treats users as customers to create respectful, understanding partnerships for resolving problems. The key to job satisfaction, feeling better, and getting better raises is to make the transition from clerk to advocate.

Making a successful transition to system advocate requires converting bad (subservient) habits into good (cooperative) ones, creating spare time for better communication and quality time for planning and research, and automating mundane and repetitive tasks.

Although changing habits is always hard, it's important to concentrate on getting a single success. Follow that with another and another, and over time these experiences will accumulate and become the foundation for good habits.

prioritize your tasks and remove extraneous activities from your workload.

Automating tasks, within the realm of a system administrator, requires competency in programming languages such as Perl, Awk, and MAKE. These are languages that have proven to be robust and provide the functionality that is necessary to automate complex tasks.

Transforming the role of clerk to advocate is hard and requires a change in attitude and working style to improve the quality of work life, provide more value to customers, and create a more professional and rewarding environment. However, the effort required to make this transition is worth it. Simply put, vendors can automate the clerical side of system administration, but no vendor can automate the value of a system advocate.

Helen Harrison and Tim Gassaway

To find spare time, people need to think outside of the box and be more critical and selective about where their time is spent. Suggestions for regaining time include stop reading USENET, get off mailing lists, take a time management course, filter mail (and just delete the ones that you can't get to in a reasonable time – e.g., at the end of the week), and meet with your boss (or key customer) to

How to Control and Manage Change in a Commercial Data Center Without Losing Your Mind

Sally J. Howden and Frank B. Northrup, Distributed Computing Consultants Inc.

Howden and Northrup presented a methodology to ensure rigor and control over changes to a customer's computing environment. They (strongly) believe that the vast majority of problems created today are caused by change. When change occurs unsuccessfully, the result can range from lost productivity to financial loss. Change is defined as any action that has the potential to change the environment and must consider the impact from software, hardware, and people. Using the rigorous method that was outlined will lower the overall risk and time spent on problems. They believe that this rigor is required for all changes, not just for significant or complex ones.

There are eight main steps outlined in this methodology: (1) Establish and document a base line for the entire environment. (2) Understand the characteristics of the change. (3) Test the changes in both an informal test and formal preproduction environment. (4) Fully document the change before, during, and after implementation. (5) Review the change with all involved parties before placing it into the production environment. (6) Define a detailed back-out strategy if the change fails in the production environment. (7) Provide training and education for all parties involved in the change. (8) Periodically revisit the roles and responsibilities associated with the change.

The authors were quite firm about testing a change in three physically distinct and separate environments. The first phase includes (unit) testing of the change on the host(s) involved in development. The second phase requires testing in a preproduction environment that, in the best case, is an exact duplicate of the production environment. The third phase is placing the change in the actual production environment.

When pressed on the suitability of using this (heavyweight) process on all changes, the authors stated that the highest priority activities are to fully document change logs and to create thorough work plans. The paper notes, however, that although this process does generate a significant

amount of work by the administrators before a given change, it has (over time) shown to reduce the overall time spent – especially for repeated tasks, when transferring information to other staff, when secondary staff are on duty, and when diagnosing problems.

Session: System Design Perspectives

Summaries by Mark K. Mellis

Developing Interim Systems

Jennifer Caetta, NASA Jet Propulsion Laboratory

Caetta addressed the opportunities presented by building systems in the real world and keeping them running in the face of budgetary challenges.

She discussed the role of interim systems in a computing environment – systems that bridge the gap between today's operational necessities and the upgrades that are due three years from now. She presented the principles behind her system design philosophy, including her extensions to the existing body of work in the area. Supporting the more academic discourse are a number of cogent examples from her work supporting the Radio Science Systems Group at JPL. I especially enjoyed her description of interfacing a legacy stand-alone DSP to a SparcStation 5 via the DSP's console serial port that exposed the original programmer's assumption that no one would type more than 1,024 commands at the console without rebooting.

Caetta described points to consider when evaluating potential interim systems projects, leveraging projects to provide options when the promised replacement system is delayed or canceled, and truly creative strategies for financing system development.

A Large Scale Data Warehouse Application Case Study

Dan Pollack, America Online

Pollack described the design and implementation of a greater-than-one-terabyte data warehouse used by his organization for decision support. He addressed such issues as sizing, tuning, backups, performance tradeoffs and day-to-day operations.

