Managing the Ever-Growing To Do List

Rémy Evard – Northeastern University

ABSTRACT

A system administrator's most important task is managing the list of user requests, work assignments, and active problems. If these items aren't prioritized and handled, issues can be forgotten or delayed, and important problems may go unsolved while immediate yet trivial problems get all the attention. In the best case, one will spend too much time working on the list of tasks instead of working on the tasks themselves.

This paper is an account of our experiences with tackling the problem of keeping track of tasks. We present a software system that we have developed and a methodology for using it to stay on top of the growing list of things to accomplish. We feel that our experiences may be of use to other system administration groups.

Introduction

Sometimes the day of a system administrator goes something like this: You come in bright and early, planning to finally finish that program you've been working on sporadically for the last month. You make the mistake of checking your mail, and see a pile of seemingly simple problems that have built up over night. About an hour later, the truly simple ones have been solved, and you've pushed the not-so-simple ones off until the afternoon. You pull out your program, but notice more mail has come in. You ignore it and start to code, only to be interrupted by the phone. You help out the poor confused user on the other end while poking around your office looking for materials for an upcoming meeting. Your manager stops you, inquires about your long-term projects, and asks you to check on a problem in the machine room. Fifteen minutes later, after fixing a jammed printer, you make it to the machine room, reboot the server, and check your mail while the server comes up. And so the day goes. Exhausted, you head home, knowing you got a lot done, but not knowing exactly what it was.

The point is that the system administrator's job consists of hundreds of tasks from many sources. Users have requests and questions. Managers assign projects and responsibilities. Problems appear from all over. And, perhaps most importantly, you have your own ideas and goals to accomplish.

It is critical to be able to organize all of these tasks. If they aren't handled in some reasonable fashion, then the simple things are taken care of first, while important (but complex) tasks go undone. Worse, some problems get completely forgotten about. The large blocks of time that are required to concentrate on difficult problems become scarce as interrupts become commonplace. As the list of undone tasks grows (if there is such a list), the overhead for keeping up with it takes more and more time. The problem only worsens as the number of users and the number of administrators grows.

This paper is an account of our experiences with tackling the problem of keeping track of tasks. We present a software system that we have developed and a methodology for using it to stay on top of the growing list of things to accomplish.

Site Information

The Experimental Systems Group manages the computing environment in the College of Computer Science of Northeastern University, consisting of approximately 350 computers of various types and around 1200 active users. The group is made up of both full-time staff members and student volunteers, totaling an average of 10 people each quarter.

A Contact Point

The first step in a solution is to create a well-known mechanism for the users to submit requests or problems. Whether or not such a mechanism exists, the problems will find their way to you, one way or another. It is to your advantage to choose what that route is. If you don't, you'll have some users visiting your office, some calling you, some emailing you and your manager, some paging you, and some calling you at home. Regardless of the actual method for reporting problems, the user population should be made aware of its existence and how to use it. By creating a method for reporting problems and telling people to use it, you'll limit the number of sources for problems, and you'll cut down on user confusion.

We use a single email alias for user problems, as do many sites. Users are told, repeatedly, to mail requests to ''systems''. Everyone on the Systems Group receives the mail. When someone replies, they send a copy of the reply to the list, so that everyone can read it and follow the conversation, should they so desire. Most people on the group filter mail to systems into a specific mail folder, making it easier for them to organize and track.

We use ''systems'' only for request-related mail. When we send mail to the other members of the group for information or discussion purposes, we use a different alias, which allows us to prioritize the mail differently, and organize user requests in one place.

We've thought several times about having more than one mailing list for the users to use. We could have ''systems'' for most problems, and ''macs'' for Macintosh-related problems. We have elected not to do this because, in our primarily student-based environment, we don't believe the users will categorize the mail correctly – if it's a network problem with Macs, where should it go? Furthermore, with just one alias, the instructions are almost simple enough for our constantly changing user population: ''send mail to systems if you have a problem.''

Training the users to send mail to systems is an ongoing effort. When the user's home directory is first created, they're given a README file that, among other things, tells them to send mail to 'systems' if there is a problem. We state this on our hardcopy documentation and on our news postings. And we say it to users in the hall and on the phone if their problem isn't an emergency. If they send mail to someone directly, we resend the mail to 'systems' and send them a canned response saying that ther mail should be sent to 'systems'. If they do it again, their mail takes a little longer to be resent...

