The Amateur Computerist

Webpage: http://www.ais.org/~jrh/acn/

Winter/Spring 1994 **Celebrating 25 Years of UNIX** **Volume 6 No. 1**

"I believe all significant software movements start at the grassroots level. UNIX, after all, was not developed by the President of AT&T."
Kouichi Kishida, *UNIX Review*, Feb., 1987

Table of Contents

UNIX and Computer Science

by Ronda Hauben

[Editor's Note: This year, 1994, is the 25th anniversary of the invention of UNIX in 1969 at Bell Labs. The following is from a "Work In Progress" introduced at the USENIX Summer 1993 Conference in Cincinnati, Ohio. This article is intended as a contribution to a discussion about the significance of the UNIX breakthrough and the lessons to be learned from it for making the next step forward.]

The Multics collaboration (1964-1968) had been created to "show that general-purpose, multiuser, timesharing systems were viable." Based on the results of research gained at MIT using the MIT Compatible Time-Sharing System (CTSS), AT&T and GE agreed to work with MIT to build a "new hardware, a new operating system, a new file system, and a new user interface." Though the project proceeded slowly and it took years to develop Multics, Doug Comer, a Professor of Computer Science at Purdue University, explains that "fundamental issues were uncovered" in the process of the research on Multics, "new approaches were explored and new mechanisms were invented." The most important, he explains, was that "participants and observers alike became devoted to a new form of computing (the interactive, multiuser, timesharing system). As a result, the Multics project dominated computer systems research for many years, and many of its results are still considered seminal."[1]

By 1969, however, AT&T made a decision to withdraw from the project. Describing that period, Dennis Ritchie, one of the inventors of UNIX at Bell Labs writes, "By 1969, Bell Labs management, and even the researchers came to believe that the promises of Multics could be fulfilled only too late and too expensively."[2]

"Even before the GE-645 Multics machine was removed from the premises," Ritchie explains, "an informal group led primarily by Ken Thompson, had begun investigating alternatives."

Thompson and Ritchie presented Bell Labs with proposals to buy a computer so they could build an interactive, time sharing operating system for it. Their proposals weren't acted on. Eventually, Ken Thompson found a little used and obsolete PDP-7 computer, a tiny machine in the class of a Commodore 64 computer.

The environment Thompson was attempting, explains Ritchie, included "many of the innovative aspects of Multics," such as "an explicit notion of a process as a locus of control, a tree-structured file system, a command interpreter as a user-level program, simple representation of text files, and generalized access to devices."[3]

Describing the primitive conditions that Thompson faced when attempting to create his desired programming environment, Ritchie writes, "At the start, Thompson did not even program on the PDP itself, but in-

stead used a set of macros for the GEMAP assembler on a GE-635 machine. A postprocessor generated a paper tape readable by the PDP-7. These tapes were carried from the GE machine to the PDP-7 for testing until a primitive UNIX kernel, an editor, an assembler, a simple shell (command interpreter), and a few utilities (like the UNIX rm, cat, cp commands) were completed. At this point, the operating system was self-supporting; programs could be written and tested without resort to paper tape, and development continued on the PDP-7 itself."[4]

The result, Ritchie explains, was that "Thompson's PDP-7 assembler outdid even DEC's in simplicity; it evaluated expressions and emitted the corresponding bits. There were no libraries, no loader or link editor: the entire source of a program was presented to the assembler, and the output file – with a fixed name – that emerged was directly executable."[5]

The operating system was named UNIX, to distinguish it from MULTICS.

As work continued to create this new operating system, the researchers developed a set of principles to guide their work. Among these principles were:

"(i) Make each program do one thing well. To do a new job, build afresh rather than complicate old programs by adding new features.

(ii) Expect the output of every program to become the input to another, as yet unknown, program. Don't clutter output with extraneous information. Avoid stringently columnar or binary input formats. Don't insist on interactive input.

(iii) Design and build software, even operating systems, to be tried early, ideally within weeks. Don't hesitate to throw away the clumsy parts and rebuild them.

(iv) Use tools in preference to unskilled help to lighten a programming task, even if you have to detour to build the tools and expect to throw some of them out after you've finished using them."[6]

By 1970, Ritchie writes, the UNIX researchers were "able to acquire a new DEC PDP-11. The processor," he remembers, "was among the first of its line delivered by DEC, and three months passed before its disk arrived."[7] Soon after the machine's arrival and while "still waiting for the disk, Thompson," Ritchie recalls, "re-coded the UNIX kernel and some basic commands in PDP assembly language. Of the 24 Kbytes of

memory on the machine, the earliest PDP-11 UNIX system used 12 Kbytes for the operating system, a tiny space for user programs, and the remainder as a RAM disk."[8]

"By early 1973," Ritchie explains, "the essentials of modern C were complete. The language and compiler were strong enough to permit us to rewrite the kernel for the PDP-11 in C during the summer of that year."[9]

Each program they built developed some simple capability and they called that program a tool. They wanted the programs to be inviting to use and to be helpful to programmers. Describing the achievements of the lab, Doug McIlroy, one of the researchers and Thompson's Department Head when he created the UNIX kernel, describes the atmosphere at the lab: "Constant discussions honed the system.... Should tools usually accept output file names? How to handle de-mountable media? How to manipulate addresses in a higher level language? How to minimize the information deducible from a rejected login? Peer pressure and simple pride in workmanship caused gobs of code to be rewritten or discarded as better or more basic ideas emerged. Professional rivalry and protection of turf were practically unknown: so many good things were happening that nobody needed to be proprietary about innovations."[10]

The research done at the Labs was concerned with using the computer to automate programming tasks, such as the ones needed to program and operate the newly installed electronic telephone switches. By a scientific approach to their work and careful attention to detail, Bell Labs researchers determined the essential elements in a design and then created a program to do as simple a function as possible. These simple computer automation tools would then be combined to build programs to do more complicated tasks.

They created a UNIX kernel accompanied by a toolbox of programs that could be used by others at Bell Labs and the Bell System. The kernel consisted of about 11,000 lines of code. "The kernel," Ken Thompson writes, "is the only UNIX code that cannot be substituted by a user to his own liking. For this reason, the kernel should make as few real decisions as possible."[11]

Thompson describes creating the kernel: "What is or is not implemented in the kernel represents both a great responsibility and a great

power. It is a soap-box platform on 'the way things should be done.' Even so, if 'the way' is too radical, no one will follow it. Every important decision was weighed carefully. Throughout, simplicity has been substituted for efficiency. Complex algorithms are used only if their complexity can be localized."[12]

The kernel was conceived of as what was essential and other features were left to be developed as part of the tools or software that would be available. Thompson explains: "The UNIX kernel is an I/O multiplexer more than a complete operating system. This is as it should be. Because of this outlook, many features are found in most other operating systems that are missing from the UNIX kernel. For example, the UNIX kernel does not support file access methods, file disposition, file formats, file maximum sizes, spooling, command language, logical records, physical records, assignment of logical file names, logical file names, more than one character set, an operator's console, an operator, log-in, or log-out. Many of these things are symptoms rather than features. Many of these things are implemented in user software using the kernel as a tool. A good example of this is the command language. Maintenance of such code is as easy as maintaining user code. The idea of implementing 'system' code and general user primitives comes directly from MULTICS."[13]

During the same period that Bell Labs researchers were doing their early work on UNIX, the Bell System was faced with the challenge of automating their telephone operations using minicomputers.

"The discovery that we had the need – or actually, the opportunity – in the early 1970s to use these minis to support telephone company operations encouraged us to work with the UNIX system," writes Berkeley Tague.[14] "We knew we could do a better job with maintenance, traffic control, repair, and accounting applications."

"The existing systems were made up of people and paper," he relates, "The phone business was in danger of being overwhelmed in the early '70s with the boom of the '60s. There was a big interest then in using computers to help manage that part of the business. We wanted to get rid of all of those Rolodex files and help those guys who had to pack instruments and parts back and forth just to keep things going."

He goes on to describe the kind of operations that the Bell Systems

needed to automate.

Just as Operating Systems people in the Bell System had come to recognize the need for portability in a computer operating system, Ritchie and Thompson and the other programming researchers at Bell Labs had created the computer language C and rewritten the majority of the UNIX kernel in C and thus had made the important breakthrough of creating a computer operating system that was not machine dependent. Eventually, 10,000 lines of the code were rewritten in C and thus could be transported to other computer systems. Describing the advance the UNIX time sharing system represented, Thompson and Ritchie presented their first paper on UNIX at the Fourth ACM Symposium on Operating Systems Principles, IBM Thomas J. Watson Research Center, Yorktown Heights, New York, October 15-17, 1973.[15]

With the research breakthrough of a portable computer operating system, "the first UNIX applications," writes August Mohr, in an article in *UNIX Review*, "were installed in 1973 on a system involved in updating directory information and intercepting calls to numbers that had been changed. The automatic intercept system was delivered for use on early PDP-11s. This was essentially the first time UNIX was used to support an actual, ongoing operating business."[16]

Also, Bell Labs made the software available to academic institutions at a very small charge. For example, John Lions, a faculty member in the Department of Computer Science at the University of New South Wales, in Australia, reported that his school was able to acquire a copy of research UNIX Edition 5 for $150 ($110 Australian) in December, 1974, including tape and manuals.[17]

The automation introduced at AT&T had the benefit of research done not only at Bell Labs, but also by researchers in the academic community.

Early in its development, word of the UNIX operating system and its advantages spread outside of Bell Labs. (Several sources attribute this to the paper that Ritchie and Thompson presented on UNIX at the Symposium on Operating Principles in 1973.)[18]

UNIX was attractive to the academic Computer Science community for several reasons. After describing the more obvious advantages like its price, that it could be shaped to the installation, that it was written in

C which was attractive when compared with assembly language, that it was sufficiently small that an individual could study and understand it, John Stoneback, a faculty member at Moravian College, writes: “UNIX had another appealing virtue that many may have recognized only after the fact — its faithfulness to the prevailing mid ‘70s philosophy of software design and development. Not only was UNIX proof that real software could be built the way many said it could, but it lent credibility to a science that was struggling to establish itself as a science. Faculty could use UNIX and teach about it at the same time. In most respects, the system exemplified good computer science. It provided a clean and powerful user interface and tools that promoted and encouraged the development of software. The fact that it was written in C allowed actual code to be presented and discussed, and made it possible to lift textbook examples into the real world. Obviously, UNIX was destined to grow in the academic community.”[19]

In trying to teach his students the essentials of a good operating system, John Lions of the University of New South Wales in Australia describes how he prepared a booklet containing the source files for a version of Edition 6 of research UNIX in 1976 and the following year completed a set of explanatory notes to introduce students to the code. “Writing these,” he recounts, “was a real learning exercise for me. By slowly and methodically surveying the whole kernel, I came to understand things that others had overlooked.”[20]

This ability to present his students with an exemplary operating system kernel was an educational achievement. Lions writes: “Before I wrote my notes on UNIX, most people thought of operating systems as huge and inaccessible. Because I had been at Burroughs, I knew that people could get to learn a whole program if they spent some time working at it. I knew it would be possible for one person to effectively become an expert on the whole system. The Edition 6 UNIX code contained less than 10,000 lines, which positioned it nicely to become the first really accessible operating system.”[21]

In keeping true to the UNIX community spirit of helping each other, Lions wrote a letter to Mel Ferentz, Lou Katz and others from USENIX (then the academic UNIX users association) and offered to make copies of his notes available to others. After some negotiation with Western

Electric over the patent licensing, he distributed the notes titled "A Commentary on the UNIX Operating System" to others with UNIX licenses on the conditions that Western Electric had set out.[22]

Describing how research UNIX and its adoption at academic institutions has served to develop computer science, Doug Comer writes: "The use of UNIX as a basis for operating systems research has produced three highly desirable consequences. First, the availability of a common system allowed researchers to reproduce and verify each others' experiments. Such verification is the essence of science. Second, having a solid base of systems software made it possible for experimenters to build on the work of others and to tackle significant ideas without wasting time developing all the pieces from scratch. Such a basis is prerequisite to productive research. Third, the use of a single system as both a research vehicle and a conventional source of computing allowed researchers to move results from the laboratory to the production environment quickly. Such quick transition is mandatory of state-of-the-art computing."[23]

Not only did research UNIX serve the academic community, but the contributions of the academic community were incorporated into research UNIX. An example, is the work at the University of California, Berkeley (UCB) of designing a virtual memory version of UNIX for the VAX computer which was later optimized and incorporated into a release of UNIX.

"A tide of ideas," explains Comer, "had started a new cycle, flowing from academia to an industrial laboratory, back to academia, and finally moving on to a growing number of commercial sites."[24]

Summarizing the relationship between Bell Labs and the academic community in developing UNIX, Comer concludes: "UNIX was not invented by hackers who were fooling around, nor did it take shape in a vacuum. It grew from strong academic roots and it has both nurtured and taken nourishment from academia throughout its development. The primary contributors to UNIX were highly educated mathematicians and computer scientists employed by what many people feel is the world's premier industrial research center, Bell Laboratories. Although they were knowledgeable and experienced in their own right, these developers maintained professional contacts with researchers in academia,

leading to an exchange of ideas that proved beneficial for both sides. Understanding the symbiotic relationship between UNIX and the academic community means understanding the background of the system's inventors and the history of interactions between universities and Bell Laboratories."[25]

John Lions, reviewing his experience as part of the UNIX community, concludes, "We have made a large number of contacts and exchanged a great deal of information around the world through this UNIX connection. Possibly that is the nicest thing about UNIX: it is not so much that the system itself is friendly but that the people who use it are."[26]

It is a rare and wonderful event in the development of human society when a scientific and technological breakthrough is made which will certainly affect the future course of social development and which becomes known when its midwives are still alive to tell us about it. The creation of UNIX, the product of research at Bell Labs, the then regulated AT&T system, and academic computer science, and a valuable invention for computer science, for computer education and for the education of the next generation of computer scientists and engineers, is such an event.

