The Restoration of Early UNIX Artifacts

Warren Toomey
wtoomey@staff.bond.edu.au
School of IT, Bond University

Abstract

UNIX turns 40 this year: many happy returns! Four decades is a vast period for the computing industry: systems from the 1970s now seem rudimentary and primitive. And yet, the early versions of UNIX were epitomes of sophisticated concepts packaged into elegant systems. UNIX' influence has been so powerful that it reverberates down to affect us in the 21st century.

The history of the development of UNIX has been well documented, and over the past decade or so, efforts have been made to find and conserve the software and documentation artifacts from the earliest period of UNIX history. This paper details the work that has been done to restore the artifacts from this time to working order and the lessons learned from this work.

1 Introduction

In 2009, the UNIX[1] operating system celebrates the 40th anniversary of its creation. In the middle of 1969, after AT&T's withdrawal from the Multics project, a number of researchers at AT&T's Bell Labs began the design and development of a simpler operating system which would be named "UNIX" [10]. Led primarily by Ken Thompson and Dennis Ritchie, but with many other colleagues involved, Bell Labs' UNIX would combine several leading-edge concepts (multitasking, a process model, a hierarchical filesystem) and some new concepts (I/O redirection, pipes, portable implementation in a high-level language) to become an elegant and sophisticated system. The 7th Edition of UNIX released in 1979 (and its 32-bit port called "32V") would become the ancestors to all of the hundreds of UNIX-derived systems that now exist[2] including AIX, Solaris, Apple's Darwin kernel and the various open-source BSD systems. UNIX and the C language would exert a significant influence on the computing industry in the 1980s and 1990s, and see the creation of such vendor-neutral standards as IEEE 1003 POSIX, ISO/IEC 9945, ANSI C and ISO/IEC 9899.

While the history of UNIX has been well-documented [5, 7, 8, 10], there was a time when the actual artifacts of early UNIX development were in great danger of being lost forever. This has been rectified in the last decade with the collection of a significantly large number of old UNIX systems. Software, however, is simply a collection of zeroes and ones if it is not able to run, and a lot of work has been done to bring these old UNIX systems back to life.

The restoration of a software artifact to working order brings with it a wealth of difficulties: documentation is missing or incomplete, source code is missing leaving only the binary executables, or conversely the source exists but the compilation tools to reconstruct the executables are missing. The restoration of an operating system to working order presents its own issues, as the system needs a suitable hardware environment in which to run, a suitable filesystem and a set of system executables to initialise the system and make it useful.

This paper presents four case studies in software restoration: two early UNIX kernels, the earliest extant C compiler, and a set of executables and source code fragments from 1972. The case studies highlight the above issues in restoration, and outline the approaches taken to resolve the issues.

2 TUHS and the UNIX Archive

In 1995 the UNIX Heritage Society (TUHS)[3] was founded with a charter to preserve, maintain and restore historical and non-mainstream UNIX systems. TUHS has been successful in unearthing artifacts from many important historical UNIX systems; this includes system & application source code, system & application executables, user manuals & documentation, and images of populated filesystems.

The proliferation of UNIX variants and the longevity of minicomputer systems such as the VAX and the PDP-11 made TUHS' task of collecting old UNIX systems and their documentation relatively straightforward. Quite quickly the society had gathered such early system as 6th and 7th Edition UNIX, 32V, System III, the BSDs, and some early commercial variants such as Ultrix-11.

The building of an archive of early UNIX systems was initially quite dubious, legally. Most of TUHS' mem-

bers were covered by various UNIX source code licenses from AT&T or USL, but not every license covered the sum of material stored in the archive. TUHS began a process of lobbying SCO,[4] then owners of the UNIX source code, for some license which would allow access to the material in the archive. With the immense assistance of Dion Johnson at SCO, in 1998 a cheap hobbyist license was made available which gave source-code access to the various PDP-11 UNIX systems, 32V and System III [11]. And in 2002, after much lobbying from TUHS, the PDP-11 UNIX systems and 32V were placed under an open-source BSD-style license.