We want mail to go to ''systems'' and not to individuals for several reasons. If the individual is gone, busy, or on vacation, no one else will know about the problem, much less be able to work on it. It's useful for others members of the group to know what's going on. Perhaps most importantly, the users are rarely correct in their guess as to who will to work on their problem. When they send mail to systems, we can pick and choose who will work on what, rather than letting the user target a specific individual.

We do handle requests from other sources when the need arises. The phone has an answering machine, and we run a help desk during certain hours for people who don't have accounts or can't use mail. And when someone in the hall has a problem, we'll try to solve it if it will take less than a minute or so.

Using a shared email address has several benefits:

- Users have one place to send things.
- Everyone reads the mail, so people are aware of ongoing issues.
- It's a great learning mechanism for junior members of the Systems Group.
- Nearly everyone can send email, and it's pretty simple, (unlike, for example, posting news or running a graphical interface), so it works for most users.
- Logging the mail for record keeping is simple.

Unfortunately, it has some real problems as well:

- Everyone reads the same mail, creating some serious overhead.
- Sometimes two or more people try to answer the same problem, wasting time and sending two (potentially confusing) answers to the user.
- Sometimes people assume someone else will answer, or forget to send a copy of their their reply back to systems.
- Everyone has their own mail queue, so everyone tracks problems to make sure they get solved. This translates into duplicated effort and confusion.
- Requests get lost or forgotten in the mail deluge.

Thus, a shared email address is a step in the right direction, but isn't a complete solution.

Failed Interim Solutions

The most critical failing of the simple email address was that it didn't keep track of requests. I, as manager of the group, spent enormous amounts of time monitoring the email queue, making sure that things got answered and that people were working on the important problems. I tried keeping lists of tasks that needed to be done on paper tablets, on white boards, and in ASCII files. I was always updating the lists and rechecking them. Those lists weren't easy to keep up-to-date, and weren't easily modifiable by everyone in the group.

In order to try to keep up with the incoming queue in real-time while making progress on long-term projects, we developed the concept of a ''hotseat'', which was intended to be occupied by a person who read systems mail, handling incoming requests and freeing time for others to work on problems requiring more concentration. This failed badly, because it wasn't possible to assign problems to other people or to record the status of a request. While the person on the hotseat did manage to trap interrupts, the next person on the hotseat spent much time duplicating the effort of the previous person.

For quite some time, we planned on developing a better solution than a simple address to keep track of incoming requests. When it became obvious that manually-maintained lists and hotseat organization weren't helping, and that the overhead in trying to keep track of the requests was substantial, we realized it was past the time to move to an automated tracking system.

Problem Tracking Systems

Many sites use automated tracking systems to keep track of their tasks. After assessing the way that we worked, we decided we needed a system that:

- Kept a database of unsolved requests.

- Attached a user, a priority, a request date, and a due date to a request.
- Let us assign an owner to a request.
- Allowed us to see the list of requests based on varying criteria.

Essentially, it needed be a tool to organize requests and help us choose what to work on next.

Many possible solutions exist, but we were unable to locate a system that fit our needs. Commercial solutions tend to be very expensive, as most require a powerful data base engine. Existing free implementations didn't quite work either. Some, like Queue-MH and PITS, required that the administrator use a specific mail program or tracking system interface. Others, most of which were based on the

