A Research UNIX Reader: Annotated Excerpts from the Programmer's Manual, 1971-1986

M. Douglas McIlroy

ABSTRACT

Selected pages from the nine research editions of the *UNIX*® *Programmer's Manual* illustrate the development of the system. Accompanying commentary recounts some of the needs, events, and individual contributions that shaped this evolution.

1. Introduction

Since it began at the Computing Science Research Center of AT&T Bell Laboratories in 1969, the UNIX system has exploded in coverage, in geographic distribution and, alas, in bulk. It has brought great honor to the primary contributors, Ken Thompson and Dennis Ritchie, and has reflected respect upon many others who have built on their foundation. The story of how the system came to be and how it grew and prospered has been told many times, often embroidered almost into myth. Still, aficionados seem never to tire of hearing about how things were in that special circle where the system first flourished.

This collection of excerpts from the nine editions of the research *UNIX Programmer's Manual* has been chosen to illustrate trends and some of the lively give-and-take—or at least the tangible results thereof—that shaped the system. It is a domestic study, aimed only at capturing the way things were in the system's original home. To look further afield would require a tome, not a report, and possibly a more dispassionate scholar, not an intimate participant. The raw readings are supplemented by my own recollections as corrected by the memories of colleagues.

The collection emphasizes development up to the Seventh Edition (v7*) in 1979, providing only occasional peeks ahead to v8 and v9. Although people elsewhere in Bell Labs and in Berkeley made significant contributions before v7, it was really with v7 that the system fledged and left the research nest. V7 was the first portable edition, the last common ancestor of a radiative explosion to countless varieties of hardware. Thus the history of v7 is part of the common heritage of all UNIX systems, while v8 and v9 will be more or less foreign to many readers. Moreover v7 happened long enough ago to be viewed with reasonable perspective.

The system was already well developed before v1 appeared in 1971. And not until v4 was the system first described in public.[1, 2] The technical side of the development from a paper exercise through a fully self-supporting model on a DEC PDP-7 to implementation on a series of PDP-11s has best been told by Ritchie.[3] In his Turing Award lecture Ritchie reflected further upon the nature of the lab environment that fostered the development.[4]

To keep the size of this report in bounds current versions of manual pages are rarely shown; interested readers will be familiar with and doubtless own the manual for v7 or one of its many descendants, such as System V or BSD. Many of the pages were chosen to show things happening, and some to show foibles. Enduring central features have been correspondingly slighted. By overlooking the news (or

* For brevity I have adopted the designation v*n* for the *n*th Edition. This usage sprang from a colloquial tendency to refer to the Sixth and Seventh Editions as Versions 6 and 7, which inevitably led to the nickname v8 for the Eighth Edition. The rest followed.

nonnews) about permanent things, I have unfortunately also overlooked the news (or nonnews) about just how well Thompson and Ritchie wrought. As many people have observed, the success of the UNIX system often owes as much to what it does *not* have as what it does. In the same way, lasting truths are likely not to be found in this story about its changes. Those I have tried to tell elsewhere.[5]

In condensing the flow of events into comprehensible and compact history, I have largely overlooked, if indeed I could even have recognized, the myriad of borrowings of style and viewpoint and the continual interplay of criticism and code-dabbling that knit a cohesive *gestalt.* Without the vision of Ken Thompson, UNIX would not have come into existence; without the insight of Dennis Ritchie, it would not have evolved into a polished presence; without the imagination of Mike Lesk and popularizing touch of Brian Kernighan, it would not have acquired the extroverted personality that commands such widespread loyalty. But without any one of the people whose contributions are cited here, neither UNIX nor the work of others in the group would be the same.

1.1. The People

Up to v7 the *dramatis personae* were relatively few. Never officially recognized in any organization chart, the group coalesced voluntarily. Most of these original contributors are still with the Computing Science Research Center or its divestiture-induced clone at Bell Communications Research.

Ken Thompson began the construction from the ground up based on a file system model worked out with Ritchie and Rudd H. Canaday. He made processors for B, *bas,* and Fortran besides the operating system proper, and personally installed customized versions of UNIX for early clients as far away as Georgia. With Ritchie, Ken wrote the first shell and piles of utilities including *ed, roff, sort, grep, uniq, plot, sa,* and *dd.* He is also known for creative decompiling of mystery code. As if all this weren't enough, he has at the same time written circuit-optimizing tools, switching and network code, C compilers, and the basic software for several special-purpose machines, especially the chess champion, Belle.

Dennis M. Ritchie, best known as the father of C, joined Ken very early on. Dennis contributed basic notions such as *fork-exec* and set-userid programs. They jointly wrote the *fc* compiler for Fortran IV. The first debugger *db* and the definitive *ed* were Ritchie's, as was the radically new stream basis for IO in v8 and much networking software. With Steve Johnson he made UNIX portable, moving the system to an Interdata machine (v7). The names of Ritchie and Thompson may safely be assumed to be attached to almost everything not otherwise attributed.

Joe (Joseph F.) Ossanna, with the instincts of a motor pool sergeant, equipped our first lab and attracted the first outside users. Joe's *nroff* and *troff* indelibly shaped UNIX word processing and typesetting.

Bob (Robert) Morris stepped in wherever mathematics was involved, whether it was numerical analysis or number theory. Bob invented the distinctively original utilities *typo,* and *dc-bc* (with Lorinda Cherry), wrote most of the math library, and wrote *primes* and *factor* (with Thompson). His series of *crypt* programs fostered the Center's continuing interest in cryptography.

Doug (M. Douglas) McIlroy exercised the right of a department head to muscle in on the original two-user PDP-7 system. Later he contributed an eclectic bag of utilities: *tmg* for compiler writing, *speak* for reading text aloud, *diff,* and *join.* He also collected dictionaries and made tools to use them: *look* (v7, after a model by Ossanna), *dict* (v8), and *spell* (v7).

Lorinda L. Cherry collaborated with Morris on *dc-bc* and *typo.* Always fascinated by text processing, Lorinda initiated *eqn* and invented *parts,* an approximate parser that was exploited in the celebrated Writer's Workbench™, *wwb*(v8).

Steve (Stephen C.) Johnson's yacc reduced *Al (Alfred V.) Aho's* expertise in language theory to practice. Upon that base Steve built the portable C compiler that was used to port the system itself and to evaluate candidate instruction sets for unbuilt machines. Johnson made the first *spell,* worked on computer algebra, and devised languages for VLSI layout.