System	Released	Features
1st Edition	Nov 1971	multitasking, multiuser, hierarchical filesystem
2nd Edition	June 1972	support for memory management on the PDP-11/45
3rd Edition	Feb 1973	pipes and C
4th Edition	Nov 1973	rewritten in the C language
5th Edition	June 1974	first version made available outside Bell Labs
6th Edition	May 1975	ported to multiple platforms
7th Edition	Jan 1979	large filesystem support, the stdio library, many new commands
32V	May 1979	port of 7th Edition to the 32-bit VAX platform

Table of Early UNIX Releases

For a while, it seemed that the archaeology of UNIX stopped somewhere around 1974. The source code and binaries for 5th Edition UNIX existed, but not the files for the manuals; conversely, only the 4th Edition UNIX manuals existed, but not the source code nor any binaries for the system. At the time, Dennis Ritchie told us that there was very little material from before 4th Edition, just some snippets of code listings. Then, around the mid-90s, Paul Vixie and Keith Bostic "unearthed a DECtape drive and made it work", and were able to read a number of DECtapes which had been found "under the floor of the computer room" at Bell Labs. These tapes would turn out to contain a bounty of early UNIX artifacts.

Two issues immediately arose with the extraction of the tapes' contents: what format were the tapes in, and the interpretation of the timestamps on the files therein. 7th Edition introduced the *tar(1)* tape archive format; before *tar(1)* there was *rkd(1)* used in 1st Edition to dump an RK05 disk's raw filesystem to nine DECtapes, *tap(1)* used from 1st to 4th Edition to dump selected parts of a filesystem to DECtape, and *tp(1)* used from 4th to 6th Edition to dump selected parts of a filesystem to tape. Fortunately, the formats for *tap(1)* and *tp(1)* are documented, and it was simple to write programs to extract the files from both archive formats.

Timestamp interpretation is a much more difficult issue to solve, as Dennis Ritchie noted in a private e-mail:

> The difficulty of [timestamp] interpretation [is due] to epoch uncertainty. Earliest Unix used a 32-bit representation of time measured in 60ths of one second, which implies a period of just over 2 years if the number is taken as unsigned. In consequence, during 1969-73, the epoch was changed several times, usually by back-dating existing files on disk and tape and changing the origin.

For each DECtape unearthed and read, the epoch used can only be determined by looking through the contents of the tape and determining where the files should be placed in the known history of UNIX development. We will consider four of the artifacts unearthed in reverse chronological order.

3 The Nsys Kernel: 1973

One of the DECtapes was labelled 'nsys', and Dennis Ritchie's initial e-mail on the tape's contents noted:

> So far as I can determine, this is the earliest version of Unix that currently exists in machine-readable form. ... What is here is just the source of the OS itself, written in the pre-K&R dialect of C. ... It is intended only for PDP-11/45, and has setup and memory-handling code that will not work on other models).
>
> I'm not sure how much work it would take to get this system to boot. Even compiling it might be a bit of a challenge. ... Best wishes with this. I'd be interested to hear whether anyone gets the [system] to run.

Initial interpretation of the timestamps in the archive led us to believe that the files were dated January 1973, but after analysing the contents and their placement in the history of UNIX, we now believe that the files are dated August 1973, just before the release of the 4th Edition of UNIX in November 1973.

Ritchie's innocuous comments on "how much work it would take to get this system to boot" seemed to be an implicit challenge, and I began the restoration task soon after receiving the tape's contents. My tools were a working 5th Edition UNIX compiler and environment running on top of the SIMH PDP-11 simulator [2], along with my own Apout emulator (see below).

Restoration work of this kind generally involves consulting existing documentation (in this case, the 4th Edition Programmers Manual and John Lions' Commentary on 6th Edition UNIX [6]), interpreting the few available source code comments,[5] single-stepping the machine code in the simulator, and intuiting what corrections need to be made based on the system's behaviour.

As predicted by Ritchie, the compilation was a bit of a challenge due to the changes in the C language between 1973 and 1974: sub-structures defined within a structure were delimited by parentheses in 'nsys', but by curly braces in 5th Edition. However, the main issue was an incompatibility of the filesystem layout between 'nsys' and 5th Edition: the *filsys* structure in 5th Edition has an extra field, `s_ronly`, and the *inode* structure in 5th Edition also has an extra field, `i_lastr`.

One last stumbling block was found which prevented the 'nsys' kernel from booting via the 5th Edition's bootstrap code. While the 5th Edition kernel starts execution at location 0, the 'nsys' kernel starts execution at location 2. With a small amount of code transposition, the 'nsys' kernel was able to boot on top of a 5th Edition filesystem and behave normally.

