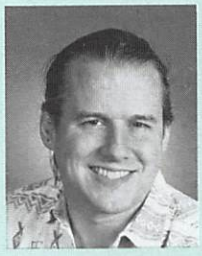
On Reliability – System Administration



by John Sellens

John Sellens has recently joined the Network Engineering group at UUNET Canada in Toronto, after 11 years as a system administrator and project leader at the University of Waterloo.

<jsellens@uunet.ca>

This time around, let's talk about system administration and how to do it more reliably. But that's what this entire series of articles is about, isn't it? Maybe I should narrow it down a little bit. Let's talk about reliability in the parts of system administration that actually takes place on a machine. First, though, let's try to define what system administration is and then discuss what things this article isn't going to be about.

"System administration" includes all the "overhead" aspects of creating and keeping a "system" running and available. Nope. This isn't working out either. Let's try another approach.

Once you have your hardware installed and your network working, there are still all the day-to-day tasks of installing software, maintaining user accounts and compiling mailing lists, monitoring, and repairing. The software- and configuration-related tasks of system administration are what I'm going to attempt to cover this time around. And I hope that's a clear enough definition, because I'm all out of ideas. For lack of a better term, I'm going to call this "system administration," but we all know that it's just one part of the total system administration workload.

Of course, no reliability article would be complete without a reminder of the basic principles: service levels, risk evaluation, costs of failures, finding the right balance, etc.

So far in this series, I've focused on hardware and wiring and touched on tasks and processes only superficially. Reliable system administration (at least as I have defined it for this article) tends to be much less related to hardware and much more related to the tasks themselves and the performance of those tasks.

I'm going to outline a few key words for reliable system administration, give a few examples of how to apply them, and then mention a few topics that don't fit neatly into those categories. (This matches nicely with system administration in general, because nothing ever fits together quite as well as you hope it will.)

Key Words

Allow me to propose a short list of important key words for reliable system administration:

- Consistency. Set your standards and procedures and stick to them; all sorts of things will be easier and more reliable.
- Documentation. If it's not properly documented, it won't pass either the "run over by a bus" test or the "won the lottery" test (i.e., if all the documentation is in your head or your mailbox, the system is unlikely to survive your sudden disappearance).
- Automation. Never do something twice by hand if you can write a program to do it for you. If tasks are (properly and carefully) automated, they're less likely to fail as a result of your being distracted or rushed or having fumbling fingers.
- Repeatability. This is often a combination of the previous three. Proper documentation and automation will help you set up a new machine (or recover or upgrade an existing one) far more effectively and reliably.

Let's talk about these in a little more detail with a few examples.

I'm convinced that one of the most effective tools in a system administrator's arsenal is consistency. Your users will thank you, your friends and family will thank you, and your evenings and weekends will thank you, too.

Consistency

This really is one of the cornerstones of system administration. If you're not consistent in how you do things, you're going to be much less effective, much less able to delegate tasks, much less able to scale your installation past a small number of systems, and much less able to put together systems that run reliably and are easily recoverable in the event of a failure.

Let's pick on everybody's favorite topic for an example: backups. (We'll be able to use this same example later for the other key words as well.) If you have inconsistent backup methods, using different commands or schedules on all your machines, I would venture to guess that you're going to find doing backups a terrible, onerous chore and not do it as well as you would hope to. And if doing backups is a complicated and convoluted task, it will be hard for you to pawn the job off on some underworked clerk (we all know, of course, that the only truly overworked people in a company are the system administrators), giving you less free time. Different backup methods for each system mean that once you get beyond a small number of machines, you will likely find it simply too complicated to do proper backups. And lastly, if you don't have one standard way of doing backups and restores, of cycling your tapes off-site, and of keeping track of which backups are on which tapes, you're going to have a heck of a time putting things back together when your company Web site goes boom, and your boss, the head of sales, the head of marketing, and the CEO are all standing around breathing down your neck.

Have I convinced you yet? OK, then how about another perennial favorite: software installation, configuration, and distribution (and if you don't believe me that it's a favorite topic, have a look at the proceedings of any of the 11 LISA conferences, and you're bound to see at least one or two related papers each year). A simple example here is the fabulous GNU "autoconf" system, which is used to generate those nice "configure" scripts that you see in so many software distributions. In the olden days, when you grabbed some software from the net, you had to read the READMEs, examine the Makefiles and .h's, and manually customize everything to suit your environment. Nowadays, if there's a "configure" script included, you can just about always rely on

```
% ./configure --prefix=/local
% make install
```

to do just what you would hope it would do, a concrete example of the time- and effort-saving benefits of consistency.

