



Python in Systems Administration: Part III -- Pexpect Automates Hard-to-Solve Problems

Cameron Laird

Unix systems administrators are familiar with Expect as a nearly indispensable tool for handling difficult problems involving passwords and other unusual forms of data entry. Few know, though, that Python can easily handle most of the same problems: Python has its own Expect.

Before demonstrating Python's Expect solutions, I want to tie up a few loose ends. This is the third in a series on what Python offers to systems administrators. I previously presented Python as an alternative to the shell and Perl programming languages that sys admins commonly use -- one that's both easy to learn and more adept than shell in such domains as networking and graphical user interfaces (GUIs).

Elegance and Balance

Several readers have raised questions based on those earlier installments, so my first order of business this month is to address those.

Greg Wyman correctly spotted a missing close parenthesis in `my_restore.py`. The `os.popen()` call printed in the December issue should have been:

```
print os.popen('tar xf %s %s' % (device, files)).read()
```

Mr. Wyman also astutely wondered why that same example program didn't take advantages of new features available with Python's 2.3 release to simplify the list comprehension. He's right -- iteration over a file object and substring-testing allow simplification to:

```
list_of_qualified_names = [file[:-1] for \
    file in os.popen('tar tf %s' % device) \
    if pattern in file]
```

That *is* a more elegant formulation and helps make the case for Python's readability. Moreover, the plans for 2.3 were clear long before both its release and the deadline for December's *Sys Admin* magazine.

I decided to focus on functionality available in 2.0 and, occasionally, 2.2, to help ensure a "good experience" for readers. Nearly all of the hosts for which I do systems administration are currently at 2.2 and before, and I expect the same is true for most readers, in early 2004. I want the attention of such readers to be on working programs, rather than on locating and installing updates.

It's true, though, that Python continues to improve, and almost certainly will do so in the future. For a snapshot of what a new release provides, see the online article "Python 2.3" that I co-authored:

http://www.byte.com/documents/s=8880/byt1062182129207/0901_laird.html

Paddy McCarthy makes several points that are difficult to summarize briefly. Most starkly, he wonders why I promote allegiance to the Unix model of "many smaller utilities doing one thing well, connected via pipes", yet script the Tkinter example as a standalone executable. He's right to wonder, and his letter, which you can read in full at:

<http://mail.python.org/pipermail/python-list/2003-December/199071.html>

presents significant alternatives. The real problem is simply the difficulty of systems administration. In a follow-up, Donn Cave observes that it "isn't really a computer programming domain like numeric, hardware control, rendering, [or Web publishing]", but more like the institutionalization of "Miscellaneous Little Tasks". In this series, I've generally aimed to provide examples that stand on their own, but are otherwise as minimal as possible. My concern is more to make ideas clear, so you can apply the ideas to your own situation, rather than to supply finished solutions.

The second installment mentioned IPython as an alternative shell. This mailing list thread:

<http://mail.python.org/pipermail/python-list/2003-December/198584.html>

details the latest news about what it's like to use IPython for daily work.

Scripting the Unscriptable

Let's return now to this month's main subject, and start by asking, "What's the point of Expect?" Suppose you need to reset a password. That's a common occurrence, one that `/usr/bin/passwd` addresses. `passwd` does everything necessary for the most common situations systems administrators face.

Suppose, though, you want to *automate* a `passwd` solution -- do a bulk re-assignment of the passwords for everyone in a particular college course, say, or set up a simple CGI-driven page that allows remote users to change their passwords. For a long-term solution, you shouldn't be using conventional Unix `/etc/passwd` accounts, or CGI, of course. There are too many security vulnerabilities to make those safe except with very careful analysis and hardening.

That sort of safety isn't always available to us. One of the great challenges of systems administration is its *reactivity*. There is no "long term" for many tasks; the idea is just to fix things up and get them running with as little delay as possible. The point of Expect isn't to encourage "hacking" or a short-term approach. Instead, Expect simply offers to automate jobs you'd be doing by hand otherwise. In this example, `passwd` does the job fine, as long as you're willing to type when *it* is ready. Redirecting a script into `passwd` doesn't work -- as with many programs that receive sensitive information, it disables keystroke echoing in a way that makes input redirection infeasible.

Pexpect, available through:

<http://sourceforge.net/projects/pexpect/>

changes all that. It's a tiny download compatible with any recent installation of Python. Once you've unpacked it, you can write Python programs such as:

```
#!/usr/bin/env python
'''Change password for one user.
```

Save this source as 'set_password.py'.

```
A production version would handle exceptions, explain that it can
only be run as 'root', and so on.
'''
```

```
import pexpect, sys

if len(sys.argv) != 3:
    print "Usage: set_password.py user new_password."
    print "%d and %s." % (len(sys.argv), sys.argv)
    sys.exit(1)
(user, new_password) = (sys.argv[1], sys.argv[2])
child = pexpect.spawn("passwd %s" % user)
for repeat in (1, 2):
    child.expect("password: ")
    child.sendline(new_password)
```

With this little program in hand, password management suddenly becomes scriptable. If you like, you can run this utility from a shell script, on the model of:

```
#!/bin/sh

INITIAL_PASSWORD=abcd1234
for USER in "user1 user2 user3"
do

    set_password.py $USER $INITIAL_PASSWORD
done
```

Different Kinds of Expect

This capability is monumental -- at least, it has been to me. With Expect, I frequently solve in half an hour problems that would take all day by hand. I am very grateful for the confidence it gives me that, if I can just understand how to solve a specific instance of a problem in systems administration or data entry, I'll be able to generalize it to an automation that runs on its own. In my work, Expect has been particularly valuable for automating such "embedded applications" as are common in routers, switches, and other network hardware, along with lab equipment.

Don Libes, a computer scientist with what's now known as the National Institute of Standards and Technology, wrote the original for Expect more than a decade ago. He based his work on the Tcl programming language. His version, and the influential book he wrote about it, *Exploring Expect*, have been so successful that many programmers and administrators believe Expect is available only in its Tcl manifestation. Libes anticipated from the beginning, though, that other languages would take up the idea.

Most important is to use *some* flavor of Expect. This makes for a big boost in productivity beside which the differences between the Expects for different languages pale. The original Tcl-based Expect has several advantages:

- The book is so well written and full of useful ideas that I recommend it even to those running Expect based on languages other than Tcl.
- Libes' version is more mature, with careful workarounds for numerous operating-system-specific faults and blemishes.
- It has advanced facilities such as **interact** not implemented for all other languages.

Don't let those discourage you, however; Pexpect is more trouble-free to install than the original Expect, and its functionality is a close match. It has its own **interact()**, and works well on such common Unixes as Linux, *BSD, and Solaris.

If you have begun to experiment with Python in your systems administration work, Pexpect makes a natural complement. Be sure to pay particular attention to Pexpect's **examples/** directory; it includes sample Pexpect-code programs to play chess, run commands remotely through ssh, remove binary files

improperly added to a CVS repository, and more.

Next month, I'll investigate a bit deeper Python's abilities to build graphical user interfaces for systems administration tasks.

Cameron Laird, a vice president at consultancy Phaseit, Inc. (<http://phaseit.net/>), has written occasionally for Sys Admin in previous years.

Copyright © 2003 UnixReview.com, [UnixReview.com's Privacy Policy](#). Comments about the Web site: webmaster@unixreview.com