Notes:

(1) Douglas Comer, "Pervasive UNIX: Cause for Celebration," *UNIX Review*, October, 1985, p. 42.
(2) From Dennis Ritchie, "The Development of the C Language," ACM, presented at Second History of Programming Languages Conference, Cambridge, Mass, April 1993, p.1.
(3) Ritchie, p. 1-2.
(4) Ibid., p 2.
(5) ibid.
(6) From M. D. McIlroy, E. N. Pinson, and B. A. Tague "UNIX Time-Sharing System Forward," *The Bell System Technical Journal*, July-Aug 1978, vol 57, no 6, part 2, pg. 1902.
(7) Ritchie, p. 5.
(8) ibid.
(9) ibid., p. 9.
(10) M.D. McIlroy, "UNIX on My Mind," *Proc. Virginia Computer Users Conference*,

vol 21, Sept. 1991, Blacksburg, p. 1-6.
(11) K. Thompson, "UNIX Implementation," *The Bell System Technical Journal*, vol 57, No. 6, July-August 1978, p. 1931.
(12) ibid., p. 1931-2.
(13) ibid., p. 1945-6.
(14) "Interview with Berkeley Tague," *UNIX Review*, June 1985, p. 59. (See also *Sorry Wrong Number*, by Alan Stone, N.Y. 1989, p. 155-156 describing the service crisis experienced by AT&T during this period and how UNIX helped to solve the problem.)
(15) See reference in UNIX(tm) Time-Sharing System: *UNIX Programmers Manual*, 7th edition, vol 2, Murray Hill, f/n p. 20). See also Ritchie's account of the creation of C by early 1973 in "The Development of the C Language," ACM, presented at Second History of Programming Languages Conference, Cambridge, Mass, April 1993, p. 1.
(16) Mohr, "The Genesis Story," *UNIX Review*, January 1985, p. 26.
(17) See, for example, McKusick, "A Berkeley Odyssey" in *UNIX Review*, January 1985, p. 31, and Peter Ivanov, "Interview with John Lions," *UNIX Review*, October, 1985, p. 51.
(18) See "An Interview with John Lions," in *UNIX Review*, October, 1985, p. 51.
(19) From John Stoneback, "The Collegiate Community," *UNIX Review*, October 1985, p. 27.
(20) Lions, p. 52
(21) Lions, p. 52-3.
(22) ibid., p. 53.
(23) Comer, p. 44.
(24) Comer, p. 43.
(25) Comer, p. 34, 42.
(26) Lions, p. 57.

Spreading UNIX Around the World:
An Interview with John Lions

[Editor's Note: Looking through some magazines in a local university library, I came upon back issues of *UNIX Review* from the mid 1980's. In these issues were articles by or interviews with several of the pioneers who developed UNIX. As part of my research for a paper about the history and development of the early days of UNIX, I felt it would be helpful to be able to ask some of these pioneers additional questions based on the events and developments described in the *UNIX Review* Interviews.

Following is an interview conducted via E-mail with John Lions, who wrote *A Commentary on the UNIX Operating System* describing Version 6 UNIX to accompany the "UNIX Operating System Source Code Level 6" for the students in his operating systems class at the University of New South Wales in Australia. Lions' important book provided some of the earliest printed commentary and documentation of the UNIX kernel. John Lions is a Professor of Computer Science in the School of Computer Science and Engineering, at the University of New South Wales.]

Q: John, I have been reading with joy the interview with you that was published in *UNIX Review* in October, 1985. I found it inspiring because it showed the hard fight you and your colleagues and students took up to be able to adopt UNIX at your University and to help to spread it in Australia and around the world. In the *UNIX Review* article you describe the arrival of UNIX saying "UNIX was a revolutionary force on our campus." You tell how the University of New South Wales decided to purchase a Cyber 72 computer in 1974. But, since the Cyber could only recognize User200 terminals which were by that time obsolete, the University bought some PDP-11/40's to emulate User200s. You describe how you wrote for information about UNIX after reading an article by Ritchie and Thompson published in the "Communications of the ACM," and explained how a copy of Edition 5 tape and manuals arrived in late December, 1974. A little later in the Interview you relate how Ian Johnstone with assistance from others wrote a new User200 emulator "that ran under UNIX. That," you point out, "became the first application of UNIX to be written in Australia .... This exercise proved to be extremely important. With a PDP-11," you explain, "completely to ourselves, we most likely would have run vanilla UNIX on it and been happy. But because we had to provide the User200 emulator, we had to learn a lot about the system and pay a lot of attention to performance issues. We needed help, but we couldn't get any from outside sources. So we ended up generating our own expertise."

Lions: Undoubtedly true....

Q: What was it about UNIX that led you to do the hard work that

you did? Were you aware of the power that it promised? Was that some of the consideration or was it more practical – that you had certain things you wanted to be able to do and could hack to get the UNIX system to do it?

Lions: UNIX was wonderfully plastic. We changed things to adapt them to our situation...because it was a challenge, and we were having fun!

Q: You then say that through your work on UNIX you started to make a few friends elsewhere on campus. Were they from any other particular department? How did you begin to build a user group? Did you start having formal meetings?

Lions: Other people at the other batch stations were interested in solving the same problems as we were, so we found a common cause. This included the Library which in those days had passwords for the ordinary user accounts, but not for the super-user...very convenient!

Q: Can you say what kinds of similar problems people in other universities were encountering at the time that led you to be able to work together?

Lions: In a word...isolation.

Q: Do you have any idea why UNIX was so widely adopted at other Australian Universities?

Lions: We spread the news evangelically...We were very anxious to share our accumulated knowledge and to experiment...and we wanted to share it with others. We were having fun!

Q: You say that UNIX has possibly made a deeper penetration in Australia than in any other country.

Lions: That comment has to be understood in its proper context. I would not make it today. UNIX penetration is now 100% by university, though not by department within universities. The Internet is heavily UNIX-dependent, so I believe.

Q: In the *UNIX Review* interview, you describe how in 1975, Ian Johnstone who was acting as a tutor for the operating systems course you taught, asked, “Why don’t we run off a few of the source code files for the kernel and ask the students to take a look at them? Then we can ask them some questions; maybe it will be interesting.” What kind of questions did you folks intend to ask?

Lions: The same kind that the *Commentary* answers....

Q: After you took his suggestion and you both selected what seemed like a reasonable subset of the kernel and handed it out to students, you report that you asked them questions, but that they didn't have enough information to answer them so "they came back to us with questions of their own many of which we couldn't answer." Can you say any more about how the students suggested that you offer the complete kernel for study?

Lions: They suggested that it should be all or nothing. The selection of code I finally printed (on a DECwriter) is only complete in a limited sense. Section Five that deals with device drivers could have been much longer.

Q: Was there any special reason that you took their suggestions? What led to the preparation in 1976 of the booklet containing the source files for a version of Edition 6 UNIX that could run on a PDP-11/40 system?

Lions: Seemed reasonable at the time...what other options reasonably existed?

Q: You say "Writing these was a real learning exercise for me. By slowly and methodically surveying the whole kernel, I came to understand things that others had overlooked." Can you give any examples of what you came to understand that others had overlooked?

Lions: No. I guess what I meant to say was that I obtained an integrated view that allowed me to see more connections in the code than others did. I used to test the students' knowledge and understanding by weekly tests. Most years there would be two tests on each of the first four sections of the code.

Students could sit for both tests in each section, but they were discouraged from submitting more than one answer for marking. If they chose to submit two answers, their mark was the better of the two, less 10%. This allowed students to recover from a bad first result, while discouraging them from trying again if their first attempt was reasonable. Marking was always a problem as overnight turnaround was needed.

The sophistication of the questions increased over the years and toward the end, some new questions were quite devious. (Don't ask me

for examples!)

Q: In the *Commentary* you say: "A decision had to be made quite early regarding the order of presentation of the source code. The intention was to provide a reasonably logical sequence for the student who wanted to learn the whole system. With the benefit of hindsight, a great many improvements in detail are still possible, and it is intended that these changes will be made in some future edition." Did you ever write the future edition making the changes?

Lions: No. There had been a three year gap between [UNIX -ed] Editions Six and Seven. This created a window of opportunity for us that never really occurred again.

Q: In your *Commentary* you say "You will find that most of the code in UNIX is of a very high standard. Many sections which initially seem complex and obscure, appear in the light of further investigation and reflection, to be perfectly obvious and 'the only way to fly.' For this reason, the occasional comments in the notes on programming style, almost invariably refer to apparent lapses from the usual standard of near perfection .... But on the whole you will find that the authors of UNIX, Ken Thompson and Dennis Ritchie, have created a program of great strength, integrity and effectiveness, which you should admire and seek to emulate."

Lions: That is what I believed then...and still do.

Q: Can you say any more about the conclusion you drew of the high standard of code in the UNIX kernel? Do you feel that students and others who studied your book and the code did emulate it? Did that help improve the level of code of those who had access to your book and the source code?

Lions: In a general sense, I believe the answer is 'yes': students did learn better coding practices.

Q: In the *UNIX Review* article, you relate that in 1977 at the University of New South Wales you were developing your own PDP version of UNIX to handle heavy student loads and that Ian Johnstone, Peter Ivanov and Greg Rose developed a "sanitized extended version of UNIX." And you made some changes to the kernel. Can you say what the most important ones were?

Lions: We fixed bugs that we found...or had introduced ourselves. I cannot recall what they were...and of course the Seventh Edition changed everything anyway. We only did it once: that was enough.

Q: Was your examination of the kernel for the *Commentary* helpful in determining what changes to the kernel were needed. For example, in the *Commentary* on p. 82 under "Some Comments" you say "'namei' is a key procedure which would seem to have been written very early, to have been thoroughly debugged and then to have been left essentially unchanged. The interface between 'namei' and the rest of the system is rather complex, and for that reason alone, it would not win the prize for 'Procedure of the Year.'" Earlier in the *Commentary* in chapter 19 (p. 82) you say "Copy the eight words of the directory entry into the array 'u.u_dent'." Then you comment, "The reason for copying before comparing is obscure! Can this actually be more efficient? (The reason for copying the whole directory at all is rather perplexing to the author of these notes.);" Were these problems clarified upon further examination or if not, did you make any effort to solve them when you folks made changes to the kernel?

Lions: No comment now. My understanding changed over the years, and some questions that may been obscure once were no longer so.

Q: In the *UNIX Review* Interview you explain that you were the first person from the UNIX community in Australia to spend a sabbatical at Bell Labs. Who invited you to the Labs? When? Why? What did you do once there?

Lions: After I started distributing copies of my notes on UNIX (Source Code and Commentary), I sent more than two hundred copies to BTL [Bell Telephone Laboratories -ed]. One night (sometime in 1978?), I had a phone call from Doug McIlroy saying BTL would like to assume responsibility for distributing those documents, and would I agree? I did. It saved me much work.

At the beginning of 1978, when I was starting to wonder what to do for my first sabbatical leave, I had another late night call, this time from Berkley Tague enquiring whether I might be willing to visit BTL, another easy decision.

In the middle of 1978, my family (us and two daughters) set off for the

USA, Madison, NJ in particular, where Berkley had arranged for us to rent the house of an academic from Drew University. (They were going to the south of France for his sabbatical!)

I can still remember arriving at 26 Morris Place, tired but pleased to be there (I think we must have rented a car from Newark airport). Shortly afterwards Berk arrived and introduced himself. We have been firm friends since then, with both him and his wife, Anne-Marie. He is undoubtedly one of nature's gentlemen.

Madison, N.J. is only a few miles (less than 10 – I forget!) from BTL. Incidentally, Berkley used to collect me each morning and drive me from Madison to the Labs, so my wife could have our car!

Q: You say in the *UNIX Review* interview that you worked in the UNIX Support Group while at Bell Labs during that first sabbatical and were able to introduce a number of utilities, including **pack**, etc. Can you say more about what your work was during that first sabbatical?

Lions: There were no expectations and I was given a free hand to follow my own interests. Fortunately for BTL I had lots of ideas, so there was never a problem.

Q: Do you know how your book was used as part of the work that USG [UNIX Support Group -ed] was doing? Do you know how it was used both elsewhere in Bell Labs and outside?

Lions: I had sent them the original copies of my notes. They reproduced them and provided one copy to each new licensee (so I believe). Each new licensee was allowed to make additional copies under specified conditions.

Q: At the end of the *UNIX Review* Interview you say that it was not so much that the UNIX system is friendly but that the people who use it are. Do you have any sense of what about UNIX makes this true?

Lions: Not really. That was just my experience. I think you ought to remember that BTL is a very special place, and its Research Department is also very special.

Q: What would you see as an appropriate way to commemorate the 25th anniversary of the creation of UNIX in 1994?

Lions: I gather USENIX is attempting to organize a meeting.

Automating Telephone Support Operations: An Interview with Berkley Tague

[Editor's Note: The following interview with Berkley Tague was also done while I was doing research for a paper on the history and significance of UNIX®. During the course of this research, I found an interview which had appeared in *UNIX Review*, ("Interview with Berkley Tague," June, 1985) The interview suggested some further questions which I sent to Berkley Tague via e-mail. His responses were very helpful and I thought that others would find them interesting as well. Therefore, we asked for permission to print them in the special issue of *The Amateur Computerist* to commemorate the 25th Anniversary of UNIX®. Following are the questions and his responses. He emphasized that his answers were his personal experience as part of the UNIX® development project and represent only one of many such views of the Bell Labs project. -RH]*

Q(1): Can you explain what the problem was that AT&T was trying to solve in the 1960's and 1970's with regard to labor intensive support operations that had to be automated (i.e., mechanized)? What kinds of operations were the problem and why did they need to be automated?

Tague: The push – it began about 1969 – was to use the computer to support the operation of the Bell System network. The effort was quite broad in scope: monitoring and alarms, maintenance and staff management, inventory and order control, revenue data collection and billing, traffic measurement and control, circuit provisioning, etc.

To take one example, the data that has to be collected to bill for calls was being stored on a variety of media – e.g. paper AMA tape, a punched paper tape a few inches wide, IBM compatible mag. tape, and probably a few others. These tapes had to be mounted, saved, trucked to DP [Data Processing -ed] centers for processing, etc. They could be lost or damaged in handling. An obvious solution was to send the data over datalinks – our own network – to the DP centers. Minicomputer-based

systems were designed to collect, pre-process and transmit this data in some cases; in others, the electronic switches themselves were programmed to do the job. The goal was to eliminate the need for people handling physical media – a process that is both expensive and error-prone.