If you look back at the LISA software installation and distribution papers, one thing that they virtually all have in common is a set of rules defining how and where to install your software "packages." Decide on one place to put your source, one structure in which to install your binaries and supporting files, and follow your rules religiously. It's much easier to be able to tell someone that the source is under `/local/src/pkg` and the installation is under `/local/dist/pkg`, than to make everything you do a special case.

Consistency applies to almost everything: day-to-day procedures, how user accounts get authorized and created, where your backup tapes are stored, how IP addresses are assigned, and on and on. Whether you call them rules, policies, procedures, or practices, once you set 'em, don't forget 'em. I spent quite a few years doing centralized system administration and software support for hundreds of different machines, with different operating systems and revisions, for thousands of users; and I'm convinced that one of the most effective tools in a system administrator's arsenal is consistency.

Your users will thank you, your friends and family will thank you, and your evenings and weekends will thank you, too.

Documentation

If you don't document everything, you're not doing your job. You're making things harder for the users (is there anyone who doesn't get frustrated when the "man" page and user documentation are missing and not to be found?) and harder for yourself and your co-workers (because you'll end up answering the same questions over and over); and if all the documentation is locked up inside your head, you're making yourself unpromotable and untransferable.

There always seems to be a strong perception that "documentation is hard" or "I'm not a good writer," or "I don't have time." I'll tell you, from experience, that man pages for most programs get to be trivial to write after you've knocked off a dozen (or a gross) or so. And who hasn't had the experience of returning to a program or system after a few months and being unable to remember or determine what your beautiful, "self-documenting" code is supposed to be doing?

As I mentioned before, the favorite metric for how much documentation is enough is the "if you got hit by a bus" rule – the question being whether your systems would keep on running if you got hit by a bus tomorrow. (I prefer to use the "if you won the lottery and suddenly quit your job to move to Tahiti" rule because that seems much more cheerful.)

Make a man page (if you're a UNIX administrator) or a Web page (for anyone else) for every command or process that you install. Document each "process" or "job step," document your "cron" jobs and mail aliases, document your installation standards, your policies and procedures, your vendor service contract information, and make your life easier. Document your backup and recovery practices and processes so your minions can take care of things while you sleep. And, of course, once you have an online copy, you can make your all-important paper copy so you won't be completely stranded when your root partition dies. Set up a "cron" job or something to automatically (oops, that's the next key word) print an updated copy every few months; obsolete emergency recovery documentation isn't much fun. And finally, don't forget to write down the root password somewhere so that your systems can survive a bus crash, even if you don't (even though it's not likely to be a significant legacy handed down through the generations).

Automation

Automation is your friend. Get it working right once, on one machine, and it will (more than likely) keep right on working, and you can usually use the same method on all your other machines, old and new.

Let's look at backups again. No one wants to type the "runbackup" command every day. And it's a sure thing that no one wants to have to type nonsense like

```
% dump 3bfu 32 /dev/rmt2 / /usr /var
```

day after day after day. Get the best backup hardware and software your budget allows. If your budget is zero, at least use something like (the terrific) "amanda" backup software from the University of Maryland [1]. If your budget is larger than zero, buy expensive backup software, buy a giant jukebox, install fiber to a suitable off-site location, set it, and forget it (well, not really, but you get the idea) [2]. The idea is, of course, to make computers do the repetitive tasks – computers are good at mindless repetition, and humans generally aren't.

. . . the favorite metric for how much documentation is enough is the "if you got hit by a bus" rule – the question being whether your systems would keep on running if you got hit by a bus tomorrow.

Making tasks repeatable usually requires a larger up-front investment of time and energy.

We've all seen Makefiles (or at least, most of us have). If you think about it, `make` [3] is one of the earliest automation tools for UNIX, and it can be used for all sorts of things, in addition to building programs from source [4]. Shell scripts combined with `cron` are another easy way to automate repetitive tasks.

My final example in my argument in support of automation is its use in software distribution. One of the few recurring themes in the myriad of software distribution papers in past LISA conferences is that of automation. Updates almost always get distributed automatically, usually fired off early in the morning from a `cron` job. People can be far more effective when they have the same software environment everywhere, and, conversely, system administrators can be far less effective if they need to copy, compile, and install the same software on each machine any time something new or updated is installed.