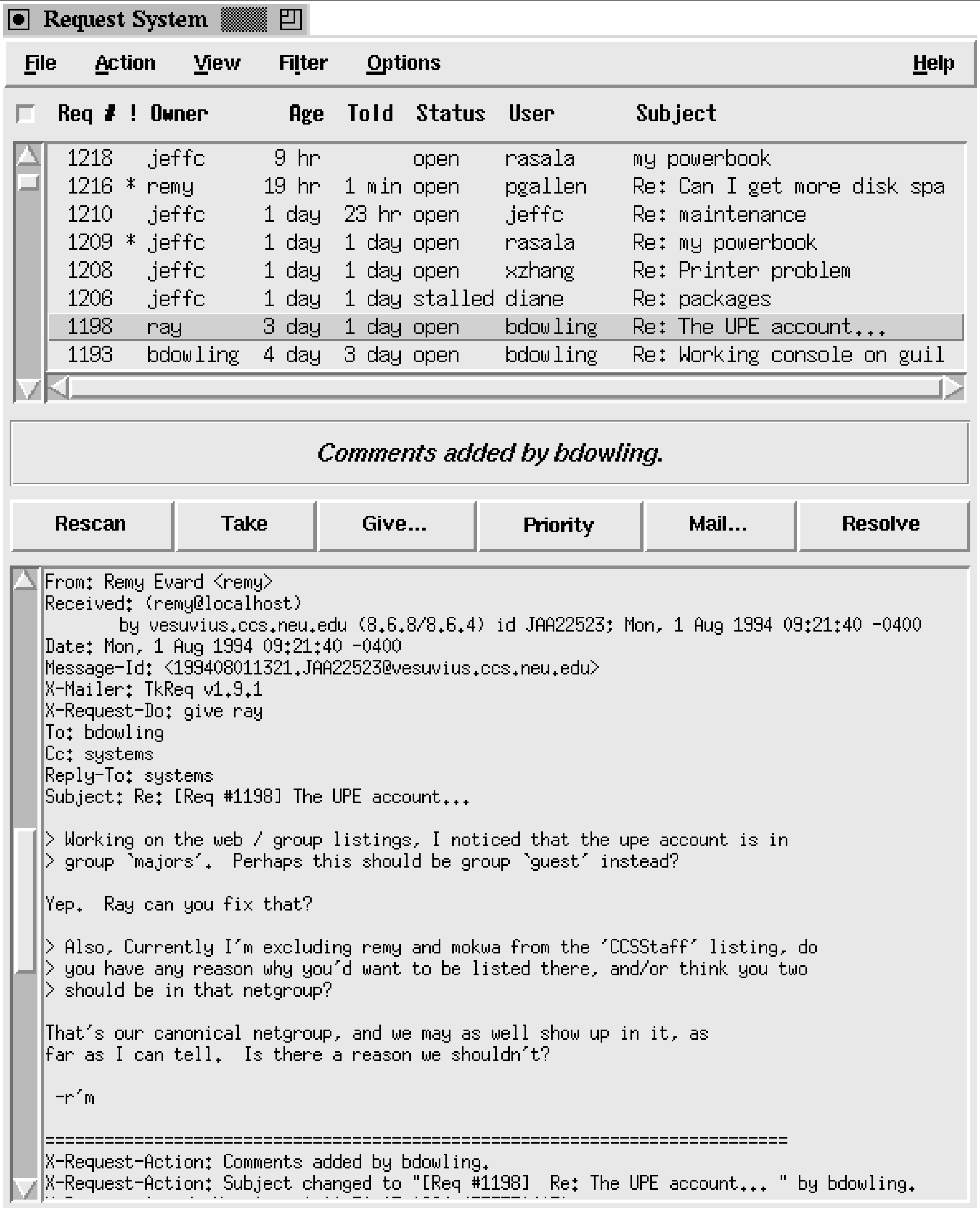

Figure 1: Example management interface screen

UNIX dbm library routines, weren't portable across all the UNIX platforms that we needed to run them on. And others, such as GNATS, just didn't fit our work model. A list of tracking systems that we tried is given in Appendix A. While they didn't fit our needs, they may well be appropriate for yours.

Since we couldn't locate a solution that we thought would work for us, we developed our own.

Our Solution: Req

Req, which is pronounced like ''wreck'', not ''reek'' (although neither is particularly complimentary), was designed to integrate with the way that we already used mail. It consists of two main parts.

The first part is an email filter. All mail that is sent to ''systems'' is assigned a unique number and stored in a file. The number is inserted into the subject line, and then the message is passed on to the members of the Systems Group.

Suppose a user sent mail with this subject line:

```
Subject: Help! How do I send mail?
```

Everyone on the mailing list would receive this mail:

```
Subject: [Req #1837] Help! How do I send mail?
```

The filter checks the subject line before inserting a new number. If an old number already exists, the message is appended to the previous file related to that request number. In this way, a log of all mail associated with a number (and therefore with a particular problem) is created. The request log is kept in RFC-822 format, with special headers indicating the owner, the priority, and so on. Thus, the log looks very much like a conventional Internet email message, and can be used as such if necessary.

The second part of the system is a management interface. It currently may be accessed on a UNIX command line or via an X Window tool; see Figure One. An interface for emacs and for Macintoshes are under development. The management interface displays the following information:

- The request number.
- The priority as assigned by the Systems Group.
- The person working on the request, or the ''owner''.
- How old the request is.
- How long it's been since the user received a reply about the problem.
- The status: ''stalled'', ''open'', etc.
- Who made the request.
- What the request is about.

While the primary role of the request system is to keep track of requests, the major use of it by administrators is to help one look at the queue of requests and decide what to work on. The allows one to scan the items that he or she owns or that are currently unowned, choosing which request to work on based on priority, length of time in the queue, or whim.