Q(2): What work was done by Bell Labs in 1964-68 as part of the Multics Project? Why did AT&T get involved with the Multics collaboration? (Can you explain what were the labs' objectives?) A reference I have found mentioned that "Bell Labs' purpose was to have a good environment for our people to work in." (See "Putting UNIX in Perspective: An Interview with Victor Vyssotsky," *UNIX Review*, Jan., 1985, pg. 59) If that seems accurate to you, can you explain why that was the objective and what it meant? And what happened with the project that AT&T decided they had to drop out?

Tague: The Multics Project was a joint project of Bell Labs, the GE Computer Systems Division, and MIT's Project MAC to develop a new computer and operating system that would replace MIT's CTSS system, Bell Labs BESYS, and support the new GE machine. Bell Lab's objective was to obtain a computing system that could simultaneously support batch, time-shared and real-time processing under an operating system that provided a full set of features with exceptional security. GE, and MIT's Project MAC, had overlapping, but somewhat different sets of objectives. Why the project ended as it did – Multics never was used by Bell Labs or AT&T as other than a research tool – is a complex story that I don't fully know. (The part I think I know also may be only partly correct.) What is clear is that Multics was a seminal effort in computing science and operating systems design. It established principles and features of operating system design that today are taken for granted in any modern operating system. This design experience and the UNIX® system itself were the payoff for Bell Labs in the long run.

Vic's remark covers one of the objectives. Multics was intended to be a standard operating environment to support Bell Labs computing. That role was eventually filled by the UNIX® system, not Multics.

Q(3): What was learned from the project and was there a sense how this could be helpful at Bell Labs?

Tague: Much that was learned from the project was embedded in C and the UNIX® system. The security rings and related concepts also have had impact on subsequent computer security even though the full-blown security apparatus of Multics was not propagated in the UNIX® system. There are undoubtedly many other Multics features and concepts that were not in UNIX® but still had impact on computing science. I don't know all of the work that was done, especially after I left the project in 1967; the system was almost entirely rewritten after I left the project.

Q(4): What was going on with computer science research during this period at the Labs?

Tague: Bell Labs terminated its participation in the Multics project (except for a couple of people wrapping up loose ends) in the summer of 1969. At the same time, Bell Labs turned the large internal computing centers over to a new central Bell Labs organization called Computing Technology. This included the Murray Hill Computing Center that up to then had been operated by Computing Research.

These changes meant that a number of researchers in Computing Science were asked to redirect their efforts. In the review and redirection that ensued, one point of view was that Computing Research should focus exclusively on theoretical subjects such as automata theory, computing complexity theory, formal linguistics, etc. Since this would have excluded the UNIX® system experiments, it is fortunate that this viewpoint didn't prevail. (I don't think it ever got beyond a talking point in the debate.)

Q(5): You mentioned you came back to Bell Labs in 1970. What was the situation that brought you back there (if it was related to the development of UNIX®)? Were you involved with the development of UNIX® during that period?

Tague: I have never left Bell Labs. The confusion is the internal nomenclature. In 1967, I left Bell Labs Research – this is the basic research area that is located at Murray Hill and Holmdel – and took a position managing a software development group in Bell Labs Federal Systems division in Whippany, N.J. that was part of the Safeguard ABM project. I returned from that assignment in September of 1969 to Murray Hill to join the Murray Hill Computing Center. This was during the

transition that ended Bell Labs participation in Multics and transferred the MH Computing Center to the new central organization (see #4). I reported to co-directors – one in Research and one in the newly formed Computing Technology organization. All of these organizations were part of Bell Labs.

I was not involved with Multics development after 1967 when I joined Safeguard and only became involved with UNIX® officially in late 1971 or so. I began exploring its use in development for telephony applications late in 1972 or thereabouts. This led to requiring the UNIX® system for some development projects and forming the UNIX Support Group in September of 1973. This was a supervisory group of about five people that reported to me and was also part of Bell Labs. It was responsible for moving the UNIX® system from research to an internally supported product for software development and operations services.

Q(6): Can you explain when the work at Bell Labs on UNIX® was thought of as being helpful toward the problem that AT&T faced in automating support operations? Was there an awareness at the Labs that the work they were doing on UNIX® was not only toward creating a good programming environment to do their research in, but also to be able to create a good programming environment for others at AT&T who would be doing programming? If so how early was there this awareness? (If you know.)

Tague: The researchers involved had as their goal an operating system that would be good for software development from the start. Because they were their own first customers, the system was aimed at their fellow researchers. There is really not that much difference between the needs of research and [the needs of -ed] development in terms of tools and features, the difference is primarily in support, stability and reliability. Developers have deadlines and don't want any more dependence on new invention than absolutely necessary. Also, if you have development deadlines, you certainly don't want anyone changing the system independently as you do your work. As a researcher you may tolerate a fair amount of upset and revision if it improves your toolkit significantly. The creation of the UNIX® Support Group was precisely to provide the stability and support that would buffer develop-

ers from the experiments that researchers were daily imposing on the evolving UNIX® system.

As mentioned above, the UNIX® system was used by developers and supported operations in the field as early as 1972. There were about three different systems at that time that had been built in research for minicomputers, but I felt that UNIX® was the best of the lot since it had captured most of the best features of Multics in a small, elegant package. (See my *UNIX Review* interview, pages 60ff.)

Q(7): What difficulties were involved in trying to have UNIX® used for the task of automating support operations? Why was it being proposed? What obstacles did it face?

Tague: One major difficulty was convincing developers that they could depend on an operating system that had no official support and consisted of some 12,000 lines of uncommented code. (Proposing the UNIX Support Group was the obvious answer to this.) But UNIX® had one major advantage: I knew of no minicomputer vendor that had anything approaching it as part of their product line. Most minicomputer systems at that time were inferior to DOS 1 in function, speed and reliability. UNIX® had a rare opportunity to fill what was effectively a vacuum. I was pushing UNIX® onto our development community because I knew they needed it in spite of its shortcomings. A number of developers were planning to build their own operating systems – typically their first system and often their first major program. It was quite rational to suggest that someone who had been working for several years on his third operating system might be ahead of them.

Q(8): What led to the creation of the UNIX Support Group (USG)? When? What was its mandate?

Tague: I think I answered this one in the answer to #7. One of the nice properties of Bell Labs is that when you perceive a need, you can propose a solution with a good chance of being asked to go do it yourself. In 1973, I pointed out that there was a need to provide central support for the UNIX® systems we had been propagating into projects and volunteered to form the group in my department. By September, 1 was in business.

Q(9): Once the USG was formed (in 1973) was there a distinction

between the priorities of the Bell Labs people and the USG people in their collaboration? August Mohr's article [i.e., August Mohr, "The Genesis Story," *UNIX Review,* Jan., 1985] seems to indicate that they both agreed there was a need for portability despite different priorities.

Tague: When the USG was formed in September, 1973, Dennis Ritchie promised me the portable version in October and delivered it. It was a "no brainer" to go into business with the portable version. A goal of my effort was to gain vendor independence so we could get competitive bids on volume buys when we deployed these mini-based systems across the Bell System. There was one project that decided it couldn't wait until October and committed to the non-portable version, but that was the only project that used the non-portable version as far as I know.

Q(10): What were the problems and successes of USG?

Tague: The biggest problem was controlling the UNIX® system variants that continually emerged. New features and function were added by every project and the USG had to choose among or merge the variants in a continuing effort to filter out the redundancies and keep the best. Berkeley, the University not me, was doing this in parallel with us and with only loose coordination possible.

The success of USG was its contribution to the success of the UNIX® system. UNIX® created open systems and a multibillion dollar market. Not bad for a two person research initiative on an obsolete mini.

Q(11): Did Rudd Canaday's work represent similar objectives? What was the division between the two groups and why?

Tague: I am not sure what you mean by "Rudd Canaday's work" in this context. Rudd shares in the patent on the UNIX® file system which he did in Research as a colleague of Ken and Dennis. Later, he moved to the Business Information Systems (BIS) project and brought the UNIX® system with him. The Programmer's WorkBench (PWB) variant grew up in his department.

The BIS problem was to get a common "workbench" that would drive code onto any of the three or four commercial vendor's mainframes that were used by BIS. By putting a UNIX® system in front of the large mainframe systems, developers got the advantages of UNIX® in developing code and a place they could store debugged standard command sequences that drove development and testing on the target

mainframe.

The PWB support group was merged with the USG in about 1975 and the USG and PWB versions were integrated. Note that during the 70s, every development group modified the UNIX® system for its needs. The PWB group was one of many and was distinguished only by the quality of its work and the fact that it was widely deployed through a large and important project (BIS). But there were also very active groups at Columbus and at Holmdel Bell Labs locations that were modifying the system and offering their mods to the USG for inclusion in the base version.

The vice and virtue of UNIX® has always been its flexibility. You love its flexibility when you meet a new need, but you want a single standard version once it meets your needs. COSE is just the latest of many efforts to coalesce the variants into a common base.

Q(12): Who invited John Lions to the Labs? When? Why? What did he do once there?

Tague: Research asked me to invite him to work with the USG. He had written his wonderful book on the UNIX® system early in the game and we had found it most useful. We agreed to publish and distribute the book and wanted John to continue his work as one of the UNIX apostles in Europe and Australia. He wanted to come to Murray Hill for his sabbatical so it was a win/win situation. He spent two or three sessions at Bell Labs over the years and supplied us with many of his graduate students for sabbaticals and permanent employment. For me, he became not only a valued contributor, but a good friend. A truly delightful gentleman!

At this distance, I don't remember exactly what he worked on, but I asked him to extend his documentation of the system to some new features while giving him license to work with the researchers in whatever way was mutually fruitful. His book is a classic that is still worth reading by the would-be operating systems designer.

Q(13): Was his book *A Commentary on the UNIX Operating System* used at the Labs or in the USG? If so how?

Tague: Yes. We offered it as a part of the documentation package for those who wanted to understand or modify the UNIX® source that the USG shipped. It was very useful as an introduction even though the

code no longer matched the book. It outlined the conceptual architecture very clearly in the early short form of the system before it had accreted all the minor changes and feature additions that disguised the original clarity of its structure. All new people were given a copy when they joined the USG and I suspect most development groups did the same.

Q(14): Who else did you invite from his school and why? What work did they do?

Tague: It's a long list and I don't trust my memory, but Andrew Hume is still in research at Murray Hill. Ian Johnstone left us for Sequent and I believe is now working in Boston with another firm. Peter Ivanov and Piers Dick-Lauder were two others who were part of my department, and there were likely others who came along after I left the scene. They all worked on UNIX® system development, but I couldn't tell you what parts they worked on. I do remember that Ian worked on the first multiprocessor versions, but he did many other things prior to that. I used to kid about running the "Australian Chair of Computing Science" at Bell Labs.

Q(15): In the *UNIX Review* interview you are quoted saying that you originally opposed having UNIX go out to colleges. But it did anyway. What was the reason you opposed it (if anything in addition to what you said in the interview)? Why was it allowed to go out over your objections? What would you say was the result of making it available to academic institutions outside of AT&T?

Tague: I don't have anything substantive to add to what I said in the interview, but perhaps I can clarify what I was trying to say there. My position in the early '70s was that we should make C available outside Bell Labs in the way we released other tools for university (and occasionally for commercial) use through our patent licensing organization. This release was "caveat emptor" — i.e., dollars on the table up front with no support included. I opposed the release of the UNIX® system without support because I was afraid it would be adopted for commercial use by someone who could call up the president of AT&T and demand help. I knew that any such request would likely find its way to my department and we were not ready to provide outside support. I also had a vague idea that the system might be more valuable to AT&T as a proprietary AT&T system. I was wrong. The system was rapidly

picked up by academic and industrial research groups that were well prepared to deal with the no support proviso and it had no takers in the DP community until it was offered as a supported product. The value of the system as a portable open standard was evident early and when AT&T was allowed to enter the computing business, it was a clear winner as a de facto standard.

Q(16): How has UNIX® been used for support operations? When was it understood that it could be used? Does it continue to be used?

Tague: The UNIX® system is the standard operating environment for almost all internal development and much research at Bell Labs and has been so since the early eighties. The BaseWorX© platform that we use and sell for operations support systems includes UNIX® SVR4 as an essential component. Operations support usage started in about 1971 and continues today.

Q(17): Was portability the only problem that had to be solved before UNIX® could be used internally for support operations? If there were other such problems what were they?

Tague: See the answers above. There were many extensions and features that have been added over the years – interprocess communications mechanisms, streams, transaction processing, databases, etc. – and most of these are useful to the broad community of Bell Labs users. Each was originally motivated by some perceived user problem. As the UNIX® system has evolved, it has incorporated valuable features from other systems and served as a base for pioneering new ideas. It is interesting to see the UNIX® to Mach to NT evolution as the kernel is subdivided to meet new fundamental needs while still maintaining original functions in almost the same form as the original V6. Even DOS imitates a good bit of the command set in a roughly familiar way.

Q(18): In the *UNIX Review* Interview you are quoted saying that you expected the internal development of UNIX® within AT&T to be mirrored outside of AT&T. Has that happened or not? Do you have any insight why?

Tague: I am going to duck this one. I am not sure what I had in mind when I said it at this point. I guess I was thinking that the external market would continue the process of expanding the system rapidly to include new features, followed by attempts to filter them into a coherent

extended standard. Neither the internal development nor the external development has gone as I might have predicted (or might have hoped) a decade ago, but the system is still alive, evolving and providing service on a broader spectrum of hardware than any other system around.

Q(19): What are the successes of it all that you see? (of the UNIX® and USG developments?) the problems?

Tague: See my answer to #10 above. The biggest problem continues to be the variants and the difficulty of getting an industry standard API that supports "shrink wrapped" software packages. The issue of a standard "desktop" GUI is probably a close second. COSE is trying again and I wish them well.

Q(20): What would you see as an appropriate way to commemorate the 25th anniversary of these achievements?

