



Python in Systems Administration: Part V -- Python Networking

Cameron Laird

Python makes network administration scriptable, which is a wonderful thing. As Unix systems administrators, our responsibilities reach beyond configuration of a single host. We also cultivate and repair the whole range of client-server networking that underlies early 21st-century computing. Too many times, though, we try to do this job with end-user applications that are clumsy for our needs. There's a better way.

For example, a few years ago, I was in a cycle of upgrading email services. At first, I could only define project goals in terms of the end-user experience. If a networked end-user could retrieve her inbox with an acceptable delay, then I declared the POP3 service good.

Opening the Hood

That's a perfectly appropriate target, but it's a mistake to confuse the criterion with the diagnostic method. Think of the task of automobile repair. When an owner complains that his car bucks around 44 mph, a test drive has to validate any remedy. That doesn't mean the way to fix the car is to cruise around until inspiration descends. Instead, a good mechanic brings the car into the shop, opens up the hood, and breaks down the problem into possible component causes.

We need to do the same. Don't let the wrapping of user interfaces prevent you from seeing the problems and root causes inside them. When I started the upgrade cycle, I knew little more about POP3 than how to configure a particular Netscape email client to a particular email server. That scales poorly. Diagnosing and validating POP3 for an entire user community that way is clumsy and time-consuming.

So I came up with better ways. I wrote tiny little scripts that helped me monitor server operations quicker and more precisely, and others that tested clients and networks. These scripts paid off in a big way. When a user reported an outage, I could check his or her mailbox and connectivity in a few seconds, rather than the couple of minutes it took me to bring up a Netscape, save its configuration, define a new account, try out a download, read the result, and restore my Netscape.

Scripting's a good idea, one that deserves to be understood more widely. One reader, Patrick Rateeff, responding to an earlier installment of this Python series, wrote: "The hardest thing I have found ... is not the OS or understanding things, but thinking about problems differently, knowing that I can develop my own solutions that function. In the past ... I would look for a shareware or commercial application that addressed the issue. ... I start to see how much more I could do."

Many Unix scripters, though, think in terms of the Bourne shell and its immediate descendants. With **sh**, scripting generally "glues" together external processes that do "real" work, especially in networking. That can be handy, just as off-the-shelf solutions have their place.

Python Expands Possibilities

Python opens up a whole new domain of scripting, however, particularly for networking problems. Python builds in a full range of networking functionality, including knowledge of such standard protocols as FTP, POP3, and IMAP4, as well as low-level socket-oriented programmability.

This is a remarkable achievement, and just think what it means. As earlier articles in this series have shown, Python scripting is as easy to learn as **sh**. If you have a need, though, you can move far beyond **sh**'s capabilities, all the way to the low-level precision usually associated with C. If any computer application can do a particular task, a Python-coded application can probably do that task, too, with all the productivity and maintainability of a high-level language. Only a few requirements -- principally raw computing speed -- disqualify Python.

How easy is Python networking? Here's a basic POP3 "inspector":

```
import poplib

account = "XXX"
password = "XXX"
host = "mail.XXX.XXX"
pop = poplib.POP3(host)
pop.user(account)
pop.pass_(password)

print "This mailbox has %d messages, totaling %d bytes." % pop.stat()
```

Judge for yourself how easy this is. The standard Python Library Reference chapter on "Internet Protocols and Support" demonstrates that Python makes other networking standards as accessible as POP3. The reference is available at:

<http://python.org/doc/current/lib/internet.html>

Low-Level Sockets

Full coverage of Python networking extends to low-level socket programming, most accurately introduced in Gordon McMillan's "Socket Programming HOWTO", which is available at:

<http://www.amk.ca/python/howto/sockets/>

I recommend "Internet Protocols..." first, though, for a couple of reasons. First, protocol-level support is far likelier to match most systems administration chores in my experience. Second, "raw" sockets are more of a development than administrative concern.

When administrators do need to move outside the defined network protocols, I think interfaces slightly higher than those of the "HOWTO" serve us best. I'm a big fan of the `asyncore` module:

<http://www.python.org/doc/current/lib/module-asyncore.html>

and, even more, the Twisted networking framework:

<http://www.twistedmatrix.com/products/twisted>

These have all the power of low-level sockets, but are packaged in a way that makes it easier to write correct programs. With Twisted, for instance, all it takes to write a minimal "echo server" is:

```
class Echo(protocol.Protocol):
    """This is just about the simplest possible protocol"""

    def dataReceived(self, data):
```

```
        "As soon as any data is received, write it back."
        self.transport.write(data)

def main():
    """This runs the protocol on port 8000"""
    factory = protocol.ServerFactory()
    factory.protocol = Echo
    reactor.listenTCP(8000, factory)
    reactor.run()

# this only runs if the module was *not* imported
if __name__ == '__main__':
    main()
```

This is copied directly from the Twisted example page:

<http://www.twistedmatrix.com/documents/examples/>

whose other programs are almost equally easy.

Scripter's Attitude

The most important knowledge this series can give is the high-level attitude of a proficient Python scripter: if there's a computing job to be done, there's probably a good way to do it with Python. This truth shines brightest on networking, because other languages handle sockets either with difficulty, or not at all, while Python networking is succinct and sensible.

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

| |
|---|
| Copyright © 2001 Sys Admin, Sys Admin's Privacy Policy . Comments about the Web site: webmaster@sysadminmag.com |
|---|