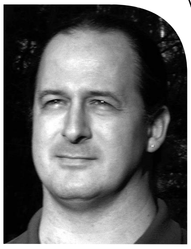
"eat your own dog food"

by Adam Moskowitz

Adam Moskowitz is a system administrator, programmer, manager of system administrators, certified barbecue judge, and author of the recently published SAGE Short Topics booklet *Budgeting for SysAdmins*.



adamm@menlo.com

You've no doubt heard this phrase before; you've probably even used it once or twice. As a system administrator, do you eat your own dog food? From the exact same can as your users? If not, why not?

In the late 1980s, years before the "dog food" phrase was ever uttered, I worked in Encore Computer Corporation's software development group. Not only did the operating system developers have to use the latest version of UMAX (our multiprocessor port of 4.3BSD), *everyone* in the group did, too. Talk about a good use of peer pressure! I can't prove that this practice actually improved the quality of our software, but when a user reported that something was broken, we tended to believe them, usually because we had already discovered the problem for ourselves.

When a user says, "Foo is broken," I think the worst possible response a system administrator can give is, "It works for me." It's even worse when the sysadmin is on a different hardware platform than most users, running a different operating system, and using a different shell. Eating your own dog food won't stop you from giving such a bad answer, but it at least mitigates your "sin." On the other hand, I think users find it comforting when your response is, "Yes, I've noticed this, too; I'm already working on fixing it." Eat your own dog food often enough and this is likely to become your usual response.

Here's how I think the concept should apply to system administrators:

First, use the same hardware platform and operating system as your users. If there are several, and it's not feasible to give every sysadmin in the group one of every platform, spread them evenly throughout the group. Apply the same patches to your machine as you apply to users' machines: no more and no fewer. Upgrade the OS on your machine no more than one day before you upgrade users' machines.

The obvious complaint at this point is usually, "But I have to test new patches/operating systems/whatever somewhere before we put them in production." Yes, you do, and that's what test machines are for. Treat your desktop system as a production (user) machine and do your testing elsewhere.

Next, shells: These are very personal things, and one's setting is even more so. I've always believed that an organization should support a (limited) number of shells, but to the greatest extent possible allow users to run whatever shell they want. By "support" I mean that an effort will be made to make sure that all supported programs run on each of the supported shells, and problems in this area will be investigated (and, one hopes, fixed) by the appropriate part of the IT organization. In an ideal world, users will be able to run all supported programs with an empty shell rc file, because the system-wide default file will contain all the required settings. If that's not possible, new users should be given an rc file that looks something like this:

```
# Uncomment the next line if you plan to run ABC
# . /etc/rcfiles/abc
# Uncomment the next line if you plan to run DEF
# . /etc/rcfiles/def
```

I know it would be folly to suggest that system administrators be forced to use the same shell(s) as users. Instead, I propose that there be a test account for every shell, either without an rc file or with the same one given to new users. If it contains lines like those shown above, uncomment them as appropriate to test a given problem (or to match the users' settings), then comment them out again when you're done. Here's a concrete example of what I mean:

```
joesixc:*:99901:23:CSH test account:/home/joesixb:/bin/csh
joesixk:*:99902:23:KSH test account:/home/joesixk:/bin/ksh
joesixs:*:99903:23:SH  test account:/home/joesixs:/bin/sh
```

(I once worked on a project where "Joe Sixpack" was the name we gave to our target customer.)

If a user reports that something isn't working, and it isn't something obvious like a malformed command, log in to the appropriate account on the affected machine, modify the rc file as necessary, and only then try to duplicate the user's problem. If, after trying all this, it still "works for you," say to the user: "I can't seem to duplicate the problem, I'll have to come to your office and work with you to figure out why you're having the problem."

If you write tools, for use either by users or your fellow system administrators, you should make it a point to use those tools and not the underlying "raw" commands (if such commands exist). For example, I once wrote a set of Perl scripts to manipulate DNS files without having to use an editor, and some very simple Web forms to call those scripts using CGI. The other UNIX admins used the command line version, and the Windows and Macintosh guys used the Web forms; the idea was to allow anyone in the group to make DNS changes without having to worry (too much) about the persnickety file formats. Even though there were three of us in the group who were perfectly capable of editing the files by hand (and getting it right, at least most of the time), the agreement was that we would always use the tools I had written.

There were several interesting results from this "experiment": First, the number of times DNS broke because one of the senior admins had edited the files by hand and gotten them wrong (or forgot to increment the serial number) dropped significantly. Second, we found several bugs in the tools that the junior people wouldn't have discovered for a while (if ever). Third, we came up with several modifications to the tools that made them even more useful to the junior staff.

Clearly, eating my own dog food not only helped improve the quality of the tool I had written, it actually helped me reduce the number of mistakes I made while doing my job. That is, after all, the reason for writing such tools in the first place – but what good are they if you don't use them?

There are other ways in which the "dog food" concept can be applied to system administration, but I trust that you can figure them out for yourself.

If you write tools, for use either by users or your fellow system administrators, you should make it a point to use those tools and not the underlying "raw" commands.

October 2003 ;login: "EAT YOUR OWN DOG FOOD" 19