Tague: Stop for a moment and contemplate how much of what we take for granted in today's operating systems was established in that post-Multics synthesis by Ken and Dennis, acknowledge the pioneers of Projects MAC and Multics, and then go back to work on defining the next plateau.

[*Note: Please understand that what you have is my personal view of the UNIX® system developments and not necessarily those of Bell Labs. Your questions made me go over an interesting part of my career with Bell Labs and you likely got more than you bargained for. Each participant in UNIX® system development has his or her own view of this period and they don't always agree as to the order or interpretation of events. I cannot claim to more than one such view and, as an early enthusiast, may well overestimate my personal role in the story. Berk Tague]

Creating a New Technology: The Technology of Software Production An Editorial on the 25th Anniversary of UNIX

[Editor's Note: The following editorial is an effort to encourage a discussion of the significance of UNIX. We welcome alternative viewpoints, comments, etc. on the issues raised in this editorial.]

In his book *Ancient Society*, Louis Henry Morgan, who has been called the father of anthropology, described the important role that the creation of iron played in the advance of human civilization. He wrote: "When the barbarian, advancing step by step, had discovered the native metals, and learned to melt them in the crucible and to cast them in moulds; when he had alloyed native copper with tin and produced bronze; and, finally, when by a still greater effort of thought he had invented the furnace, and produced iron from the ore, nine tenths of the battle for civilization was gained."[1]

"Furnished with iron tools," continued Morgan, "capable of holding both an edge and a point, mankind were certain of attaining to civilzation."[2]

Morgan called the production of iron, "the event of events in human experience.... Out of it," he wrote, "came the metallic hammer and anvil, the axe and the chisel, the plow with an iron point, the iron sword; in fine, the basis of civilization which may be said to rest upon this metal."

With the birth of the modern computer, the development of a new technology has been put on the agenda for our times. This technology will not be forged in furnaces as were the tools of our ancestors. It is not even possible to grab or hold this technology.[3] This technology is the technology of software production. The challenge for our society is to be able to develop a software production technology.

By the early 1960's, Bell Labs researchers realized that there would

be an ever increasing role that computers would play in the operations of a large utility like the U.S. telephone system. And the telephone service crisis of the late 1960's showed that indeed AT&T had to automate to be able to meet its obligations to the public in the U.S. To accomplish this automation, they realized they would have to put software production on a more rational basis. Writing about this challenge, researchers explained: "One might think that because typing is easier than soldering, it should be easier to change software than to change hardware. However, the ease of changing software depends on the language at hand, the quality of the editor, the file system, structure, etc."[4]

The UNIX time sharing system was created 25 years ago at Bell Labs to help to fulfill this need. One of the important contributions of UNIX in its 25 years of development has been the role that UNIX has played to put the production of software on a more rational and scientific basis.

Describing the problem facing computer software pioneers, Evan L. Ivie, a researcher at Bell Labs, writes: "Although the computer industry now has some 30 years of experience, the programming of computer-based systems persists in being a very difficult and costly job. This is particularly true of large and complex systems where scheduled slips, cost overruns, high bug rates, insufficient throughput, maintenance difficulties, etc., all seem to be the rule instead of the exception. Part of the problem stems from the fact that programming is as yet very much a trial and error process."[5]

Ivie explains that there is "only the beginnings of a methodology or discipline for designing, building and testing software. The situation is further aggravated," he adds, "by the rapidly changing hardware industry and by the continuing evolution of operating systems which continues to nullify much of the progress that is made in the development of programming tools. What can be done," he asks, "to move the programming industry toward a more professional and stable approach to software development?"[6]

After enumerating several possible alternatives, he proposes "a very different approach to improving the development process." The approach he proposes is that "the programming community develop a

program development 'faculty' (or facilities) much like those that have been developed for other professions." And he cites as examples a "carpenter's workbench," or a "dentist's office," or an "engineer's laboratory." This approach, he maintains, "would help focus attention on the need for adequate tools and procedures; it would serve as a mechanism for integrating tools into a coordinated set; and it would tend to add stability to the programming environment by separating the tools from the product (the current approach," he notes, "is equivalent to carpenters leaving their tools in each house they build.)"[7]

Ivie's proposal was carried out in what came to be known as the Programmer's Workbench. A set of UNIX and C software development tools were created and were made available to programmers, even though they were working on different machines and with different operating systems.

P. J. Plauger, one of the UNIX pioneers who helped to define the concept of a software development tool writes that "the programmer working as a tool builder finds that his impact extends far beyond the task he may originally have set out to solve."[8]

Doug McIlroy, one of the creators of UNIX, explains that the term "software tools" was still unnamed around the circle of UNIX pioneers until Brian Kernighan and Plauger wrote the book *Software Tools*. "The idea nevertheless became a 'guiding principle,'" writes McIlroy.

It was only with the "liberation of the grep pattern matching program from within the ed editor, that the unarticulated notion of software tools…was finally brought home."[9] "More than any other program," McIlroy explains, "grep focused the viewpoint that Kernighan and Plauger christened and formalized in 'Software Tools': make programs do one thing and do it well, with as few preconceptions about input syntax as possible."[10]

Kernighan and Plauger give the following definition of a software tool: "It uses the machine; it solves a general problem not a special case; and it's so easy to use that people will use it not build their own."[11]

Describing one of the important problems that UNIX was called on to solve, Dick Haight, another of the pioneers, explained the problem facing AT&T, one of the largest corporations in the U.S., in the early 1970s.

"We had a real problem to solve," Haight elaborates in an interview. "For one thing, we had a fairly large group of software developers at the Labs working on several different mainframes. The biggest group... consisted of people working on the IBM 360s or 370s. The programmers working on the IBM 360 or 370 had to contend with batch processing and IBM's Job Control Language (JCL) problems. Other programmers were working with the Univac and another group with Xerox computers. Only a few were using early but expensive time sharing like TSO (IBM's timesharing) and Univacs Remand Service." Haight explains that "these systems not only offered very different programming environments but proved to be very expensive to use and very unfriendly." As part of a group formed in the Business Information Systems Program, (BISP), Haight and his colleagues were charged with the task of creating a more rational programming environment for the AT&T programmers. "Basically," he explains, "we ended up trying to give all these people a cheap text editing front end for interactive program entry." Haight and his colleagues had experience with the UNIX program development system and they used it to create the PWB – a Programmer's Workbench facility that eventually included over 300 tools.

The Programmer's Workbench (PWB) created at Bell Labs in response to this need, encouraged the development of machine independent programming tools. "Each tool," they maintained, "must now function for programmers developing code for a number of different vendor machines.... One is thus forced into a more stable and generalized software development approach which should be more applicable to new machines."[12]

The PWB was conceived of in mid April 1973, installed on the first workbench computer in October 1973 (PDP 11/45) and by 1977 the Programmer's Workbench computers were serving 1000 developers. Eventually, it was used to write the thousands of lines of computer code needed for the development of the 5ESS switch that AT&T was installing.[13]

The contribution of UNIX to the creation of such advances in the programming profession needs to be studied and built on. The problems of software development are the difficult problems that the computer

revolution has thrust on center stage.

Writing in a time of similar technological change, Denis Diderot, who was the editor of the Great French Encyclopedia (*Encyclopédie, ou dictionaire raissoné des sciences, des arts et des métiers*) realized the need to catalog and graphically present drawings of the tools and industrial processes in use in industry up to that time. Though he was attacked for revealing the secrets of the trades, this work made it possible for others to study the level of tool development in use and improve on it.

In his book *The History of the Machine* (NY, 1979), Sigvard Strandh describes how tool production had been at a standstill from the middle ages until the early days of the industrial revolution. He explains that there were previous plans for building some of the components of the steam engine, but the tools that would make doing so possible were not yet available.

The work of those like Diderot to publish descriptions of what knowledge was known in the production of tools and industrial processes, helped to advance the state of the art of the technology of tool production, as they made it possible to build on what had been achieved.

In an article describing how the PWB was ported to the IBM System/370, the authors write that there were at the time 300 UNIX or C tools that were part of the Programmer's Workbench. Sadly, many of those tools no longer seem to be available. Commenting on the state of availability of UNIX tools, another UNIX pioneer, John Mashey observed, "Our ability to organize software and make it available has lagged our ability to write it."[14] Also, the concept of a software component catalog, first mentioned by Doug McIlroy in 1968 before the creation of UNIX, and then reintroduced in the Bell Labs Journal articles[15] needs to be reviewed and reestablished as a goal. The scientific principle that Plaugher emphasizes, is that the only way to make substantial progress in any field is by building on the work of others. Just as UNIX was built on the lessons that Ritchie and Thompson and others learned from the experience of CTSS and the early Multics collaboration, so it is helpful to learn from the experience of UNIX in order to make any further advances. That is the challenge that the 25th anniversary of UNIX puts on the agenda for those who want to advance.

As Henry Spencer and Geoff Collyer wrote in their article "News Need Not Be Slow": "To know how to get somewhere, you must know where you are starting from."

Notes

(1) Louis Henry Morgan, *Ancient Society*, Chicago, 1877, p.4
(2) ibid., p. 43
(3) Fred Brooks Jr. explains that "Software is invisible and unvisualizable.... The reality of software is not embedded in space." from "No Silver Bullets," *UNIX Review*, Nov. 1987, p. 41.
(4) "Microcomputer Control of Apparatus, Machinery, and Experiments" by B. C. Wonsiewicz, A. R. Storm, and J. D. Sieber, "The Bell System Technical Journal," July-August 1978, vol 57, no. 6, pt 2, p. 2211.
(5) "Programmer's Workbench — A Machine for Software Development," *Communications of the ACM*, Oct. 1977, vol 20, no 10. p. 746
(6) ibid.
(7) ibid.
(8) "Minicompilers, preprocessors and other tools," in *AFIPS Conference Proceedings*, vol 44, 1975, pg. 281.
(9) "UNIX on My Mind," *Proc. Virginia Computer Users Conference*, vol 21, Sept 1991, Blacksbury, pg. 1-6.
(10) From e-mail correspondence.
(11) "Interview with Dick Haight," *UNIX Review*. May 1986.
(12) "Programmer's Workbench," p. 749.
(13) See "A UNIX System Implementation for System/370" by W.A. Felton, G. L. Miller, and J. M. Milner, *Bell Laboratories Technical Journal*, October, 1984,
(14) "UNIX Leverage – Past, Present, Future," *Usenix Winter 1987 Conference Proceedings*, p. 8
(15) See for example "Software Tools and Components" by R. F. Bergerson and M. J. Rochkind and "Cable Repair Administration System," by P. S. Boggs and J. R. Mashey, p. 1275 in *Bell System Technical Journal*, July-August 1982

The Development of Usenet News: The Poor Man's ARPAnet

[Editor's Note: The following article is part 2 of "From ARPAnet to Usenet" which appeared in vol. 5 no. 3/4 of the *Amateur Computerist*. Since Usenet News demonstrates the power of UNIX, we felt it appropriate to include it as part of this special issue. Also Usenet News celebrates its 15th anniversary this year.

The ARPAnet provided an exciting experimental environment for those who had access to U.S. Department of Defense contracts. Many of the computer science community, however, did not have such access, but also wanted to be part of an online community. Graduate students in computer science helped to broaden access to the wonders of the ARPAnet by creating Usenet News, which they originally referred to as "The Poor Man's ARPAnet."]

Part II

Usenet News was born in 1979 when Tom Truscott and Jim Ellis, graduate students at Duke University in Durham, NC, and Steve Bellovin, a graduate student at the University of North Carolina in Chapel Hill, NC, conceived of building a computer network to link the computers at their different schools together. Using homemade 300 baud autodial modems and the UNIX to UNIX copy program (uucp) that was being distributed with the UNIX Operating System, Version 7, Steve Bellovin wrote some simple UNIX shell scripts to have the computers automatically call each other up, search for changes in the files, and then copy the changes.

While e-mail and mailing lists had been common on the ARPAnet, Gregory G. Woodbury, a Usenet pioneer at Duke describes how "News allowed all interested persons to read the discussion, and to (relatively) easily inject a comment and to make sure that all participants saw it."[1] "The 'genius' of the netnews," he explains, "was to see that the shell, the **find** command, and uucp would allow categorized news discussions to be shared between machines that were only connected by a serial line."

Soon three computer sites, duke, unc and phs (i.e. at Duke [duke],

at the University of North Carolina [unc], and at the Physiology Department of the Duke Medical School [phs]) were hooked together and a simple program was running connecting the three sites. Woodbury explains that Dennis Rockwell, a graduate student in Computer Science at Duke, had gotten a Systems Programmer job for the Physiology Department at the Medical School. When Physiology decided to use UNIX for the project that Rockwell was working on, "then it was a matter of convenience to have a hardwired circuit between the two machines for moving programs back and forth (between CS and Physiology.)" And since Rockwell, "migrated back and forth between Physiology and CS, he was instrumental in getting the connection to 'phs' implemented," Woodbury recalls, "so that he didn't have to spend his working time across the street at CS."

Woodbury reports that the Netnews program that was created, using UNIX shell scripts, was slow. Tom Truscott explains that the Usenet pioneers did not intend to use the shell scripts for any real news traffic.

Stephen Daniel, a new graduate student in computer science at Duke in 1979 describes how "a news program written, I believe, by Steve Bellovin as a collection of shell scripts was already working, but it was slow, taking upwards of a minute of time on an unloaded PDP 11/70 to receive an article. I got involved," he explains, "when I happened to drop in on a conversation between Tom Truscott and Jim Ellis who were complaining about how slow this news program was. I suggested that if it was so slow it could easily be rewritten in C to run faster. I soon found myself volunteering to do just that." Daniel agreed to write the program in C along with Tom Truscott. This was the first released C version of Net News, which was known as 'A News.'

In January of 1980, Jim Ellis presented a talk at Usenix, the UNIX users association for technical and academic users. He tells how there were 400 people at the conference with no parallel sessions, so many came to hear his talk describing the Netnews uucp program.

The invitation Ellis handed out at the January 1980 conference explained: "The initially most significant service will be to provide a rapid access newsletter. Any node can submit an article, which will in due course propagate to all nodes. A 'news' program has been designed which can perform this service. The first articles will probably concern

bug fixes, trouble reports, and general cries for help. Certain categories of news, such as 'have/want' articles, may become sufficiently popular as to warrant separate newsgroups. (The news program mentioned above supports newsgroups.)"