Lee E. McMahon's linguistic insight fostered the characteristic text-processing—as distinct from text-formatting—capabilities of the system. He wrote *comm, qsort, sed,* the current *grep,* and the concordance-builders *index* for English and *cref* for C. After an early fling with *cu* and an influential laboratory switching system done with Condon, Morris, Thompson, *Chuck (Charles B.) Haley* and Cherry, Lee became the prime software architect for Sandy Fraser's Datakit® switch.

Brian W. Kernighan, expositor par excellence, coined the name UNIX, popularized the tools philosophy,[6] wrote the best tutorials, and became a prolific inventor of specialized ''little languages'': *ratfor, eqn, awk* and *pic.* The Center's typesetting guru since the untimely death of Joe Ossanna, Brian produced the current ''device-independent'' version of *troff* and postprocessors for particular hardware.

Steve (Stephen R.) Bourne arrived at the time of v6, bringing Algol 68 with him. His definitive programs, the debugger *adb,* and the ''Bourne shell,'' although written in C, looked like Algol 68: Steve wrote `DO-OD` and `BEGIN-END` instead of `{` and `}`. Bourne also contributed macro constructs to the UNIX Circuit Design System (see 4.3).

Mike (Michael E.) Lesk, with a prescient market instinct, made text formatting accessible to the masses with the generic macros *−ms,* which were to *troff* what a compiler is to assembly language. He rounded out *−ms* with the preprocessors *tbl* for typesetting tables and *refer* for bibliographies. He also made the *lex* generator of lexical analyzers. Eager to distribute his software quickly and painlessly, Mike invented *uucp,* thereby begetting a whole global network. Over the years, often helped by *Ruby Jane Elliott,* he initiated fascinating on-line audio, textual, and graphical access to phone books, news wire (*apnews,* v8), *weather* (v8), and UNIX instruction (*learn,* with Kernighan, v7).

Stu (Stuart I.) Feldman implemented, with *Andy (Andrew D.) Hall, the efl* preprocessor to sugar Fortran with PL/I-ish syntax. He wrote the *f77* Fortran compiler single-handedly and invented the famous *make.* A man of taste and culture, Stu exhibited both in his underground classic on UNIX style.[7]

Peter J. Weinberger has moved effortlessly among number theory, databases, languages, and networking. Weinberger boasts the middle initial of *awk,* the IO library for *f77,* and a famous visage (see FACED, page 14). In v8 and v9 appeared an unbounded precision arithmetic package *mp,* a fast factoring program *qfactor,* a B-tree library *cbt,* and a new code generator for C, all Peter's work. Above all, his network file system bound a stable of machines together into a logically homogeneous system (v8).

Sandy (A. G.) Fraser devised the Spider local-area ring (v6) and the Datakit switch (v7) that have served in the lab for over a decade. Special services on Spider included a central network file store, *nfs,* and a communication package, *ufs.* Datakit, a ''central office'' for data communication, gave added impetus to research in distributed computing. Fraser undertook the Unix Circuit Design System (see CDL in section 4.3) to support his hardware projects.

Joe (Joseph H.) Condon, physicist and circuit designer extraordinaire, although not usually listed as an ''author'' of UNIX, is an indispensable presence among the group. He wrote much of the Unix Circuit Design System, including sophisticated wire-routing algorithms. He designed superfast specialized machines, including the chess machine Belle (with Thompson), domesticated real telephone switches to our laboratory environment for the experiments of McMahon and others, and made unusual connections to the telephone system that, among other things, let our lab computer detect and announce incoming voice calls.

Al (Alfred V.) Aho's unstintingly given insight into language theory and algorithms shows up in programs by many people, such as *yacc, lex, cc,* the Writer's Workbench, and the screen editor *sam,* as well as in his own *awk, egrep,* and *fgrep.*

Greg (Gregory L.) Chesson pursued all aspects of computer-to-computer communication: multiplexing with *mpx* (v7), flow-controlled channels with *con* (v7) and *dcon,* and protocols, including one used by *uucp.* The first kernel- and user-level software for Datakit was his.

Many other people contributed. By the time of v4 the role of provider to the by then sizable clientele within Bell Labs had been assumed by *Berk (Berkley A.) Tague* and his UNIX Support Group, thereby guaranteeing the future of the system. Shortly thereafter the Programmer's Workbench project undertook to

adapt the system to support large software efforts. Their concern with administrative tools led to somewhat differently flavored utilities such as *find* (v5), *cpio* (v7), and *sccs. Joe (Joseph F.) Maranzano* of the USG and *Dick (Richard C.) Haight* of PWB became *de facto* adjunct members of the research group. Haight contributed *find, cpio,* and *expr,* all in v7. *Ted (Theodore A.) Dolotta* of PWB did much to refine the manual.

Some USG and Computer Center people eventually joined the Computing Science Research Center to bring order to our zoo of equipment: *Andy (Andrew R.) Koenig* conceived and built *asd,* the automatic software distribution system that keeps v9 current across about 50 different computers, and *snocone,* a preprocessor to sugar the syntax of Snobol into a structured language. *Fred (Frederick T.) Grampp's* unrivaled practical experience in computer security shows up in subtle countermeasures now routinely used in our systems. A thorough security audit program of his ultimately became the administators' tool *quest. Ed (Edward J.) Sitar,* with devotion worthy of Ossanna, saw to it that the hardware actually worked, keeping twenty machines housed and powered, and suppliers on their mettle.

Tom (Thomas B.) London and *John F. Reiser* ported v7 to the VAX and introduced paging. Their V32 system, as filtered through Berkeley, became the progenitor of almost all research UNIX systems. Reiser later contributed a peerless compile-and-execute *bitblt* for the Teletype 5620.

People who joined the research center after v7 led development in new directions. *Bart N. Locanthi* designed the bitmapped terminal known variously as ''jerq,'' ''Blit,'' and Teletype 5620. He programmed it, too: from graphic primitives, such as the crucial *bitblt,* up to a multiterminal maze war game. *Rob Pike* supplied a multiprogramming system, host-terminal communications, mouse control, and support for overlapping virtual terminals (*mpx,* later *mux*). Pike's system fostered fascinating programs by many people, of whom I shall mention only a few (v8). Pike himself built visual editors, culminating with *sam* (v9), which went well beyond the command capability of *vi*—seemingly with no more mechanism than the venerable *ed.* (Rob's simplifying touch is also visible in the shell, in *p* for paginating, and in his recasting of Chesson's remote execution software, all in v8.)