There is one last comment to make about the 'nsys' kernel. Although the 4th Edition of UNIX (dated November 1973) has the *pipe(2)* system call, and an internal Bell Labs meeting in January 1973[6] notes the existence of pipes, the 'nsys' kernel has *pipe(2)* listed but not implemented in the system call table. 3rd Edition UNIX was the last version written in PDP-11 assembly language. During 1973, while the assembly version was still being developed, the system was also rewritten in the new C language. After discussions with Ritchie, it seems most likely that pipes were implemented in the assembly version of UNIX first, and added to the C version after most of the core functionality had been reimplemented.

4 1st and 2nd Edition Binaries: 1972

Two of the DECtapes read by Bostic, Vixie and Ritchie were labelled 's1' and 's2'. Ritchie's initial notes were:

> s1: I haven't cracked this yet.
>
> s2 (tap format): This is not source, but a dump of (parts of) `/bin`, `/etc`, `/usr/lib`, and bits of a few other directories.

The contents of the 's2' tape, being in *tap(1)* format with timestamps in 60ths of a second, were easy enough to extract but not to date. Most of the files were executables in early UNIX 'a.out' format with a mixture of 0405 and 0407 signatures.[7] This, along with the names and contents of the executables, indicate that the tape was written at a time around the 2nd Edition of UNIX: files are dated from January 1972 through to February 1973.

Having a set of early UNIX executables is nice, but having them execute is much nicer. There were already a number of PDP-11 emulators available to run executables, but there was a significant catch: with no 1st or 2nd Edition UNIX kernel, the executables would run up to their first system call, and then "fall off the edge of the world" and crash.

Fortunately, there was a solution. As part of my overall early UNIX restoration work, I had written a user-level emulator for UNIX a.out binaries called 'Apout'. [8] Like the Wine emulator for Windows, Apout simulates user-mode PDP-11 instructions, but system calls invoked by the TRAP instruction are caught by Apout and emulated by calling equivalent native-mode system calls.

Apout had already been written to run a.out executables from 5th, 6th and 7th Edition UNIX, 2.9BSD and 2.11BSD. Dennis Ritchie had luckily scanned in his paper copy of the 1st Edition Programmers Manual, and I obtained a paper copy of the 2nd Edition Programmers Manual from Norman Wilson. With these in hand, the work to add 1st and 2nd Edition executable support was possible, but not trivial. The PDP-11/20 used by 1st Edition UNIX required an add-on module known as the KE11A Extended Arithmetic Element to perform operations such as multiply or divide. The KE11A needed to be emulated, and I was able to borrow some code written for SIMH to use in Apout. There were other issues to solve, not the least being discrepancies between the UNIX Programmers Manual and the expected behaviour of the system calls being used by the executables (for example, seeks on ordinary files were in bytes, but seeks on device files were in 512-byte blocks). Eventually, a faithful emulation of the 1st and 2nd Edition UNIX executing environment was made, allowing executables such as the early shell, `ls`, `cp`, `mv`, `rm` and friends to run again:

```
# chdir /
# ls -l
total   32
236 sdrwr-   1024 May 23 14:24:12 bin
568 sdrwr-    512 May 18 06:40:28 dev
297 sdrwr-    512 May 16 03:07:56 etc
299 sdrwr-    512 May 19 07:33:00 tmp
301 sdrwr-    512 May  5 23:10:38 usr
# chdir /bin
# ls -l
total  215
374 sxr-r-   2310 Jan 25 17:20:48 ar
375 lxr-r-   7582 Jun 29 17:45:20 as
377 sxr-r-   2860 Mar  6 12:23:38 cal
378 sxr-r-    134 Jan 16 17:53:34 cat
385 sxr-r-    160 Jan 16 17:53:36 cp
   . . .
```

For those unfamiliar with the output from 1st Edition UNIX *ls(1)*, the first column shows the file's i-node num-

ber. The *s/l* character in the next column indicates if the file is 'small' or 'large' (4096 bytes or more), the *d/x* indicates if the entry is a directory or executable (there being only one executable bit per file), and the two *rw* entries show the file's read/write status for owner and other (there being no groups yet).

The 's1' DECtape (noted by Ritchie as "not cracked yet") proved to be much more intriguing and at the same time extremely frustrating. A block-level analysis showed source code in both C and PDP-11 assembly, none of which appeared to be for the UNIX kernel. There was no apparent archive structure, nor any i-nodes. All of the DECtape appeared to be used, and this led me to conclude that 's1' was one of the middle DECtapes in the set of nine used when *rkd(1)* dumped an RK05 disk's contents block-by-block out to tape. With the first tape containing the disk's i-nodes missing, the 's1' tape was merely a collection of 512-byte blocks.