He presented in a straightforward manner the problems faced by truly large computing systems: terabytes of disk, gigabytes of RAM, double-digit numbers of CPUs, 50 Mbyte/sec backup rates – all in a single system. America Online has more than nine million customers, and when you keep even a little bit of data on each of them, it adds up fast. When you manipulate that data, it is always computationally expensive.

The bulk of the presentation discussed the design of the mass storage IO subsystem, detailing various RAID configurations, controller contention factors, backup issues, and nearline storage of "dormant" data sets. It was a fascinating examination of how to balance the requirements of data availability, raw throughput, and the state of the art in UNIX computation systems. He also described the compromises made in the system design to allow for manageable system administration. For instance, if AOL strictly followed the database vendor's recommendations, they would have needed to use several hundred file systems to house their data set. By judicious use of very large file systems so as to avoid disk and controller contention, they were able to use a few large (!) file systems and stripe the two gigabyte data files across multiple spindles, thereby preserving both system performance and their own sanity.

Shuse At Two: Multi-Host Account Administration

Henry Spencer, SP Systems

Spencer's presentation described his experiences in implementing and maintaining the Shuse system he first described at LISA '96. He details the adaptation of Shuse to support a wholesale ISP business and its further evolution at its original home, Sheridan College, and imparted further software engineering and system design wisdom.

Shuse is a multi-host administration system for managing user accounts in large user communities, into the tens of thousands of users. It uses a centralized architecture. It is written almost entirely in the expect language. (There are only about one hundred lines of C in the system.) Shuse was initially deployed at Sheridan College in 1995.

Perhaps the most significant force acting on Shuse was its adaptation for ISP use. Spencer described the changes needed, such as a distributed account maintenance UI, and reflected that along with exposing Sheridan-specific assumptions, the exercise also revealed unanticipated synergy, with features requested by the ISP being adopted by Sheridan.

A principal area of improvement has been in generalizing useful facilities. Spencer observed in his paper, "Every time we've put effort into cleaning up and generalizing Shuse's innards, we've regretted not doing it sooner. Many things have become easier this way; many of the remaining internal nuisances are concentrated in areas which haven't had such an overhaul lately."

Other improvements have been in eliminating shared knowledge by making data-transmission formats self-describing, and in the ripping out of "bright ideas" that turned out to be dim and replacing them with simpler approaches. These efforts have payed off handsomely by making later changes easier.

Spencer went on to describe changes in the administrative interfaces of Shuse, and in its error recovery and reporting.

Shuse is still not available to the general public, but Spencer encourages those who might be interested in using Shuse to contact him at <henry@zoo.toronto.edu>

Spencer's paper is the second in what I hope will become a series on Shuse. As a system designer and implementor myself, I look forward to objective presentations of experiences with computing systems. It's a real treat when I can follow the growth of a system and learn how it has changed in response to real-world pressures and constraints. Often papers describe a system that has just been deployed or is in the process of being deployed; it is rare to see how that system has grown and what the development team has learned from it.

Session: Works in Progress

Summaries by Bruce Alan Wynn

Service Level Monitoring

Jim Trocki, Transmeta Corp.

Many system and network administrators have developed their own simple tools for automating system monitoring. The problem, proposes Jim Trocki, is that these tools often evolve into something unlike the original and in fact are not "designed" at all.

Instead, Jim presents us with mon: a Perl 5 utility, developed on Linux and tested on Solaris. mon attempts to solve 85% of the typical monitoring problems. The authors developed mon based upon these guidelines:

- Simple works best.
- Separate testing code from alert generation code.
- Status must be tracked over time.

The mon tool accepts input from external events and "monitors" (programs that test conditions and return a true/false value). The mon processes then examine these data and decide which should be presented directly to clients and which should trigger an alarm.

The authors are currently expanding the functionality of mon to include dependency checking of events, severity escalation, alert acknowledgments via the client, "I'm okay now" events, asynchronous events, a Web interface, and a better name.

The current version of mon is available at <http://consult.ml.org/~trockij/mon>.
Jim Trocki can be reached at <trockij@transmeta.com>.