The Blit terminal attracted *Mark S. Manasse* who (with Pike) invented *lens,* an algorithmically ingenious bitmap magnifier, and *tek4014,* a disarmingly faithful simulator of Tektronix terminals. *Luca Cardelli* contributed an *icon* builder, annoying *crabs* that devour screen images, and the catchy *vismon* that posts icons of the senders of incoming mail. *Tom (Thomas A.) Cargill* did a monumental multiview debugger *pi,* thoroughly exercising the object-oriented capabilities of *Bjarne Stroustrup's* C++. *Tom (Thomas J.) Killian* made a font editor *jf* and programs to save and print bitmap images (*blitblt, thinkblt*) that led in turn to *can,* a comprehensive suite of laser-printer software built from the ROM up. For the UNIX system itself, Killian invented the important `/proc` file system that contains core images of all running processes.

Dave (David L.) Presotto tamed networks. His *upas* brought some order to a Babel of mail addresses and his *ipc* primitives provided a common basis for communication and remote file access via Internet, Ethernet, and Datakit. *Bill (William T.) Marshall,* who shares principal Datakit software responsibility with McMahon, wrote basic protocol code. These protocols merit unusual confidence: their correctness was mechanically verified by *Gerard Holzmann.*

Andrew G. Hume wrote *proof* to put *troff* on your screen (v8, begun by Locanthi), parts of the UNIX Circuit Design System (v9), *mk* to supplant *make* (v9), and a remote *backup* service (v9). *Norman Wilson* shares with Ritchie the honors of reigning guru. He has been instrumental in porting v8 to a Cray and in rationalizing system configuration procedures. Waging a personal battle against entropy, Wilson rectified countless infelicities, glitches, and blunders in the software and the manual, always making things shorter and simpler as he did so.

1.2. The Manual

During the system's first two years one literally had to work beside the originators to learn it. But the stock of necessary facts was growing and the gurus' capacity to handle apprentices was limited. As they wished to spread the good news about their marvelous system, a manual became a necessity. Ritchie one

day set forth the first ‘‘man page’’ in a format that has stood the test of time. The terse, yet informal, prose style and alphabetic ordering encouraged accurate on-line documentation: it was not a big deal to decide how and where to describe changes as they happened. The format was popular with initiates who needed to look up facts, albeit sometimes frustrating for beginners who didn’t know what facts to look for.

The absence of any ‘‘logical’’ grouping of facilities was a deliberate result of discussion. (As encyclopedists have always known, the relationships among knowledge are too various to force into rational linear order.) Retrievability and honesty were the prime concerns. The refreshing BUGS notes served as a constant reminder of areas for improvement.

The first manual was duplicated for a very small coterie. In order to channel queries directly to the horses’ mouths, authorship was attributed to individuals. Later, as authorship diffused, on the principle of ‘‘You touched it last; it’s yours,’’ authorship was attributed only in the segregated chapter of unofficial ‘‘user maintained programs.’’ Beginning with v7, this back-of-the-bus chapter was reserved for games.

As the system was elaborated, peer pressure in the research group caused rough places to be smoothed, vague ideas to be sharpened, and feeble programs to be extinguished. Most details of the constant questioning and experimentation during the early period of rapid change are long forgotten, as are hundreds of transitory states that were recorded in the on-line manual. From time to time, however, a snapshot was taken in the form of a new printed edition. Quite contrary to commercial practice, where a release is supposed to mark a stable, shaken-down state of affairs, the very act of preparing a new edition often caused a flurry of improvements simply to forestall embarrassing admissions of imperfection.

1.3. The Events

Among the more memorable minirevolutions that the system experienced were

- The appearance of pipes elevated standard-in-standard-out design to the status of a ‘‘philosophy’’ (v3). Old software was gradually brought into line (see SORT in Section 2).
- *Grep* (Thompson, v4) ingrained the tools outlook irrevocably. Already visible in utilities such as *wc* (Ossanna, v1), *cat*, and *uniq* (v3), the stream-transformation model was deliberately followed in the design of later programs such as *tr* (McIlroy, v4), *m4* (Kernighan and Ritchie, v7), *sed* (McMahon, v7), and a flurry of language preprocessors.
- Conversion to C (v4) made basic abstractions clearer. Assembly language and magic constants gradually declined from the status of the ‘‘real truth’’ (v4) to utterly forgotten (v8).
- The novel style of *eqn* influenced the design—even the conception—of a still growing generation of special purpose languages (v5).
- The move away from PDP-11s caused a further push for portable abstractions. The main visible symptom was a proliferation of include files (v7).
- The Bourne shell almost overnight drove out the simple old shell. A PWB shell had made shell programming useful; the Bourne shell made it an essential part of UNIX programming (v7).
- Mike Lesk’s *uucp* gave operational meaning to the phrase UNIX community’’ (v7). News now travels electronically among users all over the world; and technical collaborations proceed between distant locations almost as easily as within one building.
- Direct network connections among our computers began with Sandy Fraser’s Spider network and became widespread with Datakit (Chesson and Ritchie, v7). Datakit and Ritchie’s streams (v8) made possible Peter Weinberger’s network file system, Andy Koenig’s automatic software distribution, and Dave Presotto’s connections to diverse networks. As a result ‘‘the’’ research machine is no longer identifiable; users can—and do—work on one or more of two dozen computers simultaneously.
- Bitmapped terminals operating under Rob Pike’s ‘‘jerq’’ software caused a quantum jump in personal multiprogramming and inspired intriguing new programming styles (v8).

The early editions came in quick succession. Later the interval between editions increased for several reasons. First, most of the system was mature and stable. Second, from the standpoint of the manual, much research was subliminal. For example, the biggest system changes from v6 to v7 to v8—portability and streams—barely affected the manual. Third, the need for timely printing diminished as other organizations became responsible for distribution. And fourth, as the system grew to encompass facilities beyond any individual's ken,* the task of organizing an ever-growing manual for printing became increasingly daunting.

The UNIX lab has always been an exciting place to work. As I recorded this summary, I recalled vivid individual moments when new ideas or startling combinations of old ideas flashed through the lab, when programming met theory and *vice versa*, and when advances on many simultaneous fronts built upon and reinforced one another. I was forcibly reminded over just how wide a spectrum of activities and interests each member of the group has ranged and how freely and without fanfare collegial help has flowed among them. The primary dividends to the participants have been the fun of doing, the joy of accomplishment and the satisfaction of seeing one's handiwork used. Intellectual proprietorship and physical ownership count for little; there's more than enough of both for everyone.