In places, some source files were stored in contiguous blocks, and the few comments inside allowed me to recover the source for such early programs as *ls(1)*, *cat(1)* and *cp(1)*. But for the most part, the arbitrary placement of blocks and lack of comments stymied further file recovery. Setting things aside for a while, I worked on other projects including a tool to compare multiple code trees in C and other languages.[9] It took nearly two years to realise that I could use this tool to match the fragments from the 's1' tape to source files in other early UNIX systems such as the 5th Edition. Independently and concurrently, Doug Merritt also worked on identifying the source fragments from the 's1' tape, and we used each other's work to cross-compare and validate the results. In the end, the 's1' tape contained source code for the assembler `as`, the Basic interpreter `bas`, the debugger `db`, the form letter generator `form`, the linker `ld`, and system utilities such as *ar*, *cat*, *chmod*, *chown*, *cmp*, *cp*, *date*, *df*, *getty*, *glob*, *goto*, *if*, *init*, *login* and *ls*.

5 Early C Compilers: 1972

Two other DECtapes recovered by Ritchie contain source code for two of the earliest versions of the original C compiler:[10]

> The first [tape] is labeled 'last1120c', the second 'prestruct-c'. The first is a saved copy of the compiler preserved just as we were abandoning the PDP-11/20; the second is a copy of the compiler just before I started changing it to use structures itself. ...
>
> The earlier compiler does not know about structures at all: the string "struct" does not appear anywhere. The [later] compiler does implement structures in a way that begins to approach their current meaning. ... Aside from their small size, perhaps the most striking thing about these programs is their primitive construction, particularly the many constants strewn throughout.
>
> With a lot of handwork, there is probably enough material to construct a working version of the last1120c compiler, where "works" means "turns source into PDP-11 assembler".

Interpreting the timestamps on the tapes gives a date of July 1972 for the 'last1120c' compiler and a date of December 1972 for the 'prestruct-c' compiler. Again, Ritchie's note that "there is probably enough material to construct a working version of the last1120c compiler" was taken as an implicit challenge to bring these compilers back to life. But there was a "chicken and egg" problem here: both compilers are in such a primitive dialect of C that no extant working compilers would be able to parse their source code. Good fortune was, however, on my side. Not only did the 's2' tape contain early UNIX system executables, but hidden away in `/usr/lib` were executables named *c0* and *c1*: the two passes of an early C compiler. It seemed likely that these executables running on the Apout emulator would be able to recompile the 'last1120c' compiler, and so it turned out to be. And, using the newly-compiled executables *c0* and *c1* built from 'last1120c', the compiler was able to recompile itself.

The 'prestruct-c' compiler presented a much harder problem: some of the source code was missing, particularly the hand-coded tables used to convert the compiler's internal intermediate form into final assembly code. This seemed at first an insurmountable problem, but after exploring several dead ends a solution was found. The hand-coded tables from the 'last1120c' compiler were borrowed and, with a small number of changes, the hybrid source code was able to be compiled by the 'last1120c' compiler, and then to compile itself.

6 1st Edition UNIX Kernel: 1971

Alas, with the above DECtapes fully explored, there seemed to be no earlier UNIX artifacts except Ritchie's fragmentary code listings on paper. Then in 2006, Al Kossow from the Computer History Museum unearthed and scanned in a document by T. R. Bashkow entitled "Study of UNIX", dated September 1972 [1]; this covers "the structure, functional components and internal operation of the system". Included along with the study was what appeared to be a complete listing of an assembly version of the UNIX kernel. A second document unearthed contained the handwritten notes made in preparation of Bashkow's study; dates within this document

indicate that the analysis of the UNIX kernel began in January 1972, implying that the kernel being studied was the 1st Edition UNIX kernel.

The idea of restoring the listing of the 1st Edition kernel to working order seemed impossible: there was no filesystem on which to store the files, no suitable assembler, no bootstrap code, and no certainty that the user mode binaries on the 's2' tape were compatible with the kernel in the listing; for a while the listing was set aside. Then early in 2008 new enthusiasm for the project was found, and a team of people[11] began the restoration work.

The team began by scanning and OCR'ing the kernel listing, creating a separate text document for each page. Each document was manually cross-checked for errors, then combined and rearranged to recreate the original assembly files. The next task was to find a suitable assembler for these files. We found after some trial and error that the 7th Edition assembler could be made to accept the 1st Edition kernel files, but we had to make a few changes to the input files and postprocess the output from the assembler. This raised the issue: how much change can be made to an original artifact when restoring it to working order? We chose to keep the original files intact and create (and annotate) a set of "patch" files which are used to modify the originals for the working restoration. Thus, the original artifact is preserved, and the changes required to bring it back to life are also documented.