"The mail command provides a convenient means for responding to intriguing articles. In general, small groups of users with common interests will use mail to communicate. If the group size grows sufficiently, they will probably start an additional news group...."

"It is hoped that USENIX will take an active (indeed central) role in the network. There is the problem of members not on the net, so hardware newsletters should remain the standard communication method. However, use of the net for preparation of newsletters seems like a good idea."[2]

In the scientific tradition of gaining knowledge from the testing of one's theory in practice, the pioneers of Usenet invited others to participate in the network and then to work out the problems that developed. Their Invitation urged: "This is a sloppy proposal. Let's start a committee. No thanks! Yes, there are problems. Several amateurs collaborated on this plan. But let's get started now. Once the net is in place, we can start a committee. And they will actually use the net, so they will know what the real problems are."

The software for the 'A News' program for Usenet News was part of the conference tape for general distribution at the Delaware Summer 1980 USENIX meeting. The handout distributed at this conference explained: "A goal of Usenet has been to give every UNIX system the opportunity to join and benefit from a computer network (a poor man's ARPAnet, if you will) ...."[3]

Describing why the term "poor man's ARPAnet" was used, one of the students, Stephen Daniel, explains, "I don't remember when the phrase was coined, but to me it expressed exactly what was going on. We (or at least I) had little idea of what was really going on on the ARPAnet, but we knew we were excluded. Even if we had been allowed to join, there was no way of coming up with the money. It was commonly accepted at the time that to join the ARPAnet took political connections and $100,000. I don't know if that assumption was true, but we were so far from having either connections or $$ that we didn't even try. The

'Poor man's ARPAnet' was our way of joining the CS community (Computer Science -ed), and we made a deliberate attempt to extend it to other not-well-endowed members of the community. It is hard to believe in retrospect," he writes, "but we were initially disappointed at how few people joined us. We attributed this lack more to the cost of autodialers than lack of desire."[4]

Unlike the ARPAnet, Usenet News was available to all who were interested as long as they had access to the UNIX operating system (which in those days was available at a very minimal cost to the academic community.) Posting and participating in the network was available at no cost besides what the colleges paid for equipment and the telephone calls to receive or send Netnews. Therefore, the joys and challenges of being a participant in the creation of an ever expanding network, the experience available to an exclusive few via the ARPAnet, was available via Usenet News to those without political or financial connections – to the common-folk of the computer science community.

As Daniel notes, Usenet pioneers report that they were surprised at how slowly Usenet sites expanded at first. But when the University of California at Berkeley (ucb) joined Usenet, links began to be created between Usenet and the ARPAnet. University of California at Berkeley was a site on the ARPAnet. At first, it is reported, mailing lists of discussions among ARPAnauts (as they were called by Usenet users) were poured into Usenet. Also by 1979-80, ucb was under contract to ARPA to provide a version of UNIX (Berkeley Software Distribution) for the ARPA contractors that were going to be upgraded to VAX computers.

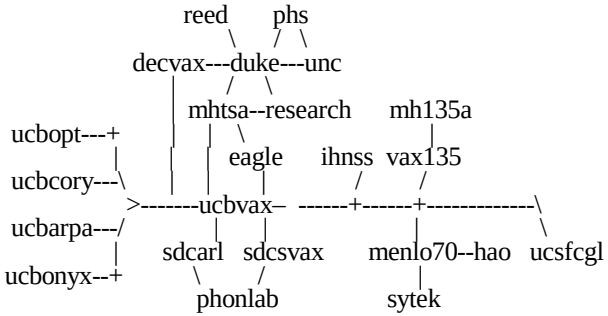
This first connection between the ARPAnet and Usenet News, only contributed to "the sense of being poor cousins," Daniel explains, "It was initially very hard to contribute to those lists, and when you did you were more likely to get a response to your return address than to the content of your letter. It definitely felt second class to be in read-only mode on human-nets and sf-lovers." (Those were two popular ARPAnet mailing lists. The human-nets mailing list, according to Tom Truscott, was a discussion of the implications of world-wide ubiquitous networking. This network of the future was referred to as "world net." Truscott reports that "it was a very interesting mailing list and possible only due to the ability of the network itself to permit those interested in this

obscure topic to communicate." -ed)

Daniel clarifies the different philosophy guiding the development of Usenet as opposed to that of the ARPAnet. He explains, "Usenet was organized around Netnews, where the receiver controls what is received. The ARPAnet lists were organized around mailing lists, where there is a central control for each list that potentially controls who receives the material and what material can be transmitted. I still strongly prefer the reader-centered view," he concludes.

With the increasing connections to the ARPAnet from Usenet, the numbers of sites on Usenet grew. A map from April 1981 shows the number of different sites on Usenet during this early period.

Usenet as of April 5, 1981:[5]



While the ARPAnet made possible electronic mail (e-mail) and discussion groups via electronic mail (i.e. mailing lists), Usenet made it possible for participants to post any message they wanted and it could be seen by everyone.

Pioneers from the early days of Usenet point out that originally there were a fair number of people who read all the posts. However, as Usenet started to grow there were different newsgroups that got set up and they were grouped by subject area, and the number of articles became too large for any individual to read them all.

Subject areas on Usenet range from the discussion of autos (i.e. rec.autos) to the science of economics (i.e. sci.econ). There are many computer related groups (e.g. comp.misc, comp.unix.misc, etc.) Originally, the creators of Usenet felt that most of the posts would be related to UNIX problems and bugs. But from the earliest days of Netnews,

there was a broad range of discussion.[6]

Often there have been problems that have developed in Usenet. The system administrators and others discuss the problem and argue out their differences. Sometimes what is called a "flame war" develops where people argue out their differences online.

The network has proven especially valuable in helping system administrators and programmers deal with the problems they run into with their work. Researchers using the network have found the collaborative work it makes possible very exciting.

Usenet now reaches over 10 million people and has more than 5,500 newsgroups. And the number of both is always growing. It has been made possible by the cooperative work of the participants and the programming tools of UNIX and C that were created by the research programmers at Bell Labs and added to by programmers and users around the world. Writing their programs using UNIX and C, participants in the UNIX community have written the A, B and C News and INN versions of Netnews which have made it possible for Usenet to accommodate an ever expanding number of megabytes of news from an ever expanding number of computer users. Also other programmers have contributed their time and labor to create newsreaders, mail programs and other software needed for the ever growing community of people participating in Usenet News.[7]

After the original porting of the ARPAnet mailing lists to Usenet, connections with the ARPAnet increased. Eventually, Usenet traffic was allowed to go through the ARPAnet. Steve Bellovin describes how the early porting of mailing lists like Human Nets and Sci-Fi Lovers onto Usenet was a force to broaden access to ARPAnet. He explains: "The first gateway of ARPAnet mailing lists to Usenet was an early force to have gateways within ARPAnet. Gateways to the ARPAnet were on the side things and in all likelihood not officially sanctioned. However, this provided the impetus for gateways into ARPAnet. This was the pressure on the ARPAnet to provide service to a larger number of people – a first step to transform the ARPAnet to become a part of the backbone of the Internet."[8]

In 1987, the U.S. government set up the NSFnet under pressure from academic scientists and computer scientists to provide additional

access to the developing network. The NSFnet replaced the ARPAnet as the backbone for the Internet and the ARPAnet was decommissioned in 1989.[9]

In its early years, Usenet was mainly transported via uucp using the telephone lines. A protocol was later created for Usenet to make it possible for it to ride on the ARPAnet and then the NSFnet and along the Internet, and thus cut down on phone costs for transporting it.

Following are some statistics that have been gathered of Usenet growth:

1979:	3 sites,	2 articles a day
1980:	15 sites,	10 articles a day
1981:	150 sites,	20 articles a day
1982:	400 sites,	50 articles a day
1983:	600 sites,	120 articles a day
1984:	900 sites,	225 articles a day
1985:	1,300 sites,	375 articles per day, 1+ Megabyte per/day
1986:	2,500 sites,	500, 2MB+ articles a day
1987:	5,000 sites,	1000, 2.5MB+ articles a day
1988:	11,000 sites,	1800, 4MB+ articles a day

Usenet sites posted about 26,000 articles per day to 4902 groups for 65 total megabytes (52 without headers) over the two week period before 8 March, 1993.[10]

Usenet has continued to grow and there are times that it seemed it would break under its ever increasing expansion. This concern is referred to as “The imminent death of the net is predicted” and has became a source of net folklore. During such difficult periods, mailing lists have been set up to discuss the problems. In one such discussion group, several of the participants put forward plans for a substantial change in Usenet, while other participants urged that it was crucial to first figure out the exact nature of the problem, if one wants to find a solution to that problem.[11]

[to be continued]

Notes:

(1) Gregory Woodbury in a post on Usenet News on April 12, 1993. "Unfortunately," Woodbury regrets, "I don't recall who had this flash of insight, and the 5 folk honored by the EFF (Electronic Frontier Foundation – ed) formed the core of folks who developed the idea into a real working system." (The Electronic Frontier Foundation gave an award to Tom Truscott and Jim Ellis and cited Steve Bellovin, Stephen Daniel and Dennis Rockwell for their creation of Usenet at its 1993 awards ceremonies.)
(2) "Invitation to a General Access UNIX Network" by Tom Truscott, Duke University. Copy from Usenet History Archives.
(3) Copy in the Usenet History Archives.
(4) From E-mail dated January 25, 1993, Usenet History List.
(5) Map from Usenet History Archives.
(6) See talk presented at the Michigan Association of Computer Users in Learning on "The Evolution of Usenet News: The Poor Man's ARPAnet," March 15, 1993.
(7) See for example Gene Spafford's "Usenet Software: History and Sources," available on Usenet News and Gregory G. Woodbury's "Net Cultural Assumptions" which has been posted on Usenet News.
(8) From Usenet History Archives.
(9) The CSNet created for Science and Computer faculty at Universities not connected to the ARPAnet was part of the pressure that led to the NSF Net. See description in "The Social Forces Behind the Development of Usenet News" in *The Amateur Computerist*, vol 5, no 1-2, Winter/Spring, 1993.
(10) Early statistics to 1988 compiled by Gene Spafford. (Oct 1, 1988, IEFT Meeting) Latest statistics by David C. Lawrence.
(11) See, for example, Usenet-II Mailing List of Nov. 10, 1985 in the Usenet History Archives. One of the posters to the list commented, "I don't want to be fixing the wrong problem."

What the Net Means to Me

by Michael Hauben
(hauben@columbia.edu)

The Net means personal power in a world of little or no personal power for those other than on the top. (Those on top are called powerful because of money, but not because of thoughts or ideas.) The essence of the Net is Communication: personal communication both between individual people, and between individuals and those in society who care

(and do not care) to listen. The closest parallels I can think of are:

- Samizdat Literature in Eastern Europe.
- People's Presses
- The Searchlight, Appeal to Reason, Penny Press, etc.
- Citizen's Band Radio
- Amateur or Ham radio.

However the Net seems to have grown farther and to be more accessible than the above. The audience is larger, and continues to grow. Plus communication via the Net allows easier control over the information — as it is digitized and can be stored, sorted, searched, replied to, and easily adapted to another format.

The Net is the vehicle for distribution of people's ideas, thoughts and yearnings. No commercial service deals with the presentation of peoples' ideas. I do not need a computer to order flowers from FTD or clothes from the Gap. I need the Net to be able to voice my thoughts, artistic impressions, and opinions to the rest of the world. The world will then be a judge as to if they are worthy by either responding or ignoring my contribution.

Throughout history (at least in the USA), there has been a phenomenon of the street-corner soapbox. People would "stand up" and make a presentation of some beliefs or thoughts they have. There are very few soapboxes in our society today. The '70s and '80s wiped out public expression. The financial crisis substituted a growing sentiment of make your money or shut up. In the late '80s and early '90s, the Net has emerged as a forum for public expression and discussion. The Net is partially a development from those who were involved with the Civil Rights movement, anti-war struggles and free speech movements in the '60s. The personal computer was also a development by some of these same people.

Somehow the social advances rise from the fact that people are communicating with other people to help them undermine the upper hand other institutions have. An example is people in California keeping tabs on gas station prices around the state using Netnews and exposing gougers. Another example is people publically reviewing music them-

selves – rather than telling others, "you should really go buy the latest issue of magazine 'X' (Rolling Stone, etc) as it has a great review." This is what I mean by people power – people individually communicating to present their view on something rather than saying go get commercial entity 'Xs' view from place 'Y.' This is people contributing to other people to make a difference in people's lives. In addition, people have debated commercial companies' opposition to the selling of used CDs. This conversation is done in a grassroots way – people are questioning the music industry's profit making grasp on the music out there. The industry definitely puts profit ahead of artistic merit, and people are not interested in the industry's profit making motive, but rather great music.

The Net is allowing two new avenues not available to the average person before:

1) A way of having one's voice heard.

2) A way of organizing and questioning other people's experiences so as to have a better grip on a question or problem.

Thus in some ways there is a regaining control of one's life from society.

These are all reasons why I feel so passionately about: 1) keeping the Net open to everyone, and having such connections being available publicly, and 2) Keeping the Net un-commercialized and un-privatized. Commercialism will lead to a growing emphasis on OTHER uses for the Net. As I said before, it is NOT important for me to be able to custom order my next outfit from the Gap or any other clothing store. Companies should develop their own networks if they wish to provide another avenue to sell their products. In addition, commercial companies will not have it in their interest to allow people to use the Net to realize their political self. Again let me reemphasize, when I say politics, I mean power over one's own life and surroundings. And this type of politics I would call democracy.

Plumbing The Depths Of UNIX: File Redirection and Piping

by Sue D. Nimms

[Author's Note: This article begins by describing the purpose of a shell; examples are given to show simple file redirection and piping; and concludes with a few real-life scenarios that involve relatively complex text-manipulation and processing using UNIX.]

A shell is a text-based command interpreter through which a user can interact (run programs and utilities) with UNIX. Similarity, a graphical user interface (GUI) is a graphical shell that, while easing the burden of typing, restricts access to commands otherwise available to the command-line shell user.