2. Primary Commands

CAT (v1 page 16†)

Cat is probably the oldest and best-known of all distinctively UNIX utilities. The current (v9) description scarcely differs from that in v1. On the PDP-7 there had been a program *pr* for copying a single file to the terminal. Having been subsumed by *cat*, *pr* was retired and its name was recycled.

Since *cat* was the prototypical filter, people were tempted to pile on options for other functions. Thus *cat* was pressed into service to block output into 512-byte chunks. That in turn led to `cat -u` (v7) to turn the feature off. This dismal admission that byte streams might not always be pure had been overcome by the time of v8. In other circles the chastity of *cat* was violated more severely; see Pike's essay on `cat -v`.[8]

CP (v1 page 17, v5 page 18)

The war-horse utility *cp* and its close relative *mv* originally worked on lists of pairs. Such lists, however, could not be generated by the shell's `*` convention. All too often mistyped lists clobbered precious files. Consequently both utilities were promptly cut back to handle just one from-to pair (v2). At the same time *mv* was generalized to move files to a named directory. Strangely *cp* picked up only a BUG note suggesting the feature. By the time of v3 both had converged to their present forms, although an unexplained option `-t` intruded briefly in v5. What seems natural in hindsight was not clear cut at the time: the final conventions arose only after long discussions about how properly to handle file permissions and multiple files. In fact the discussion is not yet closed. Whether and how to recurse on directories is still debated: v7, v8, and v9 each offered a different way to do it.

SORT (v1 page 19, v5 page 20)

This mainstay utility began as a ''user-supported progam'' in chapter 6 of v1 (Thompson). Upon query to the author it turned out that the ''wide options'' announced there involved altering the source. The first official options appeared in v4. Expanding ever since, the load of options reached 17 in v9 (McIlroy and *John P. Linderman* of the Computer Technology Research Lab). The program originally sorted in core; pressure of real use soon forced it to spill to disk.

---

* Ken's ken was probably the last to saturate. At the time of v5, shell accounting once revealed that Thompson had used 102 distinctly named programs in the course of a week. Nobody else came close.

† The *cat* page from v1 is reproduced on page 16 of this report.

The first design, typical of pre-pipe days, had an argument to name the output (see GREP below):

```
sort input output
```

During the pipeline revolution Thompson modified *sort* to be usable as a filter (v4). Unlike most utilities, though, *sort* did retain an output-naming convention because it was so often used to sort a file in place, which couldn't be done with `>`.

When he extended *sort* to handle multiple files, Thompson invented a special name "`-`" for the standard input (v5). The convention caught on and soon infected many other commands. As a property of particular commands and not of the system as a whole, "`-`" itched naggingly. The itch went unscratched until Ritchie, at Pike's suggestion, installed `fd` special files synonymous with already open file descriptors (v8). The stubborn disease remains, however.

A bug note in *join*(1) declares, "The [field-specification] conventions of *join, sort, comm, uniq, look* and *awk*(1) are wildly incongruous." Although these programs are often used together, they remain, like American weights and measures, sturdily eccentric.

MAIL (v1 page 21, v7 page 22)

Electronic mail was there from the start. Never satisfied with its exact behavior, everybody touched it at one time or another: to assure the safety of simultaneous access, to improve privacy, to survive crashes, to exploit *uucp*, to screen out foreign freeloaders, or whatever. Not until v7 did the interface change (Thompson). Later, as mail became global in its reach, Dave Presotto took charge and brought order to communications with a grab-bag of external networks (v8).

Despite the turbulent evolution of *mail*, to this day a simple postmark is all that it adds to what you write. Old UNIX hands groan at the monstrous headers that come from latter-day mailers and at the fatness of their manuals.

ECHO (v2 page 23)

Echo, seemingly the simplest of utilities, originated with Multics, where it was used to test the sanity of the shell. The present version arose as a finger exercise in C programming (McIlroy, v2). Then it turned out to be useful, a mainstay of shell scripts.

For a while *echo* was complemented by *prompt* (never documented), which did the same thing without a newline. Eventually *prompt* was displaced by `echo -n` (Ritchie, v7). Meanwhile a minor echo-amplification industry arose in some quarters. Imported versions of *echo* with elaborate syntax came and went in a midnight vendetta; for a time it was the least stable of all commands. Ritchie calmed the altercation with a Solomonic but un-UNIX-like option to switch between two competing syntaxes (v8). Not surprisingly, the more elaborate choice has never been needed in any shell script in `/bin` or `/usr/bin`. The whole episode inspired McIlroy's parable, "The Unix and the Echo," quoted in Kernighan and Pike[9] page 79.

GREP (v4 page 24)

Grep, generally cited as *the* prototypical software tool, was born when Ken Thompson was asked how to search for patterns in a file that was too big for the editor. Thompson promptly liberated his regular expression recognizer and christened it after the *ed* command `g/re/p`. It was an instant hit and soon entered our language as a verb.

The success of *grep* suggested other utilities. One of these, *gres* for global substitution, evolved into the stream editor *sed* (McMahon, v7). *Grep* also inspired Al Aho to apply his encyclopedic knowledge of language theory to make the very general *egrep* and the highly specialized *fgrep* (v7). Over the years Aho honed *egrep* into an awesomely performing program, simple on the outside but the highest of tech on the inside.

The v4 manual page for *grep* is typical of earlier editions. Options were described in running text along with the basic usage; the more readable convention of always displaying options separately, no matter how few a command might have, gained ground only slowly. In typical early style the synopsis permits an output argument as well as an input argument—poor human engineering, because of the disastrous consequences of invoking `command input output` as if were `command input1 input2`. This dangerous syntax was expunged from most commands by v7, but at least one fossilized instance survives in BSD 4.3.

3. Programming

3.1. The Shell

SH (v1 pages 25-27, v3 pages 28-33, v4 pages 34-36)

The name and the general outline of the ''shell'' originated in Multics, but for UNIX the shell had to be pared back to basics to fit in an 8K user space. There wasn't even enough room to do * name expansion; that task was handed off to another program *glob*, signifying ''global'' (Ritchie, v1). (*Glob*, written in B, was the first piece of mainline UNIX software to be done in a higher level language.)