Using the interface, an authorized user may:

- Review a request.
- Send mail to the user about the request.
- Take ownership of problems, or assign them to others.
- Prioritize items.
- Merge related items.
- Change their status.
- Add comments to a file.
- Resolve a request.
- Browse the active requests or the resolved requests.

Almost all of these functions can also be performed via email, by sending a message with the request number in the subject line and a command in the message header. For example, if the line:

```
X-Request-Do: give dave
```

appears in the header, then the owner of the request will be changed to ''dave'', and an entry will be made in the log of that request noting the change and who made it.

We designed the system to be as free of policy as we could, in order to provide maximum flexibility while we tested it. Some policy decisions, such as the number of priority levels, were encoded by necessity. Others, such as who may gave a request to whom, were left open.

Additional Features

Req also has a few features that, while not essential to the purpose, we have found to be quite useful.

- Common answers. The X interface to req has options to include and to make files of frequently asked questions. We keep these comprehensive answers in a directory, and use them whenever someone asks one of those questions.
- User access. Users may run an interface to req that shows them the status of their own requests. They can read the log of their own requests, seeing who has been working on it and checking on the current state of their request.
- Statistical analysis. We have programs that analyze the request queue, showing us trends in the requests, such as busy times of the quarter, average length to resolve a request, and number of requests per user.

Usage

Over time, we've settled into a usage pattern with req. We still tell users to mail their requests to ''systems''. Each request is assigned a number, entered into the system, and is sent on to the group.

We revived the ''hotseat'' idea once req was in place. The role of the person on the hotseat is to work with the req interface, keeping an eye on incoming problems and acting as a buffer for the other members of the group. If a new problem can be handled in 15 minutes or less, the person on the hotseat works on it immediately. If it can't be, the hotseat person gives the request to some other member of the systems group, who is notified by mail. The person on hotseat also answers the phone and sits at the help desk during help desk hours. Essentially, this person acts as the only interface to user problems, shielding the other members of the group from interruptions while giving quick feedback to simple user requests. When this person isn't handling interruptions, he or she works on the request queue, making sure that all the problems are owned and making progress on simpler requests. We trade off hotseat duty – each group member is on the hotseat at most one day a week.

The other members of the group work on longer-term projects. They use the req interface to look at their queues, choosing what to work on based on priority. Because the hotseat person is acting as a shield, others can often put in two or three hours of solid work on a problem, rather than being continually interrupted.

As the manager, I occasionally take the time to look over the whole queue, making sure that I agree with the priorities assigned to items, ensuring that progress is being made on the requests, and assigning some jobs to people who have less work than others.

Observations

Using a problem tracking system has made it considerably easier to prioritize our time and to keep track of requests. Much less time is spent keeping up with the queue, therefore more time is spent making progress on the items themselves.

We haven't lost any user requests since we installed req. We do have problems that have been in the queue for four months, but those are low priority items. The associated users have been made aware that we won't forget about the problem and we'll get to it when the time is right.

About one week into using req, we made the decision to use it to keep track of everything we needed to do, not simply limiting it to user-related requests. So we put our own tasks in the queue, adding, for example, the list of software we wanted to install, a number of hardware repairs, and everything else that we had been keeping track of on other lists. We had thought briefly about using req to manage another mailing list, but that would have meant that we would have had two queues to look at and choose from, not one. This decision has had an extremely positive effect: instead of trying to remember to do things, we simply send mail to systems, and let req act as our memory device.

While the req system was designed to be relatively policy-free, policies are definitely important. For example, we made the policy early on that anyone could give any request to anyone else. To give a request to someone means that you think they'll do a better job than you will, not that you think they should do it. This has cut down on the political issues related to being on the hotseat and giving incoming requests to your co-workers or your boss.

Our solution is a bit odd, in that group members see both the mail sent to the systems mailing list and the request in the req system. For the most part, we use the mail as a way of watching what's going on and a convenient way of quickly replying to something, while we use req for keeping track of items and deciding priorities. People actively delete the mail in their systems mailbox now, as opposed to keeping it around forever as they did before req. We will probably move to a system where no one gets any systems mail, but we're reluctant to lose the communication and education functions of the mailing list.

Problems Left

The request system has, for the most part, solved the problems that we had with incoming requests. However, a few problems related to time management still remain.

We need to learn how to manage large-scale projects. The request system is great for keeping track of small items, like a request to install software or fix a printer, but it's not quite appropriate for planning next year's networking infrastructure. When one is in charge of a big project and has lots of small things to do, it's easy to get stuck on the small things while ignoring the big thing. Our current solution is to create a request item for the project and use it to log progress on the project. This may or may not work – we don't know yet.

