Application On-Demand Without Installation,” *Proc. 2011 Usenix Large Installation System Administration Conf.*, Usenix Assoc., 2011; http://static.usenix.org/events/lisa11/tech/full_papers/Guo.pdf.
3. P.J. Guo and D. Engler, “CDE: Using System Call Interposition to Automatically Create Portable Software Packages,” *Proc. 2011 Usenix Annual Tech. Conf.*, Usenix Assoc., 2011; http://static.usenix.org/events/atc11/tech/final_files/GuoEngler.pdf.

Philip J. Guo recently graduated from Stanford University with a PhD in computer science and now works at Google Research in Mountain View, California. Visit www.pgbovine.net to learn more about his research interests, which involve making programming easier for people who aren’t professional software engineers. Contact him at philip@pgbovine.net.