License Management: LICCNTL – Control License Protected Software Tools Conveniently

Wilfried Gaensheimer, Siemens AG

Gaensheimer presented an overview of a number of tools that can help control and monitor the use of software licenses. The tools can also generate reports of license use over time.

For additional information on these tools, contact Gaensheimer at <wig@HL.Siemens.DE>.

Inventory Control

Todd Williams, MacNeal-Schwendler Corp.

One of the less exciting tasks that system and network administrators are often faced with is that of taking a physical inventory. Typical reasons for this requirement include:

- Maintenance contract renewal
- Charge backs for resource use
- Identifying the type of machinery

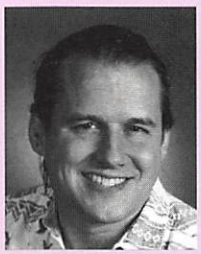
Williams began tracking his inventory by including comments in the system's "hosts" files, but quickly outgrew this

Figure 2: A simple modification of newsyslog script

```
#!/bin/sh

# location of audit log files
cd /etc/security/audit
# Tar all files matching the pattern, not Y2K compliant :)
/usr/sbin/tar cf tarfile 19*.199*
# then remove the individual file to save space
/bin/rm 19*.199*

LOG=tarfile
test -f $LOG.Z.6 && mv $LOG.Z.6 $LOG.Z.7
test -f $LOG.Z.5 && mv $LOG.Z.5 $LOG.Z.6
test -f $LOG.Z.4 && mv $LOG.Z.4 $LOG.Z.5
test -f $LOG.Z.3 && mv $LOG.Z.3 $LOG.Z.4
test -f $LOG.Z.2 && mv $LOG.Z.2 $LOG.Z.3
test -f $LOG.Z.1 && mv $LOG.Z.1 $LOG.Z.2
test -f $LOG.Z.0 && mv $LOG.Z.0 $LOG.Z.1
mv $LOG.Z  $LOG.Z.0
# compress the new tar file to save space
/usr/bin/compress tarfile
```



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>

On Reliability – What About Yourself?

In past articles on reliability, I've talked about general principles of reliability, computing hardware, networking, and some aspects of system administration. Most of those things are really quite tangible – if you can't put your hands on them physically, you can at least copy them to a printer or a tape drive and hold them in your hands that way.

Since I wrote the last article (for the December issue, publishing deadlines being what they are), I've been to the 11th LISA conference in San Diego, where we spent a lot of time (more than usual) talking about management, motivation, and people issues. Since returning home, I've found myself doing some reading on management and people and thinking more about the people issues that we face in our jobs (and other activities). And I've spent a heck of a lot of time in meetings, working with people, and thinking about motivation, coordination, and how people can really enjoy their work.

So I find myself here with my laptop on my daily commute on the intercity bus and with the Christmas holidays and a new year looming up before me, composing a reliability article with a different flavor this month. I'm compelled to consider, from a purely amateur point of view, personal reliability. By that I mean to consider how we interact with our co-workers, vendors, customers, and, to a lesser extent, friends and families. How does one act "reliably"?

How is this relevant to system administrators and computing professionals in general? How does this help to make our computer systems and networks run better and more effectively? System administration is very closely tied to personal interaction, with individuals and with groups, and sometimes with people that you will never see or talk to

directly. I'll try to give a few examples of why I think that is the case and why reliability and trust are important.

System administration is a service activity – we supply the computing resources so that other people can do their work (or play). We solve problems for people, we design systems and software to serve people, and we help people learn to accomplish their computing tasks in the most effective ways. Any time we install a new command, send out a notice or advisory message, or answer the phone on the help desk, the underlying end product is (almost always) a service for some person or group. When we take a system down for maintenance, submit a request for more funding for more equipment, design a mission-critical computing environment, start fixing a computer or network problem, or propose a solution to suit someone's needs, we're asking for trust: trust that we are using good judgment, trust that we are knowledgeable and competent, and trust that our intentions are good. In short (and I'm sure you've been waiting for this), we are asking others to rely on us. And that's where reliability comes into things this time around.