Just as there are various GUI shells, there exist various command-line shells: the Bourne Shell (**sh**), the C-Shell (**csh**), the Korn Shell (**ksh**), the Bourne Again Shell (**bash**) and the Plan-9 Shell (**rc**); the Bourne Shell (**sh**) is the most ubiquitous shell being available on all UNIX platforms.

The differences among the shells, for the most part, include:

- special features unique to one (like C shell's job-control – a feature designed to ease the management of programs running in the background)
- the syntax of internal commands (for-loop, while-loop and if/else constructs)

(Note: **bash** combines the best features of the **sh**, **csh** and **ksh**)

The shells allow customizing of the user's environment, the prompt, etc., and writing shell scripts. A shell script is a group of commands saved in a file and executed by giving the name of the file rather than typing in the commands interactively.

These differences aside, the underlying concepts of program execution remain constant no matter what shell one uses. Each program, when run, has three defaults (file-descriptors) associated with it:

1) it expects input from the keyboard (**stdin**: standard input),

2) it outputs normal text (**stdout**: standard output) to the screen and,
3) it also outputs error messages (**stderr**: standard error) to the screen.

These defaults can be reset, by the user, using redirection and piping to suit the appropriate situation.

The simplest example of redirecting output is saving the output of a UNIX command to a file. The **ls** command lists the contents of a directory to the screen, by default. This output can be redirected to a file using the symbol > placed between the command and a filename (the file will be created, if it does not exist):

EXAMPLE 1.

ls > foo

The output of **ls** (i.e. the list of files) is now contained in the file named 'foo'.

EXAMPLE 2.

ls -l > foo

The **-l** option to the **ls** command indicates that the user wishes a long-form listing (a listing that shows file-size, creation-date, ownership, etc). This example demonstrates that the redirection symbol and filename must come after any options.

The syntax for redirecting input is similar to that of redirecting output; the < symbol is used instead, followed by the filename whose contents should be in a form the command expects them.

EXAMPLE 3.

mail elf < foo

The program **mail** can be used to both read and deliver electronic mail (e-mail) to other users. In the above example, the file 'foo' contains the directory listing created in Example 1. This file is mailed to a user whose user-id is 'elf'.

Again, the redirection symbol and filename come after any options (in this case, the user-id of the recipient).

The command-pipe (represented by the symbol **|** is used to channel the output of one program into the input of another. It is placed between two commands.

EXAMPLE 4.

If a directory contains 100 files, a terminal-screen 25 lines long is not sufficient to view all the file-names; they will scroll off the screen. The **more** program is a pager, i.e. it pauses after every page for an indication (a key press) to continue.

ls -l | more

In this example, the output of **ls** (100 lines) is piped into the **more** program. The **more** program displays 24 lines of output of the **ls** command and pauses.

EXAMPLE 5.

ls -l | tr '[a-z]' '[A-Z]' | more

In this example three commands are used with pipes between them. The **tr** program is used to translate all lower-case characters (a-z) taken from its input (the **ls -l** command) to upper-case characters (A-Z). The output of the **tr** program is then piped into **more**.

Any number of commands may be run with piping between them.

SCENARIO 1.

Suppose you are a professor that has a teaching assistant who performs the grading of test-papers (150 students) and submits the marks to you by e-mail. To maintain privacy, only the student-numbers (in the first column) and marks (the second column) are submitted:

0124879 99
0132988 73
1987724 55
etc.

This list is saved in a file called ‘marks-list.’ It is sorted numerically on the first field, the student numbers.

You have a list of student names and student numbers in a file called ‘class-list’:

0124879 Scythe, J.
0132988 Smith, K.
1987724 Smith, L.
etc.

However, for a meeting, you require the student names and marks, essentially, a selective merging of the two files: the second field of the ‘class-list’ (the names file) and the second field of the ‘marks-list’ (the marks file).

The solutions to this problem are numerous. Two solutions will be demonstrated here; the more involved one first.

SOLUTION 1.

We begin by cutting the required fields from each of the files. We need the second field from both the ‘class-list’ (i.e. the names) and the ‘marks-list’ (the marks).

First we search for a command that fits our purpose by using the **apropos** command and the key-word “cut.” We find, among others, the **cut** command. At the $ prompt we type:

$ *apropos cut*
We find: ...
:
cut (1V) - remove selected fields from each line of a file
:

To use the **man** command to view the manual-page for **cut**, we type:

$ *man cut*

We learn we can **cut** field two (f2), specify that the field-separators are spaces ('**-d** ') [note: there is a space after the d], take input from the 'marks-list' file and save the output in the file called 'marks':

cut '-d ' -f2 < marks-list > marks

Similarly, we can **cut** field two (f2) of the 'class-list,' the names and save the output in the file called 'names':

cut '-d ' -f2 < class-list > names

Since we now wish to join the two files column by column, we attempt an **apropos** on the keyword "column," which yields the command **pr**:

pr (1V) - prepare file(s) for printing, perhaps in multiple columns

:

We find that:

pr -m names marks > meeting-list

finishes the task.

SOLUTION 2.

There is an easier way. An **apropos** on the keyword "join" yields:

join (1) -relational database operator

The **join** command handles this task with ease, precluding the extra steps of cutting the appropriate fields in the files:

join -o 1.2 2.2 class-list marks-list > meeting-list

The output ('-o' option) contains field two ('1.2') [file1, field2] of the first file 'class-list' and field two ('2.2') [file2, field2] of the second file 'marks-list':

Scythe, J. 99
Smith, J. 73
Smith, J. 55

Alternatively,

join -o 1.2 1.1 2.2 class-list marks-list > meeting-list

would have resulted in the student-number (first field of the first file) being included between the name and mark.

Here, this one line accomplishes the whole task. Repetitive tasks do not lend themselves to an interactive user interface; the tedium of repeatedly typing in commands becomes tiresome. Any one command or set of commands you would normally type in interactively can be placed in a file, called a shell-script, and submitted for execution. The solution to SCENARIO 1 as a shell script would look like this (saved in a file perhaps called 'make-marks'):

```
#!/bin/sh
cut '-d ' -f2 < marks-list  > marks
cut '-d ' -f2 < class-list  > names
pr -m names marks > meeting
```

There are two ways to execute this file, it can be given to a shell from **stdin**. At the prompt type:

sh < make-marks

or it can be given execute permission:

chmod a+x make-marks

and executed by typing in the file-name ('make-marks') at the shell-prompt.

UNIX was designed to be greater than the sum of its parts. Simple tools used as building blocks, with pipes acting as the "glue," create ever more sophisticated tools. The UNIX user, using these tools can become a toolmaker and create tools customized for his or her own purposes.

Note:

The key to finding the correct tool for the task at hand is judicious use of the **man -k** (a.k.a **apropos**) followed by a keyword (refer to SOLUTION 1). Remember that

UNIX has been evolving for over a decade and many tasks we encounter can be classified into a range for which there already exists a tool (and one that has been thoroughly debugged).

The paradigm of simple tools used to build sophisticated tools does not preclude the creation of sophisticated tools as an end unto themselves. **perl** (Pathologically Eclectic Rubbish Lister), written by Larry Wall, is a tool that combines the power of the interactive shell (**sh**) and the C programming language. It has established itself as a very powerful tool that can at times be used as a replacement for creating tools that would otherwise require writing a C program.

Using UNIX Tools to Design A Tool for A Researcher's Toolkit

[Editor's Note: The following description of some of the processes of creating a tool for a researcher's toolkit is intended not as a tutorial or a how-to article, but rather to demonstrate how UNIX tools can be used to design and create new tools which can be customized to one's own needs and purposes. It is our hope that this article will demonstrate why the UNIX programming environment is so important. The book *The UNIX Programming Environment*, by Rob Pike and Brian W Kernighan (NJ, 1984) is a very fine introduction to the power of UNIX tools.]

The following describes the creation of a UNIX tool. This tool is the first of several planned as part of a Researcher's Toolkit, a UNIX based toolkit for researchers to be able to handle and manipulate their data. The most important aspect of this toolkit is that it is intended to help researcher's do intellectual work that will enhance their research using these UNIX tools.

I. Form of Files

In the process of working on research on the history and development of UNIX, I felt it would be valuable to have research tools using UNIX to help with my research. I decided to see if I could create tools that would not only help me with tasks that might be drudgery, but more importantly, tools that would give me a way to have the computer help

me as an intellectual aid.

I wanted the computer to be able to help me to search through files to be able to see which files contained various ideas and then to provide me with the context of the ideas.

A) To do this I had to first create a standard format for the files I typed in. Modeling what I did on what I learned from working through *The UNIX Programming Environment* I named my files consistently, typing notes from each of my sources into individual files which I named:

unixtool.000
unixtool.001
unixtool.002
.
.
.
unixtool.041
unixtool.042
unixtool.043

B) Each of the files contained the name of the file as the top line of the file and then the name of the source, author, publication information and then the quotes I typed with page references.

For example, if I type *$ cat unixtool.001* I will get:

unixtool.001

from "The Bell System Technical Journal," July-August 1978, vol 57, no 6, pt 2

UNIX Time-Sharing System:
Preface by T. H. Crowley
pg. 1897-1898

"Programming activities under way at Bell Laboratories cover a very broad spectrum. They range from basic research on ..."

II. Form that the files took in my directory

Following is the form the files took in my directory.

*$ ls -als unixtool.0**

```
3 -rw-r--r-- 1 ronda   users    2159 Jan 31 11:54 unixtool.000
3 -rw-r--r-- 1 ronda   users    2481 Jan 26 22:48 unixtool.001
7 -rw-r--r-- 1 ronda   users    7036 Jan 26 22:49 unixtool.002
                    .
                    .
                    .
8 -rw-r--r--  1 ronda   users    7964    Mar 12 02:05 unixtool.045
8 -rw-r--r--  1 ronda   users    7985    Mar 12 02:37 unixtool.046
6 -rw-r--r--  1 ronda   users    5938    Mar 12 13:16 unixtool.047
```

III. Then I used a series of UNIX tools to search the files.

The first tool was a tool to search for a keyword (expression) in my files using the UNIX tool **grep**. I could type from the keyboard:

*$ grep -n keyword unixtool.0**

I got a listing of filenames, the line number of the keyword, and one line (with the keyword in it), separated by colons. I then made this into a shell command which I could enter from the command line and enter the keyword by using $1. The command was named **nu.grep**

```
$ cat  nu-grep
grep -n $1 unixtool.0*
```

[Note: I use **cat** to show this one line file. The file could be created with **vi** or other editor or with **cat.**]

I made certain the permissions on the file nu.grep were set so I could execute the file.

```
$ chmod 755 nu.grep
```

Thus typing from the keyboard the command nu.grep and a keyword (e.g. CTSS), I got a listing of the filename, the line number of the keyword, and one line with the keyword in it, separated by colons.

$ nu.grep CTSS

unixtool.004:14:CTSS system. This claim is intended as a complement to both UNIX
unixtool.004:15:and CTSS. Today, more than fifteen years after CTSS was born, few
.
.
.
unixtool.014:367: in turn been inspired by J. Saltzer's Runoff Program on CTSS

IV. The Value of Making a Tool

I found that the tool I was creating would help me to identify both the file name, the line number of the keyword and the line containing the keyword. This was an intellectual aid as it helped me to identify something that I might be interested in pursuing further or something that was interesting that I might not have realized.

For example, I found there were ideas highlighted through this process that were helpful for me to focus on in doing my research.

V. Finding the Context of the Reference

Next I wanted to be able to look up the context of the references that I found in the keyword search. I found I could make a tool using sed which would give me the context of the keyword. The sed tool I used was:

sed -n 'x1,x2p' filename

After locating the following line with the grep tool:
unixtool.014:367: in turn been inspired by J. Saltzer's Runoff Program on CTSS

I decided that I would look at five lines above and five lines below the line number of the keyword. The line number of the keyword was 367. Thus my sed tool was:

$ sed -n '367-5,367+5p' unixtool.041

VI - A Tool to Automate the Context Output

I wanted a tool that would automate this process by giving me first a list of the references from all the files with the keyword in it and then a printout containing the ten line context of the keyword.

I wanted to print the context of the keyword or pattern.

VII. **grep** to list the fields, **awk** to read them off, **sed** to do context

To get the context of the keyword, **awk** was used to read off the fields of the listing produced by grep. Then I used **sed**.

For example, making a new shell script **nu.grep2**:

```
$ cat  nu.grep2
grep -n $1 unixtool.0* | awk -f nu.awk | sh
```

and a shell script **nu.awk**

```
$ cat  nu.awk
  BEGIN {FS=":" }
  {print "echo", $1, $2 "----------"}
  $2 > 5 {print "sed -n ' " $2-5 "," $2+5 "p' ",$1}
  $2 < 6 && $2 > 1 {print "sed -n ' " 2 "," $2+5"p' ", $1}
  {print "echo =============================="}
```

(Make sure permissions are set on both nu.grep2 and nu.awk so that files can be executed.)

```
$ chmod 755 nu.grep2
$ chmod 755 nu.awk
```

Using the shell script **nu.grep2** with a keyword:

```
$ nu.grep2 CTSS
```

produced contextual references, including the following, for example:

```
  unixtool.004 14----------
```

"In most ways, UNIX is a very conservative system. Only a handful of its ideas are genuinely new. In fact, a good case can be made that it is in essence a modern implementation of MIT's CTSS system. This claim is intended as a compliment to both UNIX and CTSS. Today, more than fifteen years after CTSS was born, few of the interactive systems we know of are superior to it in ease of use; many are inferior in basic design."

from D. M. Ritchie, "A Retrospective," from "*The Bell System Technical Journal*," vol 57, No. 6, part 2, July-August 1978, p. 1948.