The kernel listing and the 1st Edition Programmers Manual indicated that the system required a PDP-11/20 with 24 Kbytes of core, RF-11 and RK03 disks, up to 8 teletypes on a DC-11 interface, and a TC-11 DECtape device. The SIMH PDP-11 simulator was configured to provide this environment. With the kernel assembled into an executable binary, we next had to recreate the boot sequence. Luckily, we were able to side-step this issue by commanding the SIMH simulator to load the executable directly into the system's memory, initialise some registers and start execution at the correct first instruction.

With fingers crossed, the 1st Edition UNIX kernel was started for the first time for several decades, but after only a few dozen instructions it died. We had forgotten that this early system required the KE11A co-processor. Restoration halted while KE11A support was added to SIMH using the PDP-11/20 processor manual [3]. On the next attempt the kernel ran into an infinite loop, and after studying the code we guessed that the loop on the paper listing was missing a decrement instruction. With this fixed the kernel was able to run in "cold UNIX" mode, which had the task of writing a minimal filesystem onto the RF-11 device along with a number of device files, the *init* program and a minimal command shell.

The filesystem's format was hand-checked using the format description from the Programmers Manual and determined to be valid, so we pressed on to try booting the kernel in "warm UNIX" mode. After another couple of kernel source errors were fixed, the 1st Edition UNIX kernel was able to run the *init* program, output a login prompt and invoke a shell on successful *root* login. This was a rather limited success: the early UNIX shell had no metacharacter support (no * expansion), and *echo* was not a built-in. So, with only *init* and *sh* on the filesystem, nothing could be done at the shell prompt. We had several executables from the 's2' tape, but the 1st Edition kernel only supported those with the 0405 header; we took the decision to modify the kernel source to add support for the 0407 executables. Then, with the existing RF-11 filesystem and the Programmers Manual, a standalone program was written to create and populate a filesystem image with the 's2' executables. Now the kernel could be booted to a usable system with nearly all of the documented 1st Edition system tools, editors, document processing tools and programming languages.

We now had the system running in single-user mode, but the kernel listing showed that it normally ran in multi-user mode: there was only one process in memory at any time; all other processes were kept on swap. Our attempts to configure the system for multi-user mode simply resulted in the system 'hanging' at boot time. Again, a hardware configuration deficiency was found: the SIMH simulator had no support for the DC-11 serial interface device. Using the 1972 PDP-11 peripherals handbook [4] we added DC-11 support to SIMH, and finally brought 1st Edition UNIX up into multi-user mode. The restoration of the kernel was complete.

While the C language did not exist for the 1st Edition of UNIX, there was a C compiler in existence by the time of the 2nd Edition [9]. We had the 'last1120c' C compiler source code and working executables, but to run them the restored kernel & filesystem needed to be modified to provide a 16 Kbyte process address space and 16 Kbyte swap areas on the disk. With these modifications the restored system was able to run the C compiler, and the C compiler was able to recompile itself.

7 Lessons Learned

From successfully completing the restoration of the above UNIX software artifacts, we have learned several lessons about the craft of software restoration:

Restoration is only possible with adequate documentation. This not only includes user manuals, but manuals for system calls, libraries, file and storage structures, documentation on how to configure and boot systems, and technically solid hardware manuals.

Comments and documentation are often misleading. Though documentation is required, it is not always accurate or complete. A restorer must treat all documentation as dubious, and look for independent sources

which corroborate each other.

Restoration is only possible with a working environment. All software requires an environment in which to execute. User mode executables require a CPU to run instructions, some memory, and access to a set of system calls: this is what emulators like Wine and Apout provide. Operating systems require a suitable hardware environment, real or simulated. If the correct environment cannot be recreated, restoration cannot proceed.

Restoration from source requires a working compilation environment. Source code is tantalizingly close to being executable, but without a compiler or assembler that can recognise the source and produce the executable, the source code is just a collection of bits.

Any restoration will affect the purity of the original artifact. It is next to impossible to recreate the environment required to run a software artifact older than a decade, as the hardware and supporting software often no longer exist. It is therefore usually necessary to modify both the software artifact and the recreated environment so that they are compatible. When this occurs, it is imperative to preserve the purity of the original artifact, and copy & "patch" it to perform a working restoration.