Why is it important to be reliable? Quite simply, if we are to call ourselves "professionals," we must rely on our reputations, and the most important part of a (positive) reputation is the trust that people can place in us, our judgment, and our abilities. If we cannot be relied upon, all of our experience and abilities will be far less valuable to our customers and co-workers. The ability of others to rely on us is the foundation of the value that we bring to the profession of system administration.

How do you demonstrate your reliability? How do you earn the trust of your constituents? I think the most important piece of advice is to avoid the "us vs. them" mentality that we see (or hear about) all too often. Recognize that you and your users are (or should be) working toward the same goals and toward the success of your enterprise. Although the goals and needs of different groups sometimes seem to be at odds, a little goodwill and effort to understand will make it far easier to work together toward the best solutions.

Consider the other people in your organization, and work to understand their concerns and needs. System administration is not done in a vacuum – a system is only as worthwhile as the systems and solutions that it provides. A beautiful, carefully designed, "perfect" computing system is useless if it is conceptually pure but unsuited to solving the problems at hand.

When interacting with customers or others in your organization, be honest and open. If there's a problem, admit it; and if it's a result of something you did (or didn't do), own up to it, and take responsibility. Any short-term pain will be far outweighed by the long-term gain as your users trust and rely on you. Say what the problem is (or was) and what you've done to keep it from happening again. Give advance warning when you're about to change something, and be realistic about expected downtimes. And remember to follow through: do what you said you would do, when you said you would do it. And finally, be proactive: talk with your users, solicit their feedback and concerns, and act on them. Earn their trust, and you'll be far better off in the long run.

And if the word "lusers" is a part of your vocabulary, you might want to reconsider your use of it.

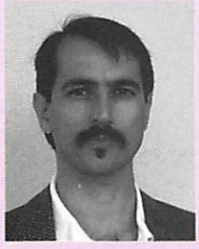
If you're a manager or leader of system administrators, can the people in your group rely on you? Are you supportive, understanding, fair? Do you send people home when they are sick, or do you tell them to "tough it out"? Are you an advocate for your co-workers? Do you defend them if they're being attacked (deservedly or not)? Do you

And if the word "lusers" is a part of your vocabulary, you might want to reconsider your use of it.

champion them in interactions with other groups and higher-ups? Do you fight for appropriate conference and training budgets and extra pay or comp time when they work overtime? Can the people in your group rely on you? And finally, allow me to offer some words from Dee Hock, founder and CEO emeritus of Visa: "If you don't understand that you work for your mislabeled 'subordinates,' then you know nothing of leadership. You know only tyranny."

When I started in system administration years ago, I spent a lot more time concentrating on my "relationship" with the machines. These days, I spend a lot less time dealing with machines and a lot more time dealing with the people who surround them. I'm starting to learn which of those relationships is the more complicated and the more rewarding and where the true value and the true satisfaction lies. (The machines really don't care whether I'm reliable or not, so long as I keep the AC power coming and the backup tapes loaded.)

Well, that's enough of that. I suspect that I've been "preaching to the choir" a little bit here. Next time I promise something a little more concrete that you can sink your teeth into: backups, restores, and disaster recovery.



by Daniel E. Singer

Dan has been doing a mix of programming and systems administration for 13 years. He is currently a systems administrator in the Duke University Department of Computer Science in Durham, North Carolina, USA.

<des@cs.duke.edu>

ToolMan: Upcoming Tools; Analyzing Paths

It's a new year, a new volume of *;login:*, perhaps time for new resolutions. A resolution I made late last year was to incorporate tools from other tool makers into ToolMan articles, primarily to keep the series more useful and interesting. Toward that goal I've included tools by two of my co-workers in the previous and current issues. I'll be writing about tools by people from the wider community in future articles, though I've found that working out the details in these situations takes much longer. But a few things are in the works, so please stay tuned.

Some topics I'm considering for future articles include:

email folder processing	RCS/SCCS wrappers
accounts	tar wrappers
netgroups	backups
disk quotas	directories/files
disk space	processes
finding	searching/replacing
comparing	sorting
text manipulation	printing
documenting	email alias parsing
remote execution	scheduling/calendar
Web (of course)	

In other words, the possibilities are wide open. Tools relating to these topics might be geared toward system administrators or general users. When possible, I'll survey several tools relating to a given topic. If you have any tools that fit this list or other categories that you would like me to include, please send a note.