The shell read commands from the same standard input as did programs that it invoked. Thus commands and data were interleaved in command files, or ''runcoms,''* now usually called shell scripts. There was no way to mark the end of embedded data files; programs that buffered their input would consume input not intended for them. Consequently programs that were likely to read from shell scripts, especially *sh* and *ed*, were made to read their input one character at a time. It was impossible to pipe into a shell script because the standard input was already dedicated to the script. For the same reason a program in a shell script could not take input from a terminal except when given the terminal's real name. None of these problems was addressed until the Bourne shell solved them all at once (v7).

A shell notation for composing programs into pipelines accompanied McIlroy's proposal for pipes. This new idea, with its curious syntax, took almost a page to describe (v3). The syntax was reformed when, to avoid the embarrassment of describing it in a big public talk, Thompson proposed the appealing infix `|`. Thus naturalized, pipelines became describable in just four sentences in v4.

The UNIX shell gave up the Multics idea of a search path and looked for program names that weren't file names in just one place, `/bin`. Then in v3 `/bin` overflowed the small (256K), fast fixed-head drive. Thus was `/usr/bin` born, and the idea of a search path reinstated.

GOTO, : (v2 pages 37-38)

Goto manipulated the standard input to give the illusion of a programmable shell (Thompson, v2). The associated *:* command, which thanks to ASCII collating sequence boasted the first page in the manual, was a big nop.

Other flow-of-control programs were *if*, to execute a command conditionally (Ritchie and Thompson, v2), and *exit* (Thompson v2). With the luxury of bigger computers, Bourne made a genuinely programmable shell and abolished these clever but clumsy tricks (v7). Nevertheless *:* survives as a built-in shell command and *test* has inherited the boolean capabilities of *if*.

3.2. System Calls

STAT (v1 page 39, v4 page 40, v7 pages 41-42)

Stat is one of very few original system calls to have changed at all, mostly because through it user code glimpses system tables. In v4 group permissions appeared and file sizes went from 16 to 24 bits. The

* *Runcom*, a program that could run a short script of commands in the background, was the closest thing MIT's CTSS had to a callable shell. A vestige of the name survives in the boot script, `/etc/rc`.

size change was tough; it meant rebuilding every existing file system, and the longer addresses compelled some trickery to avoid sacrificing space or performance on smaller files.

The snapshots of *stat* also show how the style of description evolved from assembly language (v1) through C (v4) to a more abstract and portable form with declarations hidden in include files (v7). The traditional sense of what constituted the real system interface eroded very slowly. Even though almost all programs were written in C, assembly language held pride of place in the manual through v6. Banished to a footnote in v7, assembly language did not disappear completely until v8, twelve years after the birth of C.

PIPE (v3 page 43)

The basic redirectability of input-output made it easy to put pipes in when Doug McIlroy finally persuaded Ken Thompson to do it. In one feverish night Ken wrote and installed the *pipe* system call, added pipes to the shell, and modified several utilities, such as *pr* and *ov* (see 5.1 below), to be usable as filters. The next day saw an unforgettable orgy of one-liners as everybody joined in the excitement of plumbing. Pipes ultimately affected our outlook on program design far more profoundly than had the original idea of redirectable standard input and output.

All programs placed diagnostics on the standard output. This had always caused trouble when the output was redirected into a file, but became intolerable when the output was sent to an unsuspecting process. Nevertheless, unwilling to violate the simplicity of the standard-input-standard-output model, people tolerated this state of affairs through v6. Shortly thereafter Dennis Ritchie cut the Gordian knot by introducing the standard error file. That was not quite enough. With pipelines diagnostics could come from any of several programs running simultaneously. Diagnostics needed to identify themselves. Thus began a never quite finished pacification campaign: a few recalcitrant diagnostics still remain anonymous or appear on the standard output.

The first implementation of pipes used a single file descriptor for both reading and writing (v3). The modern scheme, with two file descriptors, came in the next edition.

INTR (v1 page 44), SIGNAL (v4 page 45)

In v1 there was a separate system call to catch each of interrupt, quit, and two kinds of machine traps. Floating point hardware (v3) brought another and no end was in sight. To stop the proliferation, all traps were subsumed under a single system call, *signal* (v4). The various traps were given numbers, by which they were known until symbolic names were assigned in v7. We have not been fully weaned yet: the numbers are still needed in the shell *trap* command.

A simple unconditional *kill* was available to terminate rogue programs in the background (v2). In v5 *kill* was generalized to send arbitrary signals. Never, however, was the basically unstructured *signal-kill* mechanism regarded as a significant means of interprocess communication. The research systems therefore declined to adopt the more reliable, but also more complex, Berkeley signals.

In general, research UNIX systems, out of a belief that asynchrony is nasty, have provided less overt support for it than have some of their relatives. Thus when multiprocess coordination is unavoidable, as for receiving mail, curious synchronizing tricks with dummy files have been resorted to. The short-lived multiplexor (Chesson, v7), then Ritchie's `/dev/pt` named pipes and *select* (adapted from Berkeley) lent some support for these special needs (v8); Presotto's *ipc* code goes still further (v9).

The research systems remain standoffish about unbridled parallelism; the recent facilities for asynchrony are used only by cognoscenti. Nevertheless users of UNIX systems are probably more at home with parallel computing—as structured by pipes—than is almost any other user community.

STTY (v2 pages 46-47), IOCTL (v7 page 48)

Ioctl is a closet full of skeletons. The *ioctl* story began with *stty* (v2), the primary use of which—setting modes upon logging in—was unexceptionable. Trouble set in when other programs began to use it. These programs would work for their owners on their owners' terminals, but could fail frustratingly in other

settings. Thus was the slippery slope to *curses* first glimpsed.

Stty accumulated features gradually, and often incompatibly. Eventually it was rechristened *ioctl* to keep abreast of plans for the corporate standard release 3.0 (v7). This faceless name, with no intrinsic meaning, quickly acquired more than enough. It was somehow exempt from the ethos of simplicity that kept the lid on new system calls. All kinds of functions were piled onto *ioctl*. The interface varied bewilderingly from function to function and from system to system. Documented willy-nilly throughout chapter 4 and sometimes only in source code, its true dimensions can never be appreciated.

3.3. Standard IO

PRINTF (v4 page 49)

Output formatting was originally left up to programmers (or to Fortan). Although v1 included conversion routines like *atof*, the *printf* routine that Ritchie had long since devised for BCPL did not arrive until the C change (v4). Formatted input was even slower in coming: Mike Lesk's portable IO library that included *scanf*, as well as *gets* and *ungetc*, did not become official until v7.