VIII. Following is the resulting tool which I called **signif**.
[Note: Make one shell script for **signif**, and another script for **contex**.]

```
$ cat signif
#!/bin/sh
#signif
#make sure that there are at least two args to the command

case $# in
   0)  echo usage:;
       echo "  " 'basename $0' keyphrase files
       exit ;;
   1)  echo usage:;
       echo "  " 'basename $0' keyphrase files
       exit ;;
esac

if egrep -n $* /dev/null
then
   contex $*
else
   echo Sorry
fi

$ cat contex
#!/bin/sh
#contex
awkfile=/tmp/nu.awk.$$
case $# in
   0)   echo usage:;
```

```
        echo "  " 'basename $0' keyphrase files
        exit ;;
    1)  echo usage:;
        echo "  " 'basename $0' keyphrase files
        exit ;;
  esac

cat>${awkfile} <<END_AWK_SCRIPT
   BEGIN {FS=":" }
   {print "echo -n", \$1 ":" \$2 ":" }
   {print "sed -n ' "  \$2 "p' ",\$1}
   {print "echo",":"}
   \$2 > 5 {print "sed -n ' " \$2-5 "," \$2+5"p' ",\$1}
   \$2 < 6 && \$2 > 1 {print "sed -n ' "2"," \$2+5 "p' ", \$1}
  {print "echo ============================="}

END_AWK_SCRIPT
  #echo executing:
  #echo " egrep -n ${key} $* | awk -f ${awkfile} | sh"
  egrep -n $* /dev/null | awk -f ${awkfile} | sh
  /bin/rm -f ${awkfile}
```

[Editor's Note: make sure you make the shell scripts executable.]

IX. Other versions of the tool:

A tool to list the whole file:

```
# signif-w
#!/bin/sh
#outputs whole files which include the keyword
awkfile=/tmp/nu.awk2.$$
case $# in
    0)  echo usage:;
        echo "  " 'basename $0' keyphrase files
        exit ;;
    1)  echo usage:;
        echo "  " 'basename $0' keyphrase files
        exit ;;
esac

cat>${awkfile} <<END_AWK_SCRIPT
      BEGIN {FS=":"}
      {print "echo ",\$1 ":" \$2":" \$3}
```

```
    {print "cat ", \$1}
    {print "echo ==================="}

END_AWK_SCRIPT
echo executing:
echo "  egrep -n ${key} $* | awk -f ${awkfile} | sh"
egrep -n $* | awk -f ${awkfile} | sh

/bin/rm -f ${awkfile}
```

A tool to list just one line references:

```
# signif-l
#!/bin/sh
# make sure that there are at least two args to the command
case $# in
    0)  echo usage:;
        echo "  " 'basename $0' keyphrase files
        exit ;;
    1)  echo usage:;
        echo "  " 'basename $0' keyphrase files
        exit ;;
esac
if egrep -n $* /dev/null
then
   echo OK
else
   echo Sorry
fi
```

Example: $ signif -l tool unixtool.0*

[Tool design and creation was done as part of an independent study with Dr. Narasimhamurthi at the University of Michigan Dearborn, in Winter 1994.]

C Program

(To grab e-mail addresses from files)

[Editor's Note: The following K&R (Kernighan and Ritchie) C program was submitted by a beginner programmer. He called the compiled version **findadd** and used it in the form:

$ *findadd < file >> addresses*]

```
------------cut here---------------
/* C program to grab E-mail addresses containing '@' */
#include <stdlib.h>
#include <stdio.h>
#define ADDMAX 100
/* maximum address expected 99 characters */

main()
{
  int c;
  char add[ADDMAX], *find_add();

  while (find_add(add) != NULL) {
    printf("%s\n", add);
  }
  exit(0);
}

char *find_add(add)
char add[];
{
  int c, i, flag;
  c = getchar();
  while (c != EOF)
  {
    /* skip whitespace */
    while (c == '  ' || c == '\t' || c == '\n')
        c = getchar();
    flag = 0;
    i=0;
    while (c != '  ' && c != '\t' && c != '\n' && c != EOF)
    {
         add[i++] = c;
/* '@' occurs in most e-mail addresses */
         if (c == '@')
```

```
            flag = 1;
          c = getchar();
      }
/* \0 ends the string */
      add[i] = '\0';
/*check if string is an address*/
      if (flag == 1)
      return(add);
   }
      return(NULL);
}
-------------cut here---------------
```

New Net Book

In honor of the 25th Anniversary of the ARPAnet and of the UNIX operating system, and the 15th Anniversary of Usenet News, I am proud to announce a net-book. This net-book provides some of the historical perspective and social context needed to understand the advance represented by the global telecommunications network. This net-book is for those who want to contribute to the care and nurture of the Net.

The Book's title is: *The Netizens and the Wonderful World of the Net: An Anthology.*

Any comments on the book would be welcome, as it is currently in draft form. We are making it available on-line as we feel it will be helpful for people, and your comments will help us to make the book more valuable.

In addition, it would be worthwhile to have the book published in a printed edition. Any suggestions towards this would be appreciated.

A draft is now available via anonymous ftp at: wuarchive.wustl.edu in the directory: /doc/misc/acn/netbook

The on-line book is also available to browse via gopher on the gopher server: gopher.cic.net in the directory (or by going through the menus): e-serials/alphabetic/a/amateur-computerist/netbook My .gopherrc entry looks like this:

Name=Netizen's Netbook
Type=1
Port=70
Path=1/e-serials/alphabetic/a/ amateur-computerist/netbook
Host=gopher.cic.net

URL: (For WWW browsers like Mosaic, lynx, cello, etc.)

gopher://gopher.cic.net/11/ e-serials/alphabetic/a/
amateur-computerist/netbook

or from my homepage at: http://www.cc.columbia.edu/~hauben/home.html under the link to its title.

Click <a href="http://www.cc. columbia.edu/~hauben/home.
html">here</A>

-Michael Hauben
hauben@columbia.edu

The Linux Movement

by Phil Hughes (phil@fylz.com)

[Editor's note: Linux is a non-commercial version of UNIX being created by volunteer developers. The *Amateur Computerist* welcomes articles about the different flavors of UNIX that might be available to amateur computerists.

We want to point out that we differ with the encouragement in the following article of a commercial future for Linux. Our experience is that commercial interests try to freeze rather than develop what they are making money off of.]

With over 13 years of experience with the UNIX operating system, I was skeptical when I read that some college student had implemented

a “free UNIX” called Linux. But I felt a duty to verify that Linux was just a toy. Now, with over a year of Linux experience, I want you to know that Linux is not a toy, it offers some advantages over commercial UNIX implementations and it has a sense of community that you won’t find with other software.

Linux is a UNIX-like operating system designed specifically to run on Intel 80386 and higher processors. It offers virtually all the functionality of other PC-based UNIX systems along with one amazing side benefit – it’s free. To understand why, you need to look at its history and its creator.

The Linux kernel was developed by a college student at the University of Helsinki named Linus Torvalds. Linus took a C and UNIX class in the fall of 1990 and got really interested in UNIX. He bought Andy Tanenbaum’s book on Minix, bought a 386 laptop and, in early 1991, installed Minix on that system.

Linus quickly figured out that Minix wasn’t what he wanted and started developing his own UNIX-like kernel. By using a C compiler and other utilities developed by the Free Software Foundation, Linus was able to put together a rudimentary UNIX-like system by January, 1992. Since that time the Linux effort has grown from the work of Linus Torvalds and a few friends into an international movement involving hundreds and possibly thousands of developers and tens of thousands of users.

What makes this effort unique is that all the development is done in public view, for free by people spread all over the world. This sort of effort has been possible because these people use the Internet to exchange information. Cooperation has been on such a level that when someone found out that Linus was still making payments on his laptop, they decided to take up a collection – over the Internet, of course – to raise money to pay off his system.

The Development Effort

The best way to describe Linux development is a successful implementation of Communism. No, this isn’t a political statement. The dictionary definition of Communism is “from each according to his ability, to each according to his need.” This is what is happening with

Linux. High-powered software engineers and programmers from different backgrounds in different countries are chipping in to write the parts of Linux they understand – from Ethernet card drivers to documentation. The result is a large-scale, commercial quality effort with no staff costs, no office costs and no marketing or distribution costs.

Because development is done in the open anyone can request a feature or point out a bug and the developer can directly respond. The result is that new software is being written at breakneck speed and bugs are being fixed in a matter of days instead of months or possibly years.

For example, last year someone posted a message to Usenet asking if Linux supported a floptical disk. (This is a disk that has a SCSI interface and supports a 20MB removable disk the size of a 3.5" floppy.) Within hours a followup message was posted that pointed out that such disk drives needed to be sent a special initialization sequence. The next followup was from the SCSI driver developer asking if anyone knew what that special sequence was. A couple of days later the developer posted a message pointing out that he had added the required code to the SCSI driver.

What do you get with Linux?

This is where most dyed-in-the-wool UNIX users expect Linux to fall short. But, it doesn't. Linux includes all the networking, development, graphics and device support of virtually any commercial version of UNIX and then some.

The reality is that you can get a complete UNIX-like system including everything you would expect in a commercial package with all the extras including editors, compilers, development software, text processing systems, shells, database systems, electronic mail and news, networking, communications programs, the X-Windows graphics system and even an assortment of games for close to nothing. Virtually all of the standard UNIX utilities come from the GNU project, an effort of the Free Software Foundation.

There are various vendors that package the Linux kernel with other freely re-distributable software. Packages vary in size from seven floppy disks to a CD-ROM chock-full of software. The prices of these packages vary from free to around $50 and are, themselves, freely re-distributable.

Is it Reliable?

It depends on who you ask. Opinions vary all over the map. But I can explain why. People who try every new Alpha release will have problems with reliability. And, unlike commercial ventures where bugs tend to be hidden behind marketing hype, everyone reading the Linux newsgroups on the Internet quickly finds out about any bugs. Rather than justify, let me just speak from personal experience.

In my "home office" which is now the editorial office for Linux Journal as well as for my consulting business, I have had a UNIX-based system for about 4 years. Up until last year it was based on AT&T System V, Release 3.2. It was running on a 386 system and proved to be reliable. Only a couple of bugs could cause an occasional crash.

Last year I replaced this system with a Linux system because I wanted support for long file names and larger disks. And, I wanted Network File System (NFS) and this would cost hundreds of dollars for my System V system. We now have three computers networked together. Two computers run Linux all the time and one runs either Linux or MS-DOS. They all talk over Ethernet. NFS works fine.

The only reason I have had to reboot any of the systems is because of a bug in the serial I/O driver which causes a port to hang occasionally under heavy use. (The system has three modems on it, talking to them at 38,400 bps.) This bug was first noticed a couple of weeks before a patch came out to correct it. I just haven't had the time to install the patch.

This bug is very similar to one that one of my clients has with their "commercial" UNIX system. The difference is that the two vendors involved (the communications board manufacturer and the UNIX vendor) both deny that they could possibly have such a bug. So it still exists in their "commercial" system after three years, two operating system upgrades and countless communications board software releases.

Is a Commercial Future Possible for Linux?

The short answer is yes. Even though Linux is free, it will make lots of money for lots of people. Here is why.

First, to run Linux you need a computer. Offered the chance to have a real UNIX-like operating system at home for free many grade a current

system to support it. Thus, hardware will be purchased because Linux is available.

Second, some companies will elect to use multiple Linux systems as an alternative to either a single computer and dumb terminals or a single computer and X-terminals. And, in some cases, Linux will replace more expensive workstations.

An example is one company where I have consulted. Their initial requirements were fairly simple. Three to five local users plus a couple of modem lines. The users would either be using a database or using standard UNIX editing and publishing tools. The solution was also fairly simple and consisted of a 386-based system running SCO Xenix and a few dumb terminals.

But, today, their requirements have grown. They need more horsepower and they need graphics capabilities at two of the desktops. To meet these requirements with traditional answers would require the purchase of at least two more copies of the UNIX or Xenix operating system, networking software, and X-windows software as well as the additional hardware. Such a purchase is beyond the budget of the company so they continue to limp along with their current configuration.

If, instead, Linux is offered as an alternative, a solution can be found. Linux is freely copyable so the operating system cost is almost zero. And Linux includes both the needed networking software (NFS and TCP/IP) and X-windows. This means that other than having to purchase an upgrade for Xenix to add the networking, all the upgrade budget can be reserved for the purchase of the necessary hardware.

The difference here is that with the Linux solution, the customer got what they needed – a system up-grade – and the computer industry, as a whole, came out ahead because new equipment was, in fact, purchased. Thus, what is called “free software” made money for some and solved the problems of others.

Finally, with a free operating system, some software developers will port their applications to this new platform making it possible for people to purchase the application. Without the free operating system the computerized solution could have been out of reach financially.

The Future

I see an interesting future for Linux. First, it will remain a hacker’s

system for some. They will get the source code, play with it and, in the process, learn a lot about computers and operating systems. For them, whether it is an individual effort or through a school, Linux will be the tool needed to pave the way into a successful career in computing.

Some of these hackers and some computer professionals will see Linux as a tool to develop a solution that otherwise couldn't be done. Interfacing Linux to a voice mail board or any other special hardware is simple. If the hardware fits in an ISA or EISA-bus slot and you have hardware documentation you can write a driver and integrate it into the Linux system. In fact, this is happening every day with new sound boards, new SCSI disk controllers and new Ethernet cards. Other hackers will probably become computer consultants.

Having all of Linux available to anyone means that you could do serious consulting in your own home town instead of having to move to the home of the developers of the software.

Linux can and has replaced X terminals. An X terminal is a graphical terminal that is designed to talk to a host computer running as an X-windows server. As Linux comes with the necessary X client software as well as the necessary networking code (TCP/IP) Linux can easily do the task. Thus an expensive, special-purpose terminal can be replaced with common, inexpensive PC hardware.

Linux includes both TCP/IP networking and the Network File System. This means Linux can offer connectivity and server capabilities to other systems at a very low cost. For example, a PC running DOS and/or MS-Windows can be cheaply added to a network of Linux (and UNIX) systems. This is because it is only necessary to add an ethernet card and the networking software (such as Sun's PC-NFS) to the single PC. There is no networking cost associated with the Linux systems.

Development continues on a program called Wine. Wine is software that will allow MS-Windows based applications to run on Linux systems. Rather than either emulate MS-Windows or just allow Linux to act as a platform for running MS-Windows, Wine runs MS-Windows applications directly by translating calls from the applications for MS-Windows services into their X-windows equivalents. While Wine is still being developed, its future looks promising. Because of its design, people will not have to purchase MS-Windows to use it. Further,

it has all the potential to run as fast, or faster, than MS-Windows on the same computer hardware.