Repeatability

There are two ways to think of repeatability – how to redo or repeat tasks on a single machine and how to apply (or repeat) the same actions across multiple machines.

Using our favorite example (backups), repeatability means that your backups can run the same way, day after day, with as little manual intervention as possible. This includes little things like making sure your log files get rolled from time to time so they don't just grow without bound and fill your disk. Repeatability also means that you can install the software and get it working again after an OS upgrade (or recovery), and it also means that once you have your backups working on one machine, you can easily get them working on all your other machines.

Making tasks repeatable usually requires a larger up-front investment of time and energy. You need to think about and write things like install scripts, Makefiles, setup scripts, and so on. In software development, we talk about the development cost being only a small part of the total cost and the bulk of the cost being in the ongoing support and maintenance of the software. I'll claim that system administration often has an analogous rule – that the bulk of the cost of system administration is often the ongoing and repetitive tasks that stretch forward through the years. But I'll also claim that the future costs can be reduced (often substantially) by making a modest investment up front in ensuring repeatability.

Two more quick examples of tasks that really benefit from repeatability: account creation (especially in an educational setting or anywhere else with high user turnover) and PC and/or workstation setup (typically through software automation, disk cloning, or custom boot disks). Very few of us look forward to creating the ten thousandth user account or setting up the fifteen hundredth "wintel" PC by hand. (Yes, repeatability is often very closely tied to automation.)

Odds and Sods

Now that we've gone over what I claim are the key words and basic ideas, I'll quickly mention a few more items that don't seem to fall obviously under one of the key words.

Use BOOTP and/or DHCP servers to dole out IP addresses and distribute DNS, gateway, and other configuration information. Even if you statically allocate IP addresses to all your PCs, X terminals, etc., the ease of setup and the ability to renumber or reconfigure easily and automatically when necessary are great ways to increase reliability (and your free time) by avoiding having to visit each desktop or having to track down which two machines are trying to use the same IP address.

Avoid manual or custom setups whenever possible. Use a standard workstation software setup, and discourage (or prevent) users from changing or adding things.

Be familiar with tools like `rdist` and `track` to automate your file distribution – they're a great way to automatically replace or repair files that have become corrupted or gone missing (as long as it's not your master host that has problems!).

Use a change control system, like RCS, SCCS, or CVS, whenever possible, and always put something meaningful in the change logs. It's a great way to keep track of what you've changed and an easy way to keep old versions available (and much more reliable than a series of `.bak` files).

Set up as much automatic monitoring as possible. Use a log file watcher to monitor syslog and other log files and dispatch warning messages in the appropriate ways (email, zephyrgrams, broadcasts, pagers, voice modems, cell phones, etc.). Or even better, have your log file watcher fire off repair scripts to fix things automatically. And you should also periodically run commands or scripts to watch things like mail and printer queues, free disk space, load average, and so on. (It's always nice to notice and repair a problem before the users notice and start calling.)

Use a `cron` job to save copies of important files (like `/etc/passwd`) to help guard against errors, accidental deletions, and filesystem corruption.

And to help you react appropriately, make sure you have the set of tools that help you do your job in the most appropriate way – tools like laptops, network connections at home, cell phones, and pagers can be annoying, but if they save you a late-night trip into work, you might be better off.

For Another Time

There are some topics that could have been included in this article but are important or large enough to merit separate discussion. In future articles, I hope to cover backups (in much deeper detail), restores, and disaster recovery and discuss how your "people practices" – training, communication, coordination, who's on call, etc. – fit into your approach to reliability, and, of course, the perennial favorite, security, and a review of how your security policies and practices affect the reliability of your systems.

Finally, if there are any reliability-related topics that I haven't covered (thereby demonstrating that I may not be as reliable a source of information as one might hope), please let me know. I'd appreciate your input.

Notes

[1] <ftp://ftp.cs.umd.edu/pub/amanda/> contains everything about Amanda, including copies of "The Amanda Network Backup Manager" by James da Silva and Ólafur Gudmundsson from LISA VII, 1993, and "Performance of a Parallel Network Backup Manager" by da Silva, Gudmundsson, and Daniel Mossé from the 1992 Summer USENIX Technical Conference.

[2] This kind of approach actually works great, by the way. Details on request.

[3] S. I. Feldman, "Make – A Program for Maintaining Computer Programs," 1978. Available at <http://plan9.bell-labs.com/7thEdMan/vol2/make>.

[4] See, for example, "'Make' as a System Administration Tool" by Bjorn Satdeva in the SANS III proceedings from 1994.