PUTC (v1 page 50), STDIO (v7 page 51)

Buffered IO was, and still is, a necessary evil. A rudimentary buffering package with *getc*() and *putc*() in v1 required the user to supply buffers and manage file descriptors. This scheme persisted until Ritchie's *stdio* reconciled the buffering package with Lesk's portable IO, hid the dependence on file descriptors, and eliminated per-character function calls. In one clean sweep *stdio* made C programs easily portable. In the ANSI draft standard for C *stdio* enjoys equal status with the language proper.

3.4. Languages

Almost everybody in the group has done languages. Thompson and Ritchie built assemblers from scratch. On the tiny PDP-7 the assembler was supplemented by *tmg*, Doug McIlroy's version of Bob McClure's compiler-compiler. Using it Thompson wrote B, the evolutionary link between Martin Richards's (Cambridge University) BCPL and C. The PDP-11 assembler, a desk calculator *dc*, and B itself were written in B to bootstrap the system to the PDP-11. Because it could run before the disk had arrived, *dc*—not the assembler—became the first language to run on our PDP-11. Soon revised to handle arbitrary-precision numbers (v1), *dc* was taken over by Bob Morris and Lorinda Cherry. It now ranks as the senior language on UNIX systems.

CC (v2 page 52)

As a testbed for floating-point routines in v1 Thompson wrote *bas*, a Basic-like interpreter. It survived as long as we used PDP-11s. V2 saw a burst of languages: a new *tmg*, a B that worked in both core-resident and software-paged versions, the completion of Fortran IV (Thompson and Ritchie), and Ritchie's first C, conceived as B with data types. In that furiously productive year Thompson and Ritchie together wrote and debugged about 100,000 lines of production code.

Conversion to C made UNIX, already elegant and capable, into a system also intelligible, pliable, and ultimately portable. It elicited a flood of utilities and made it easier to refine the kernel. As the compiler evolved, the system benefited too: better object code meant speedups and space savings across the board. More than once an overgrown kernel was squeezed back into place by attending to the compiler.

Portable utilities written in C spread easily to other computing environments at Bell Labs. Gradually users became able to deal with systems from several manufacturers on the same terms: programs like *ls*, *cp*, and above all *sh* worked similarly everywhere. With the human, if not the machine interfaces already established, the ultimate transition to UNIX systems proper on mainframes was remarkably gentle.

YACC (v3 page 53)

With *yacc* Steve Johnson reduced to practice Al Aho's expertise in language theory (v3). *Yacc*, abetted by Mike Lesk's *lex* (v7), stimulated a language industry. Among the better known *yacc*-based processors are

eqn for typesetting equations (Kernighan and Cherry, v5)

ratfor, which provided C-like syntax for Fortran (Kernighan, v6)

bc for arbitrary-precision computation (Morris and Cherry, v6)

m4, a general macroprocessor (Kernighan, v7)

apl, Iverson's language (Thompson, v7)

struct, convert Fortran to Ratfor (*Brenda S. Baker*, v7)

f77, a Fortran 77 compiler (Feldman and Weinberger, v7)

awk, a pattern-directed file-processing language (Aho, Weinberger and Kernighan, v7)

pcc, a portable C compiler (Johnson, v7)

pic for typesetting line drawings (Kernighan, v8)

ideal, a constraint-based language for typesetting line drawings (*Chris (Christopher J.) Van Wyk*, v8),

C++, an ''object-oriented'' extension of C (Bjarne Stroustrup, v8)

hoc, a C-like ''desk-calculator'' language (Kernighan and Pike, v8)

grap for typesetting graphs (Kernighan and *Jon L. Bentley*, v9)

Several of these languages are *tours de force*: *eqn* for its insightful syntax, *bc*-*dc* for its unique variable-precision math library, *struct* for finding both structure and theorems in undisciplined code, *ideal* for enlisting symbolic computation in the service of drafting. *Yacc*, by eliminating much drudgery of compiler-writing, made possible the extensive experimentation that underlies these novel languages. It is no exaggeration to say that without *yacc* some would never have been undertaken, and many would not have evolved into more than mere demonstrations.

4. Applications

4.1. Text Processing

Among the early justifications for the research activity that produced the UNIX system was the potential for text-processing. Thus the first significant application program was *roff*, which printed the manuals for v1, v2, and v3 (Thompson and Ritchie) and attracted the first outside client, the patent department at Bell Labs.

Thereafter Joe Ossanna became the driving force behind UNIX text formatting software. His macro-based, trap-driven *nroff* appeared in v2. When Graphics Systems, Inc., announced an inexpensive typesetter with ASCII paper tape input, Ossanna sprang for one, replaced the tape reader with a wire to the computer, and modified *nroff* for multiple fonts and proportional spacing. *Voilà*, *troff*. It blew the manufacturer's mind, and touched off a flurry of homemade documents in flamboyant layouts—good enough, however, to fool referees into suspecting that the manuscripts had been published before.

Ossanna's ultimate intent, that macros should foster higher-level typsetting languages, was finally realized with the invention of the `-ms` macros (Lesk, v7, but dating from 1976).

FORM (v1 page 54), TYPO (v3 page 55)

''Text processing'' means considerably more than mere text formatting. The elegant *form* letter generator was written by Thompson from Morris's original on CTSS (v1). Augmented by a special editor *fed* (Cherry, v2) *form* provided a genuine personal database. McMahon, Morris, and Cherry together collected

a substantial corpus of text and studied it statistically.[10] Some of their tools, particularly *uniq* (Thompson, v3) and *comm* (McMahon, v4) became staples. Out of that work came the remarkable *typo*, which spotted typing errors by statistical inference (Morris and Cherry, v3). Eventually Steve Johnson's *spell*, a virtuoso demonstration of tools in use (see page 126 of Kernighan and Plauger),[6] shouldered *typo* aside (v5) and spurred McIlroy to engineer a sophisticated version (v7).

Form was really a macroprocessor with persistent memory. The long tradition of macros at Bell Labs assured that others would appear: *m6* by McIlroy, Morris and Andrew D. Hall, then *m4* (Kernighan and Ritchie). And macros of course were central to the text formatters, *roff*, *nroff*, and *troff*.

OV (v3 page 56)

Ov did multicolumn formatting cheaply and elegantly (Ossanna, v3). One printed a narrow column with every other page offset one-half a page width and then ran the output through *ov*, which or-ed pairs of pages together. The following 4-column wonder was shown off on the first day of pipes:

```
roff file >ov>ov>
```

which means in modern language, `roff file | ov | ov`. *Ov* was ultimately killed off by `pr -`*n* and features built into *nroff* (v5).