Simulated hardware is infinitely easier to obtain, configure and diagnose than real hardware. Tools like SIMH can be configured to simulate a vast combination of different CPUs, memory sizes and peripherals. They can be easily single-stepped, and register & memory dumps can be taken at any point. This allows the diagnosis of errant software behaviour much more quickly than with real hardware.

Never underestimate the 'packrat' nature of computer enthusiasts. Artifacts that appear to be lost are often safely tucked away in a box in someone's basement. The art is to find the individual who has that box. The formation of a loose group of interested enthusiasts, TUHS, has helped immensely to unearth many hidden treasures. And professional organisations such as the Computer History Museum are vital if the computing industry wants to preserve a detailed record of its past.

In conclusion, the restoration of some of the earliest software artifacts from the development of UNIX has been time-consuming, frustrating but most importantly extremely rewarding. It is now more important than ever to begin to preserve computing history, not as a collection of "stuffed exhibits", but to keep old systems running as was intended by their designers.

8 Acknowledgments

None of the work described in this paper would have been possible without the generosity of the members of the UNIX Heritage Society, who donated software, documentation, anecdotes & memories, provided suggestions & insights, and gave time to lobby the powers that be to place the early UNIX systems under an open source license. Dennis Ritchie in particular has not only provided artifacts, memories and advice, but has also encouraged and mentored the restoration process: to him I owe a profound thanks. Finally, we are all indebted to Ken Thompson, Dennis Ritchie, the researchers at Bell Labs and the cast of thousands who made UNIX into such a powerful, sophisticated and pleasant system to use.

References

[1] BASHKOW, T. A Study of the UNIX Operating System, Sep 1972. http://www.bitsavers.org/pdf/bellLabs/unix/PreliminaryUnixImplementationDocument_Jun72.pdf.

[2] BURNETT, M., AND SUPNIK, R. Preserving Computing's Past: Restoration and Simulation. *Digital Technical Journal* (1996), 23–38.

[3] DEC. PDP-11/20 Processor Handbook, 1971. http://www.bitsavers.org/pdf/dec/pdp11/handbooks/PDP1120_Handbook_1972.pdf.

[4] DEC. PDP-11 Peripherals Handbook, 1972. http://www.bitsavers.org/pdf/dec/pdp11/handbooks/PDP11_PeripheralsHbk_1972.pdf.

[5] LIBES, D., AND RESSLER, S. *Life with UNIX*. Prentice Hall, 1989.

[6] LIONS, J. *A Commentary on UNIX 6th Edition with Source Code*. Peer-to-Peer Communications, 1996.

[7] MAHONEY, M. The UNIX Oral History Project, 1989. http://www.princeton.edu/~mike/expotape.htm.

[8] RITCHIE, D. M. The Evolution of the UNIX Time-Sharing System. *BSTJ 63, 8* (1984), 1577–1594.

[9] RITCHIE, D. M. The Development of the C Language. In *Proceedings of the Second History of Programming Languages Conference* (Apr 1993).

[10] SALUS, P. H. *A Quarter Century of UNIX*. Addison Wesley, 1994.

[11] TOOMEY, W. Saving UNIX from /dev/null. In *Proceedings of the AUUG Open Source Conference* (1999).

Notes

[1] UNIX is a registered trademark of The Open Group.

[2] See the excellent UNIX family tree by Éric Lévénez at http://www.levenez.com/unix/

[3] See http://www.tuhs.org

[4] Old SCO, as against the SCO Group (TSG).

[5] Early UNIX source code has very spartan commenting.

[6] See page 4 of http://bitsavers.org/pdf/bellLabs/unix/Unix_Users_Talk_Notes_Jan73.pdf

[7] 1st Edition UNIX used an 0405 a.out signature. 2nd Edition UNIX changed to an 0407 signature, indicating a slightly different format.

[8] See http://www.tuhs.org/Archive/PDP-11/Emulators/Apout/

[9] See http://minnie.tuhs.org/Programs/Ctcompare/

[10] See http://cm.bell-labs.com/cm/cs/who/dmr/primevalC.html

[11] The team was led by Tim Newsham & Warren Toomey, along with Johan Beiser, Tim Bradshaw, Brantley Coile, Christian David, Alex Garbutt, Hellwig Geisse, Cyrille Lefevre, Ralph Logan, James Markevitch, Doug Merritt and Brad Parker.