Interested in connecting to the Internet? Many people have Internet access by using their PC as a terminal and dialing up a local service provider. [A growing number of freenets are available without charge – ed] Many of these same service providers offer both uucp and SLIP connections. [Also there are free uucp connections available in some areas – ed] Linux include both uucp, the standard UNIX-to-UNIX communications program and SLIP, a TCP/IP-like protocol designed to allow connections over dial-up serial lines. Thus, with a Linux system you can be one step closer to the Internet by establishing a uucp account. Or you could connect your Linux machine directly to the Internet by establishing a SLIP account.

For the casual user who wants to know more about multi-user systems or for the mainframe COBOL programmer who has heard about UNIX and C and wants to get his or her feet wet, Linux offers a virtually no-cost way to test the waters. In computing, the way to learn something new is to give it a try. For anyone who currently has a reasonable sized PC, Linux can be installed on a partition on an existing disk and you have your own personal UNIX (Linux) system to use as a learning tool.

Where does this lead? Does Linux replace every personal computer and every UNIX system in the world? In a word, no. But it does offer an alternative to other commercial systems, a stepping stone for some and a cost effective solution for many more.

Copyright 1994, Phil Hughes. Parts of this article appeared in the March, 1994 issue of *Puget Sound Computer User*.

The Ten Commandments for C Programmers (Annotated Edition)

by Henry Spencer

[Editor's Note: One of the important advances of UNIX was when it was recoded in C and thus became portable (not dependent on any particular computer hardware). Thus it seems appropriate to include Henry Spencer's "Ten Commandments" article on C as part of the commemoration of UNIX.]

1. Thou shalt run lint frequently and study its pronouncements with care, for verily its perception and judgement oft exceed thine.

> This is still wise counsel, although many modern compilers search out many of the same sins, and there are often problems with **lint** being aged and infirm, or unavailable in strange lands. There are other tools, such as Saber C, useful to similar ends.
> "Frequently" means thou shouldst draw thy daily guidance from it, rather than hoping thy code will achieve **lint**'s blessing by a sudden act of repentance at the last minute. De-linting a program which has never been **lint**ed before is often a cleaning of the stables such as thou wouldst not wish on thy worst enemies. Some observe, also, that careful heed to the words of **lint** can be quite helpful in debugging.
> "Study" doth not mean mindless zeal to eradicate every byte of **lint** output for — if no other reason, because thou just canst not shut it up about some things — but that thou should know the cause of its unhappiness and understand what worrisome sign it tries to speak of.

2. Thou shalt not follow the NULL pointer, for chaos and madness await thee at its end.

Clearly the holy scriptures were mis-transcribed here, as the words should have been "null pointer," to minimize confusion between the concept of null pointers and the macro NULL (of which more anon). Otherwise, the meaning is plain. A null pointer points to regions filled with dragons, demons, core dumps, and numberless other foul creatures, all of which delight in frolicking in thy program if thou disturb their sleep. A null pointer doth *not* point to a 0 of any type, despite some blasphemous old code which impiously assumes this.

3. Thou shalt cast all function arguments to the expected type if they are not of that type already, even when thou art convinced that this is unnecessary, lest they take cruel vengeance upon thee when thou least expect it.

A programmer should understand the type structure of his language, lest great misfortune befall him.

Contrary to the heresies espoused by some of the dwellers on the Western Shore, 'int' and 'long' are not the same type. The moment of their equivalence in size and representation is short, and the agony that awaits believers in their interchangeability shall last forever and ever once 64-bit machines become common.

Also, contrary to the beliefs common among the more backward inhabitants of the Polluted Eastern Marshes, 'NULL' does not have a pointer type, and must be cast to the correct type whenever it is used as a function argument.

(The words of the prophet ANSI, which permit NULL to be defined as having the type 'void *,' are oft taken out of context and misunderstood. The prophet was granting a special dispensation for use in cases of great hardship in wild lands. Verily, a righteous program must make its own way through

the Thicket Of Types without lazily relying on this rarely-available dispensation to solve all its problems. In any event, the great deity dmr who created C hath wisely endowed it with many types of pointers, not just one, and thus it would still be necessary to convert the prophet's NULL to the desired type.)

It may be thought that the radical new blessing of "prototypes" might eliminate the need for caution about argument types. Not so, brethren. Firstly, when confronted with the twisted strangeness of variable numbers of arguments, the problem returns...and he who has not kept his faith strong by repeated practice shall surely fall to this subtle trap. Secondly, the wise men have observed that reliance on prototypes doth open many doors to strange errors, and some indeed had hoped that prototypes would be decreed for purposes of error checking but would not cause implicit conversions. Lastly, reliance on prototypes causeth great difficulty in the Real World today, when many cling to the old ways and the old compilers out of desire or necessity, and no man knoweth what machine his code may be asked to run on tomorrow.

4. If thy header files fail to declare the return types of thy library functions, thou shalt declare them thyself with the most meticulous care, lest grievous harm befall thy program.

The prophet Ansi, in her wisdom, hath added that thou shouldst also scourge thy Suppliers, and demand on pain of excommunication that they produce header files that declare their library functions. For truly, only they know the precise form of the incantation appropriate to invoking their magic in the optimal way.

The prophet hath also commented that it is unwise, and leads one into the pits of damnation and subtle bugs, to attempt to declare such functions thyself when thy header files do the job right.

5. Thou shalt check the array bounds of all strings (indeed, all arrays), for surely where thou typest "foo" someone someday shall type "supercalifragilisticexpialidocious."

As demonstrated by the deeds of the Great Worm, a consequence of this commandment is that robust production software should never make use of **gets()**, for it is truly a tool of the Devil. Thy interfaces should always inform thy servants of the bounds of thy arrays, and servants who spurn such advice or quietly fail to follow it should be dispatched forthwith to the Land Of rm, where they can do no further harm to thee.

6. If a function be advertised to return an error code in the event of difficulties, thou shalt check for that code, yea, even though the checks triple the size of thy code and produce aches in thy typing fingers, for if thou thinkest "it cannot happen to me," the gods shall surely punish thee for thy arrogance.

All true believers doth wish for a better error-handling mechanism, for explicit checks of return codes are tiresome in the extreme and the temptation to omit them is great. But until the far-off day of deliverance cometh, one must walk the long and winding road with patience and care, for thy Vendor, thy Machine, and thy Software delight in surprises and think nothing of producing subtly meaningless results on the day before thy Thesis Oral or thy Big Pitch To The Client.

Occasionally, as with the **ferror()** feature of **stdio**, it is possible to defer error checking until the end when a cumulative result can be tested, and this often produceth code which is shorter and clearer. Also, even the most zealous believer should exercise some judgement when dealing with functions whose failure is totally uninteresting ... but beware, for the cast to void is a two-edged sword that sheddeth thine own blood without remorse.

7. Thou shalt study thy libraries and strive not to re-invent them without cause, that thy code may be short and readable and thy days pleasant and productive.

Numberless are the unwashed heathen who scorn their libraries on various silly and spurious grounds, such as blind worship of the Little Tin God (also known as "Efficiency"). While it is true that some features of the C libraries were ill-advised, by and large it is better and cheaper to use the works of others than to persist in re--inventing the square wheel. But thou should take the greatest of care to understand what thy libraries promise, and what they do not, lest thou rely on facilities that may vanish from under thy feet in future.

8. Thou shalt make thy program's purpose and structure clear to thy fellow man by using the One True Brace Style, even if thou likest it not, for thy creativity is better used in solving problems than in creating beautiful new impediments to understanding.

These words, alas, have caused some uncertainty among the novices and the converts, who knoweth not the ancient wisdoms. The One True Brace Style referred to is that demonstrated in the writings of the First Prophets, Kernighan and Ritchie. Often and again it is criticized by the ignorant as hard to use, when in truth it is merely somewhat difficult to learn, and thereafter is wonderfully clear and obvious, if perhaps a bit sensitive to mistakes.

While thou might think that thine own ideas of brace style lead to clearer programs, thy successors will not thank thee for it, but rather shall revile thy works and curse thy name, and word of this might get to thy next employer. Many customs in this life persist because they ease friction and promote productivity as a result of universal agreement, and whether they are precisely the optimal choices is much less important. So it is with brace style.

As a lamentable side issue, there has been some unrest from the fanatics of the Pronoun Gestapo over the use of the word "man" in this Commandment, for they believe that great efforts and loud shouting devoted to the ritual purification of the language will somehow redound to the benefit of the downtrodden (whose real

and grievous woes tendeth to get lost amidst all that thunder and fury). When preaching the gospel to the narrow of mind and short of temper, the word "creature" may be substituted as a suitable pseudo-Biblical term free of the taint of Political Incorrectness.

9. Thy external identifiers shall be unique in the first six characters, though this harsh discipline be irksome and the years of its necessity stretch before thee seemingly without end, lest thou tear thy hair out and go mad on that fateful day when thou desirest to make thy program run on an old system.

Though some hasty zealots cry "not so; the Millennium is come, and this saying is obsolete and no longer need be supported," verily there be many, many ancient systems in the world, and it is the decree of the dreaded god Murphy that thy next employment just might be on one. While thou sleepest, he plotteth against thee. Awake and take care.

It is, note carefully, not necessary that thy identifiers be limited to a *length* of six characters. The only requirement that the holy words place upon thee is uniqueness within the *first* six. This often is not so hard as the belittlers claimeth.

10. Thou shalt foreswear, renounce, and abjure the vile heresy which claimeth that "All the world's a VAX," and have no commerce with the benighted heathens who cling to this barbarous belief, that the days of thy program may be long even though the days of thy current machine be short.

This particular heresy bids fair to be replaced by "All the world's a Sun" or "All the world's a 386" (this latter being a particularly revolting invention of Satan), but the words apply to all such without limitation. Beware, in particular, of the subtle and terrible "All the world's a 32-bit machine," which is almost true today but shall cease to be so before thy resume grows too much longer.

May Day in the Morning

by Floyd Hoke-Miller

[Editor's Note: the promise of UNIX and of automation is the promise of a shorter workday for all. Since May Day is the holiday celebrating the 200 year struggle to shorten the working day, we include this poem in honor of May Day in this special issue.]

Awake my fellow workers, it's May Day and it's spring;
Let's celebrate its meaning and what our might could bring.

Let not your hearts be troubled, you're master of your fate;
So plan to use the things you've made and paradise create.

Clear the cobwebs from your mind the master class has spun
Through centuries of directing all the things you've done.

Arise you sons of toil and brush your brows of sweat
And look toward the horizon of things the OBU would get.

See how the world has prospered and who enjoys its good
Then cross your hearts forever to form one brotherhood.

[Note: The OBU (pronounced Oh Be You) is the One Big Union that the Industrial Workers of the World (the IWW or Wobblies) seek as the proper workers' organization.]

Some of the more than 500 poems that Floyd Hoke-Miller wrote are collected in the book *A Laborer Looks at Life, Then and Now, Poems from the Shop Floor*, Flint, Mi, 1984.

What is the Free Software Foundation?

[Editor's Note: The following is reprinted from **GNU's Bulletin**, June, 1993.]

The Free Software Foundation is dedicated to eliminating restriction on people's abilities and rights to copy, redistribute, understand, and modify computer programs. We do this by promoting the development and use of free software in all areas of computer use. Specifically, we are putting together a complete integrated software system named "GNU" (GNU'S Not UNIX) (pronounced "guh-new") that will be upwardly compatible with UNIX. Most parts of this system are already working, and we are distributing them now.

The word "free" in our name pertains to freedom, not price. You may or may not pay money to get GNU software. Either way, you have two specific freedoms once you have the software: first, the freedom to copy the program and give it away to your friends and co-workers; and second, the freedom to change the program as you wish, by having full access to source code. Furthermore, you can study the source and learn how such programs are written. You may then be able to port it, improve it and share your changes with others. If you redistribute GNU software, you may charge a fee for the physical act of transferring a copy, or you may give away copies.

Other organizations distribute whatever free software happens to be available. By contrast, the Free Software Foundation concentrates on development of new free software, working towards a GNU system complete enough to eliminate the need for you to purchase a proprietary system.

WHAT IS COPYLEFT?

The simplest way to make a program free is to put it in the public domain, un-copyrighted. But this allows anyone to copyright and restrict its use against the author's wishes, thus denying others the right to access and freely redistribute it. This completely perverts the original intent.

To prevent this, we copyright our software in a novel manner. Typical software companies use copyrights to take away your freedoms. We use the copyleft to preserve them. It is a legal instrument that requires those who pass on the program to include the right to further redistribute it, and to see and change the code; the code and the right become legally inseparable.

The copyleft used by the GNU Project is made from a combination of a regular copyright notice and the *GNU General Public License* (GPL). The GPL is a copying license which basically says that you have the freedoms discussed above. An alternate form, the *GNU Library General Public License* (LGPL), applies to certain GNU Libraries. This license permits linking the libraries into proprietary executables under certain conditions. The appropriate license is included in all GNU source code distributions and in many of our manuals. We will also send you a printed copy upon request.

Free Software Foundation, Inc.
675 Massachusetts. Ave.
Cambridge, MA 02139 USA
E-mail: gnu@prep.ai.mit.edu

The opinions expressed in articles are those of their authors and not necessarily the opinions of the *Amateur Computerist* newsletter. We welcome submissions from a spectrum of viewpoints.

ELECTRONIC EDITION

ACN Webpage:
http://www.ais.org/~jrh/acn/
All issues of the *Amateur Computerist* are on-line.
Back issues of the *Amateur Computerist* are available at:
http://www.ais.org/~jrh/acn/Back_Issues/
All issues can be accessed from the Index at:
http://www.ais.org/~jrh/acn/NewIndex.pdf

EDITORIAL STAFF

Ronda Hauben
William Rohler
Norman O. Thompson
Michael Hauben (1973-2001)
Jay Hauben

The *Amateur Computerist* invites submissions. Articles can be submitted via e-mail: jrh@ais.org
Permission is given to reprint articles from this issue in a non profit publication provided credit is given, with name of author and source of article cited.