WWB (v8 page 57)

Cherry's work on approximate parsing and Aho's on fast pattern search turned out to be just the right foundation for an English style-appraiser suggested by Prof. William Vesterman of Rutgers. That in turn was elaborated into Writer's Workbench by *Nina (Antonina H.) MacDonald* and others in the Human Performance Engineering Department (v8). WWB attracted unusual attention from the popular press, including the New York Times and the Today show.

4.2. Navigation

SKY (v4 page 58)

By some quirk of providence, many members of the group have been fascinated by navigation, geodesy, and astronomy. The first celestial program was Ossanna's satellite predictor, *azel*, which had guided the ground station for Telstar (v2). Morris's, then Thompson's *sky* programs predicted everything else, giving daily voice and mail announcements like, ''Eclipse of the moon at 8:42 PM.'' McIlroy's *map*, intended to display the earth on dozens of projections (v7), was used by McMahon to portray the heavens. Upon databases of maps, stars, cities, airports, and weather were built route planners for car (*blitmap*, Lesk and Elliott, v8) and plane (Thompson) and an astronomer's informant (*scat*, Pike, v9).

WWV (v8 page 59)

Already in v1 were routines, *cal* (Thompson) and *ctime* (Ritchie), that knew calendrical facts well beyond any immediate system need. In the 1980s, as the lab's work spread across a network of machines, the many aberrent clocks became intolerable. Andy Koenig made the most of a cheap and jittery receiver for the National Bureau of Standards broadcast time standard *wwv*, arranging for individual clocks to adjust to the standard smoothly, without backdating. Later Condon installed and Thompson programmed a robust filter for a raw time receiver on an almost uninterruptable computer called ''the rock'' (v9).

4.3. Design Automation

CDL (v7 pages 60-63)

Although most users do not encounter the UNIX Circuit Design System, it has long stood as an important application in the lab. Originated by Sandy Fraser and extended by Steve Bourne, Joe Condon, and Andrew Hume, UCDS handles circuits expressed in a common design language, *cdl*. It includes programs

to create descriptions using interactive graphics, to lay out boards automatically, to check circuits for consistency, to guide wire-wrap machines, to specify combinational circuits and optimize them for programmed logic arrays (Chesson and Thompson). Without UCDS, significant inventions like Datakit, the 5620 Blit terminal, or the Belle chess machine would never have been built. UCDS appeared in only one manual, v7.

5. Communication

TSS (v2 page 64)

The members of the research group had no desire to isolate themselves from the rest of the Bell Labs computing community. Nor could they at first justify the purchase of equipment such as line printers and tape drives, which cost more than their whole computer. Thus, besides dial-up access, which was a *sine qua non*, communication with other machines was a necessity. A 2000bps link provided remote job entry to the GECOS system at the Bell Labs computer center (*opr*, Thompson, v2). GECOS guru Charlie Roberts contributed *tss* to exploit the link for remote login and interactive file transfer (v2). Similar programs to communicate with IBM mainframes followed.

Unlike most of the basic facilities of the system, which distilled years of well-established practice, computer-to-computer communication has been a subject of almost continuous experimentation. The need to connect to foreign systems exacerbates the problem of making a coherent model for communication. Ken Thompson more than once returned to ground zero, building experimental communication-based systems from scratch.

NFS (v7 pages 65-66), DCON (v8 page 67)

Sandy Fraser, aided by Jane Elliott, made our first local-area net, Spider, and its far-more-than-local-area successor based on the Datakit switch. Spider provided *nfs*, a remote network file store for a dozen minis (v7). Greg Chesson wrote remote-connection programs for direct machine-to-machine links. These programs, ultimately to become *dcon* (Chesson) and *rx* (Pike) for remote login and execution, were adapted to Datakit as soon as it became available. More recently they, and connection programs for other networks, have been reworked to exploit a general server mechanism by Ritchie and Presotto (v8).

Datakit connections sparked a version of the standard IO library that gave access to remote as well as local files (Fraser and Priscilla Lu). Ritchie's work on IO streams eventually made possible Weinberger's network file system, `/n`, which provided remote access through the kernel. Not to be confused with the central file server for Spider, Weinberger's file system is simply the union of all the files across all communicating machines. There is no manual page about how to use it because there is nothing to say. Remote file systems are ''mounted'' locally so that remote accesses look just like local accesses.

6. Security

SU (v1 page 68, v8 page 69)

Whether the system was actually run securely or not, considerable care has always been taken to assure that it is possible to do so. Permissions and Ritchie's patented set-userid mechanism were already supported in v1. From Cambridge, England, came the idea of password encryption that went into v3.

Trojan Horse tricks and countermeasures were discovered in an ongoing game that has been recounted by Morris and Fred Grampp.[11] Notice, for example, the removal of *login*(1) to chapter 8 and the intrusion of `/etc/` into the synopsis for *su* (Grampp, v8). These fillips defeated the old chestnut of leaving programs named *login* or *su* lying around in hopes of capturing a password typed by an unwary system administrator. Other subtle features of the modern *su*: dot is excluded from the shell search path and the burglars' favorite shell variable `IFS` is reset.

CRYPT (v3 page 70, v9 page 71)

Morris's first file encrypter appeared in v3 with the explicit intent to stimulate code breaking experiments. Stimulate it did. Morris himself broke *crypt* by hand. Later Ritchie automated the cryptanalysis using a method of Jim Reeds (Berkeley). Completed with an editor interface, a new crypt went public in v7. It also succumbed to an attack by Reeds and Weinberger—and fortunately, too: more than one person who locked data in *crypt* and threw away the key has been rescued by code breakers.

But the still arduous process of code-breaking is not the easiest way to attack *crypt.* A simpler gambit is to catch a system administrator off guard and install a Trojan horse in *crypt* itself to snatch every new secret as it passes by. Thus the very presence of *crypt* may have just the opposite effect on security from what was intended.

Even if *crypt* were perfectly safe, it would be unwise to encrypt files of lasting value. It is too easy to lose the key, either inadvertently or deliberately. Consequently *crypt* has been demoted to the games chapter (Grampp, v9).