While we have merged the systems lists into one location, people still have individual to-do lists of their own, including items like meeting schedules, phone calls to return, and so on. We would like to integrate these types of lists into the overall solution, so that it is possible to tell in one glance what one should be focusing on next.

Critical Points in a Solution

We have built a system that helps us keep track of user requests and systems administrator tasks. With it, we are able to respond quickly to user requests while still putting concentrated time into long-term projects. The important points of our solution include:

- Letting the users know how to submit requests.

- Keeping track of those requests.
- Organizing the requests in ways that allow us to prioritize them.
- Designing policies and procedures to maximize response time and concentration time.

This solution fits our work model, which is based on an email paradigm and a relatively small group of administrators. The solution that works for you may or may not be similar to ours, and should fit your own work model.

Availability

The req software was designed to be installed outside of our environment. Req is built in C and perl, while Tkreq, the X interface to req, is written in Tk/Tcl. These packages, as well as documentation that goes into much more detail about req than this paper, is available at ftp.ccs.neu.edu in the file /pub/sysadmin.

Acknowledgements

In a flash of overnight inspiration, Robert Leslie wrote Tkreq, The X interface to the req system. For weeks thereafter, he was badgered into adding features to it.

Other members of the Systems Group, including Lauren Burka, Brian Dowling, Geoff Hulten, Ivan Judson, Shane Kilmon, Dave Kormann, Ray Matthieu, Jim Mokwa, and Matthew Wojcik, were essential in the design phase of Req, and coped with my endless experimentation and questioning once it was operational.

Author Information

Rémy Evard is the leader of the Experimental Systems Group at Northeastern University, where he has been for two busy years. He received his M.S. in computer science from the University of Oregon in 1992, where he worked as a graduate student systems administrator. His current research interests include distributed virtual environments and automation of systems administration.

Bibliography

Tinsley Galyean, Trent Hein, and Evi Nemeth, Trouble-MH, A Work-Queue Management Package for a >3 Ring Circus, in LISA IV, pp 93-96, Colorado Springs, Co, 1990.

William Howell, Managing In The 90s: Meeting The Challenge, July 1992. Presented at: SANS-I, Washington, D.C., July 1992, UNC-CAUSE, Asheville, NC, October 1992; International Help Desk Conference, Orlando, FL, February 1993; NC Help Desk Chapter, Greensboro, NC, March 1993

David Koblas & Paul M. Moriarty, PITS: A Request Management System, in LISA VI, pp 197-202, Long Beach, CA, 1992.

RFC 1297, NOC Internal Integrated Trouble Ticket System; Functional Specification Wishlist.

James M. Sharp, Request: A Tool For Training New Sys Admins and Managing Old Ones, in LISA VI, pp. 69-72, Long Beach, CA, 1992.

Appendix A

This appendix lists the non-commercial problem tracking systems which we discovered or evaluated. They are presented here in the hopes that they may be useful to others. No attempt to compare them has been made, and there are systems that are not on this list that were unaware of or unable to locate.

Most of these tools, as well as comments about them, may be found at ftp.ccs.neu.edu in the file /pub/sysadmin/tracking.

- GNATS, The Gnu Problem Report Management System, available on prep.ai.mit.edu as /pub/gnu/gnats-3.2.tar.gz. GNATS is oriented towards bug report tracking, but can be used for systems administration. GNATS has a Tk/Tcl based front end called tkgnats, which is available in the GNATS contrib directory.
- The NEARnet Trouble Ticket System, available on ftp.near.net as /pub/nearnet-ticket-system-v1.3.tar. It is built on an Informix Relational Database, and uses Embedded-SQL on top of MMDF.
- NETLOG, the JvNCnet trouble ticketing system, is available via anonymous ftp from ftp.jvnc.net as /pub/netlog-tt.tar.Z. It runs on UNIX systems and does not use a database.
- Queue MH, available on ftp.cs.colorado.edu as /pub/sysadmin/utilities/queuemh.tar.Z, is a set of scripts built around the UNIX MH mail system.
- PTS/Xpts, on ftp.x.org as /contrib/pts* is an X Windows based problem tracking system for both users and administrators.
- Request, a Task Tracking Tool, is on pearl.s1.gov as /pub/request/request2.1.1.tar.Z. It is written in perl and uses dbm libraries, providing an ASCII interface for problem submission and management.
- Requete, available on ftp.crim.ca in /pub/requete-*.tar.Z is still under development. It has an X interface.