How to Be a Better System Administrator and Then Something Else

ELIZABETH ZWICKY

After years as a system administrator, Elizabeth has experienced an almost full recovery but happily has maintained her ability to intimidate the help desk into providing sensible answers. Zwicky@otoh.org

I once was a system administrator. Now I'm not. This caused somebody to think I might have useful career advice, which is absurd if you look at my career trajectory, which looks like a ball of string after the kitten got to it. In fact I am, as the saying has it, a highly paid computer professional and have been all along, and most of the things that have caused people to give me other jobs are skills I honed as a system administrator.

There is one thing which will make no difference to your performance as a system administrator but will make getting another job in technology vastly easier, and that is a college degree, preferably in computer science or a related field. (No, I don't know what "a related field" is, except I have evidence that if you make this claim and turn out not to have had any classes involving a computer, people will fire you.) This is a fact about the job market, and its actual wisdom or lack thereof is irrelevant. If you can get a college degree, do so; and, if involving computing in it is still a reasonable option for you, drag some computers into it.

Then learn to code. If you do this by solving problems at work, it will make you a better system administrator. The process will also teach you things about coding that most courses and books don't teach well. If you ask my colleagues, they will tell you, with affectionate condescension, that I "don't really code." You might think, based on that, that I rarely write code (in fact, I do so almost every day), or that I never modify production systems (in fact, I am one of the people called upon to write changes directly into major customer-facing systems when that needs to be done on the fly), or that I don't use the main programming languages we use (this is almost true, but I do review and fix code in all of them).

In fact, what they mean is that I don't spend all day writing code, and that I almost never produce an entire application myself.

I really learned to code as a system administrator. I have a computer science degree, which as I mentioned above is a valuable piece of paper, and it certainly taught me what variables are, how command structures work, and why $O(n^2)$ is bad. However, it is the single least relevant piece of my computing experience to my work life—well, behind my teenage days trying to make BASIC play Animal better. Because my real learning was on system administration tasks, it leaned heavily on regular expressions, on reading and porting other people's code in whatever language they chose, and on building utility chains. This means that if you want an application written from the ground up, I am nearly useless at actually producing the code. On the other hand, if you want somebody to do that with moderate competence, you can probably find three at the nearest coffee shop. All of them will blanch if you ask them to write a regular expression, and if they do write it, it will contain at least one unnecessary and dangerous ".*.".

So embrace the funkiness of writing the code that your situation needs. It may not look like coding in school, or even coding as a full-time programmer, but it will give you the ability to round out a programming team.

And embrace the rest of the limitations you may face. You have to debug third-party software with no source code? Great! You will never be any of the people who took my bug reports and

closed them because "the code doesn't do that," which is a phrase liable to inspire me to violence. I really don't care whether the code does it or the pack of trained gerbils living under the floor does it. If I reported it as a bug, I want it to stop already. You will also not be the person who never uses a debugger because you can always just put in some print statements, right? (No, you can't.)

Learning to debug distributed systems written by other people will also give you a head start on being data-driven, because when you are trying to figure out what happened on somebody else's computer system, you don't have a lot of choice except to start digging data out of the system. What's in the logs? What files got changed? Programmers who can work in a change-and-try system don't get forced to that mode of thought. Admittedly, only the good system administrators do. Debugging is a theme, and excellence at system administration is what turns into transferrable skills.

What else goes with that theme?

- **Understand the technology you work with; in particular, know how networks and file systems work.** For some reason, computer science degrees manage to get people to recite facts about compilers, operating systems, and maybe even databases, but most of them are not entirely sure the Internet is not made of tubes.
- **Learn to be good in emergencies.** System administrators get called in the middle of the night when things are broken and people are screaming. This builds important life skills for times you don't always have a pager. In particular, it teaches you to be very, very careful, because everybody is stupid in the middle of the night and you need to be able to recover. I don't edit important files in place. I don't do it much in broad daylight when I'm relaxed and caffeinated, but I don't do it at all when the stakes are high—I don't trust software and I don't trust myself. So I make a copy, and a backup copy, and then I edit the copy, and then I copy the edited copy to the original location and see what happens. You can argue about the details of this process—I'm sure you're dying to inform me that all your files are in source control, which is even better—but the point is that somebody has already paid to have this caution instilled in me, which is a win for all my recent employers.
- **Learn to be customer-oriented.** I don't care if you call the people who use your systems "users," but you need to be able to think about what every change means to them. Again, this is a skill that differentiates senior people from junior people, regardless of where you are, and it's one you can learn in stark and dramatic terms in system administration, when your users are often senior to you and capable of physically yelling at you.
- **Learn capacity planning and performance tuning.** This is full of useful mathematics; if you don't figure out why the mean is not the interesting average when you start looking at how users use disk space, or send email messages, or how your network is utilized, you are never going to understand numbers. Building big production systems is dependent on an ability to think about the numbers in ways they don't teach in school, based on the number of mid-level programmers who, when asked questions like "Will I be able to do that in my 100-millisecond budget for this operation?" or "How much space will that data take up?" give answers like "It's really fast" and "It shouldn't be a problem in the cloud." (Hint: I didn't ask idly. If your company built cloud technology, it is because it has problems that require a cloud, instead of problems that are trivial with a cloud.)
- **Learn about security.** When you're responsible for the system, you're responsible for protecting it. This is the time to learn about threat models and attack surfaces. Engineering a system to resist an active attacker is a different skill from engineering one that is resilient to pure error. And those debugging skills I mentioned earlier? They're security skills as well, because one of the chief jobs of security and abuse teams is distinguishing security and abuse issues from other issues. Not only are programmers bad at security, but carefully purpose-trained security people spend years seeing bad guys behind every crash. Converted system administrators have a finely honed respect for the basic malevolence of the universe.
- **Learn to pick up new technologies.** Again, as a system administrator, you are often pretty much stuck adapting yourself to the choices of your management and the people you support. Embrace and extend; get good at figuring out the rudiments of whatever it is they are doing. When I am hiring new programmers, my first choice is one who knows one of the languages we use a lot really well but is happy to learn others. My second choice is somebody who doesn't really care deeply what languages we use a lot but knows a little bit about a couple of our languages. The person who has a really strong opinion about what languages we ought to use is not very high up on my list, no matter how good they are at the languages they know, and even if they think our current choice is right, because we slowly but surely change our language balance over time.

There are dozens of other things I could add. I've mentioned security, programming, and network engineering, but system administration could also lead you into release engineering, or technical writing, or project management, or database administration, all skills that are careers on their own but which system administrators are often expected to do on their own. I hope I have convinced you—and helped you to convince future employers—that system administration is not only valuable work in itself, but also a solid platform for doing other work if that's of interest.