7. Curiosities

NUMBER (v6 page 72)

This filter that converts numbers to check-writing form was whipped up by Thompson (v6) to preprocess input to *speak,* which converted English to phonemes for a Votrax speech synthesizer (McIlroy, v3).* *Number* was the glue with which Thompson fashioned a talking desk calculator:

```
dc | number | speak
```

The talking calculator could also be reached from an audio shell, through which, with some effort, the whole system could be run from a Touch-Tone phone.

DSW (v1 page 73)

The nostalgically named *dsw* was a desperation tool designed to clean up files with unutterable names. This had been done on the PDP-7 by a program with console switch input; hence the sobriquet ''delete from switches.'' It survived from v1 until it was displaced by `rm -i` (Ritchie, v7).

DD (v5 page 74)

Originally intended for converting files between the ASCII, little-endian, byte-stream world of DEC computers and the EBCDIC, big-endian, blocked world of IBM, *dd* was endowed with an appropriately bastard syntax (Thompson, v5). Pike has noted a cultural quirk. Much as families perpetuate the quaint sayings of children, users are wont to invoke *dd* with the JCL-ish formula, `dd if=input of=output`, or perhaps with `cat input | dd of=output`, but rarely with the elementary utterance `dd <input >output`.

FACED (v9 page 75)

Among the novelties inspired by bitmapped terminals was Pike's and Presottos's face server, which provides pictures of all users. The face server, like Weinberger's network file system and Killian's `/proc` directory of running core images, is a process that functions as a file system: it interprets file names upon *open*—in this case as connections to a remote repository—and delivers data upon *read.*

By far the most popular application of the face server is Luca Cardelli's *vismon,* which announces incoming messages with pictures of their senders.

---

* The UNIX lab was then the only place in the world where one could hear arbitrary text uttered by a machine on the spur of the moment. Visitors did a double-take upon being greeted by name. One hapless lounger listening to the fun from outside the lab door fled at a sudden sentence, ''Stop messing around and get some work done.''

Enigmatic images resembling that on the FACED page have frequently and inexplicably appeared on organization charts, posters, magazine covers, construction fences, and even a water tower.

8. Front matter

Titles and introductions (pages 76-96)

This chronological set of front matter comprises all title pages and prefaces plus samples of introductions and tables of contents.

In a bow to user-friendliness, Ritchie added ‘‘How to Get started’’ to the introduction for v3. At the typographic watershed where the manual was converted from *roff* to *troff* there appeared the thinnest, least forbidding UNIX Programmer’s Manual of all time, v4 (pages 88-89). For v6 and v7 Lorinda Cherry prepared a pocket reference, familiarly known as the ‘‘Purple Card,’’ which is not shown. Thompson gathered a set of *Documents for Use with UNIX*, also not shown, to accompany v6. Kernighan expanded the collection for the v7 manual, but nobody has had the ambition to modernize ‘‘Volume 2’’ since. Doug McIlroy compiled a glossary for the trade book form of v7[12] and all later editions (page 94). Cherry prepared the topical table of contents for v9 (page 96).

Page 97 touches on software by Pike *et al.* for the Teletype 5620, the terminal of choice for v8 and v9. With a megabyte of memory and downloadable programs, the 5620 is a computer system in its own right. This fact is reflected in the organization of chapter 9 into subchapters parallel to the usual chapters: commands, system calls, subroutines, data layouts, games, etc. The 5620 software outweighs that of the original v1 in bulk, and offers a similar number of ‘‘primitive’’ functions. These facilities radically change the feel, but not the principles, of the UNIX system. Significant as it is, the 5620 software has been deliberately slighted in these readings about the common descent of UNIX, because the work is still tied to one kind of hardware and because it strikes off in new directions that more properly belong to a sequel.

Tables of contents (pages 98-114)

9. Acknowledgments

Many of the people mentioned have willingly provided fresh recollections, unearthed sources, and commented on drafts. I am grateful for the generous help of L. L. Cherry, J. H. Condon, R. J. Elliott, B. W. Kernighan, A. R. Koenig, L. E. McMahon, R. Pike, D. M. Ritchie, K. Thompson, P. J. Weinberger, and Norman Wilson. The selection of materials and whatever inaccuracies or oversights remain mine and mine alone.

References

1. D. M. Ritchie and K. Thompson, "The UNIX Time-Sharing System" in *Proceedings of Fourth Symposium on Operating System Principles*, 7, pp. 1-9, Yorktown Heights NY (October 1973).
2. D. M. Ritchie and K. Thompson, "The UNIX Time-Sharing System," *Comm. of the ACM*, 17, pp. 365-375 (July 1974). Reprinted in [13].
3. D. M. Ritchie, "The Evolution of the UNIX Time-sharing System," *AT&T Bell Laboratories Technical Journal*, 63, part 2, pp. 11-17 (1984). UNIX special issue. Reprinted in [13], Vol. II, 11-17.
4. D. M. Ritchie, "Reflections on Software Research," *Comm. of the ACM*, 27, pp. 758-760 (1984).
5. M. D. McIlroy, "The UNIX Success Story," *UNIX Review*, 4, pp. 32-42 (October 1986).
6. B. W. Kernighan and P. J. Plauger, *Software Tools*, Addison-Wesley, Reading MA (1976).
7. Stuart I. Feldman, "An Architecture History of the UNIX System" in *USENIX Conference*, Salt Lake City (Summer 1984).
8. R. Pike and B. W. Kernighan, "Program Design in the UNIX System Environment," *AT&T Bell Laboratories Technical Journal*, 63, part 2, pp. 1598-1601 (1984). UNIX special issue. Reprinted in [13], Vol. II, 21-24.
9. Brian W. Kernighan and Rob Pike, *The UNIX Programming Environment*, pp. Prentice-Hall, Englewood Cliffs, NJ (1984).
10. L. E. McMahon, L. L. Cherry, and R. Morris, "Statistical Text Processing," *Bell System Technical Journal*, 56, part 2, pp. 2137-2154 (1978). UNIX special issue. Reprinted in [13], Vol.I, 227-244.
11. F. T. Grampp and R. H. Morris, "UNIX Operating System Security," *AT&T Bell Laboratories Technical Journal*, 63, part 2, pp. 1649-1672 (1984). UNIX special issue. Reprinted in [13], Vol. II, 18-28.
12. Bell Telephone Laboratories, *UNIX Programmer's Manual*, Vol. 1, Holt, Rinehart and Winston, New York (1983).
13. AT&T Bell Laboratories, *UNIX System Readings and Applications*, Vols. I and II, Prentice-Hall, Englewood Cliffs, NJ (1986).