UNIX

A History and a Memoir

Brian Kernighan

specification that declares what has to be done, and write a program to interpret the specification. This approach replaces code with data, and that's almost always a win.

Yacc, Lex and Make are very much with us today, because they address important problems that programmers still face, and they solve those problems well enough that the designs and sometimes even the original implementations are still in use.

As a digression, I first met Stu in about 1967 when I was a grad student at Princeton and he was an undergrad; he was working part-time on Multics for Bell Labs during the school year. He joined 1127 after getting his PhD in astrophysics at MIT. He went to Bellcore in 1984, then to IBM, then to Google, where in a bit of good fortune for me, he was my manager several levels up when I visited there for summers.

5.3 Document preparation

Unix had good tools for document production from very early on, and this helped to make its documentation good. This section tells an extended story about the history of document preparation tools on early Unix systems. Like so much of Unix, it's a story of how the interactions among programs, programmers and users formed a virtuous cycle of innovations and improvements.

When I was an intern at MIT in 1966, I encountered Jerry Saltzer's Runoff program. (The name comes from expressions like "I'll run off a copy for you.") Runoff was a simple text formatter: its input consisted of ordinary text interspersed with lines beginning with a period that specified formatting. For example, a document might say

```
.ll 60
.ce
Document preparation
.sp 2
.ti 5
Unix had good tools for document production ...
.sp
.ti 5
When I was an intern at MIT in 1966 ...
```

This "markup" told Runoff how to format the text: set the line length to 60 characters, center the next line, space down two lines, temporarily indent 5 spaces, print the paragraph in lines of at most 60 characters, then space down one line and temporarily indent again for the next paragraph.

Runoff had a dozen or two commands like this that made it easy to format simple documents—manual pages, program descriptions, letters to friends—any text formatting that one might do today with a tool like Markdown.

Early formatters

Runoff was a revelation to me, a way to use computers that had nothing to do with mathematical computations or compiling. It became easy to refine one's writing over and over again at little cost. It may be hard for readers today to appreciate just how labor-intensive it was to prepare documents before the creation of word processing programs, when there were only mechanical typewriters—better than clay tablets or quill pens, to be sure, but any change of more than a few words in a document would require a complete retype. Thus most documents went through only one or two revisions, with handwritten changes on a manuscript that had to be laboriously retyped to make a clean copy.

When I started to write my thesis in the fall of 1968, I really wanted Runoff, since the alternative would have been to type the thesis myself on a manual typewriter (and retype it for each set of changes), or pay someone to do it for me. I'm a fast but inaccurate typist, so the former was not practical, and since I was both cheap and poor, the latter wasn't either.

Thus I wrote a simple version of Runoff that I called "Roff," for "an abbreviated form of Runoff." The problem was that there was no interactive computer system like CTSS at Princeton; there weren't any computer terminals either. All that was available was punch cards, which only supported upper case letters. I wrote Roff in Fortran (far from ideal, since Fortran was meant for scientific computation, not pushing characters around, but there were no other options) and I added a feature to convert everything to lower case while automatically capitalizing the first letter of each sentence. The resulting text, now upper and lower case, was printed on an IBM 1403 printer that could print both cases. Talk about bleeding edge! My thesis was three boxes of cards. Each box held 2,000 cards, was about 14 inches (35 cm) long and weighed 10 pounds (4.5 kg). The first 1,000 cards were the program and the other 5,000 were the thesis itself in Roff.

Readers who have never worked with cards may find this confusing. Each card contained at most 80 characters, either one line of Fortran code or one line of thesis text. If some part of the text needed to be changed, the replacement text was punched onto a few new cards that replaced the old cards, which were discarded. Fixing a spelling mistake would generally require replacing only one card, though if the new text were much longer, more cards might be needed.

$$\sum_{j=1}^{m} c[p(j),k] = \sum_{j=1}^{r} c[q(j),k]$$

This follows from the fact that for any i, the cost c[q(i),k] is allocated among that subset of the p(j)'s which are copies of q(i). That is, $\Sigma c[p(j),k] = c[q(i),k]$ for any such subset. Summation of this equality over all q(i) proves the claim.

By construction, the cost for edges leaving the i-th copy of node k in the derived tree is

$$c[k(i),k'(i)] = c(k,k') \frac{c[p(i),k]}{\sum_{j=1}^{m} c[p(j),k]}$$

But

$$\sum_{j=1}^{m} c[p(j),k] = \sum_{j=1}^{r} c[q(j),k] \leq c(k,k')$$

Therefore

$$c[k(i),k'(i)] \geq c[p(i),k]$$

and hence monotonicity of subroutine graph costs is preserved in the tree. Equality of values of edges leaving a copy of a particular node is obviously preserved since the same multiplying factor is used for all the edges leaving the given node.

Figure 5.6: A page of my thesis, formatted with Roff

I had to manually insert a few special characters like summation signs (Σ) into the printed pages but this kludgy mechanism worked surprisingly well, certainly enough for me to print my thesis, which I believe was the first computer-printed thesis at Princeton. (Figure 5.6 shows a random page.) For some years afterwards, there was a student agency that would "roff" documents for students for a modest fee. Roff was thus the first program I ever

```
any i, the cost
of the p(j)'s which
= c[q(i),k] for
ty over all q(i)

ng the i-th copy of

,k]
---
),k]

k,k')

graph costs is
of edges leaving a
eserved since the
the edges leaving
```

atted with Roff

rs like summation signs (Σ)
ı worked surprisingly well,
I believe was the first com-
ows a random page.) For
cy that would "roff" docu-
ıus the first program I ever

wrote that was used by other people in any significant way.

When I got to Bell Labs, I found that there were a couple of other roff-like programs also underway, including one by Doug McIlroy that was based on Saltzer's original. And Joe Ossanna shortly thereafter wrote a much more powerful version that he called Nroff, for "new Roff," which made it possible for the patent department to format patent applications. As I described earlier, Nroff was the critical tool that enabled the purchase of the first PDP-11 computers for Unix.

This little pocket of document preparation enthusiasts, and a community of active users of such programs, fit my interests perfectly, and I spent a significant part of the next ten years happily working on tools for text formatting.

Troff and typesetters

Roff and Nroff only handled fixed-width ("monospace") character sets, not much more than the standard alphabetic characters found on the Model 37 Teletype, so the output quality wasn't very high. In 1973, however, Joe Ossanna arranged to buy a phototypesetter, a Graphic Systems CAT, which was popular in the newspaper industry. Joe's intent was to produce better-looking internal technical documents and also to help the patent department to prepare better patent applications.

The CAT could print conventional proportionally spaced fonts in roman, italic and bold, along with a set of Greek letters and special characters for mathematics. It printed on long rolls of photographic paper that had to be developed in a couple of baths of noxious and messy chemicals. This technology predates laser printers, which did not become widely available for at least another 10 years. Furthermore, the output was black and white; inexpensive color printing did not arrive until several decades later.

Each font was a piece of 35mm film with character images, mounted on a rapidly spinning wheel. The wheel held four fonts of 102 characters each, so the total repertoire for a single job was 408 characters. The typesetter flashed an intense light through the film strip image onto photographic paper when the paper and the desired character were in the right positions. It was capable of 16 distinct sizes.

The typesetter was slow—changing sizes required it to rotate a mechanical lens turret—and the photographic chemicals were most unpleasant, but the output quality was high enough that we could produce professional-looking documents. Indeed, there were occasions when a paper submitted to a journal by a Bell Labs author was questioned: it looked so polished that it