LINUX JOURNAL

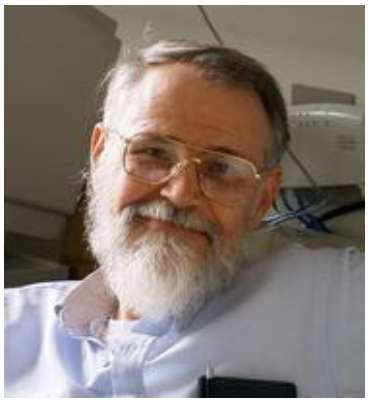
(/)

Interview with Brian Kernighan

Community (/taxonomy/term/18)

by Aleksey Dolya on July 29, 2003

Below, Aleksey Dolya interviews Brian Kernighan, one of the creators of the AWK and AMPL languages and someone who has seen the birth of UNIX and C.



Linux Journal: Could you tell us a little about yourself?

Brian Kernighan: I was born in Toronto and went to the University of Toronto as an undergraduate, in a course [of study] called Engineering Physics. It was basically a lot of science, math and engineering for kids who were good at math and thought they might be engineers but didn't

know what kind. It was a tough course, and about two thirds of the people eventually dropped out, but I managed to survive and learned a lot (a small amount of which I still remember).

I didn't really know much about computers--this was in 1960-64--and there was only one big computer at Toronto, an IBM 7090, plus a small 1620 in the electrical engineering department. I saw my first computer, an IBM 650, after my second year, and I learned a bit of Fortran when I was in my third year. I spent a summer writing Cobol for a big oil company (honest) after that. It was enough to get me hooked on programming, though I sure didn't know what I was doing and was a terrible programmer.

At Toronto, I also did a senior thesis (a literature survey, really) on artificial intelligence, which was showing all kinds of promise in 1964. So I decided to go to graduate school, without really knowing what that was all about. But it was easier than looking for a job. I wound up at Princeton because they made a better financial offer than any other school. I had a good friend, Al Aho, who was already there; he had been one year ahead of me at Toronto, in the same course.

Princeton didn't have a CS department at that time, only a group of good young people in electrical engineering, but I enjoyed it and had a good time for several years before settling down to work on a thesis. It was a very nice place to be a graduate student.

LJ: How did your life become connected with computers?

BK: I think the real turning point was the summer of 1966, where through good luck I got a job at Project MAC at MIT, [working] for Fernando Corbato. This was a fantastic experience: I was using CTSS, which was the first general purpose time sharing system and is still one of the nicest to use. It was infinitely more productive than the punch cards I was used to up to that point. I learned to program in MAD and

wrote programs to help collect information for the Multics machine; the first GE 645 arrived that summer. It was a wonderful place to live and work, with great people (like Corby, who is still alive and active). It was definitely one of the best times of my life.

The next summer, probably because of the MIT experience, I got a job at Bell Labs in the Computing Science Research Center. This time I learned assembly language properly and met a bunch of the people who I had heard of while at MIT (they also were working on Multics). Another great summer.

I went back to Bell Labs the summer after that. This time, I got lucky and worked with Shen Lin, a great mathematician and problem solver. Shen was interested in hard combinatorial optimization problems, such as the Traveling Salesman Problem. I had been working in a casual way on what came to be called the graph partitioning problem for my Princeton thesis. Shen had an idea of how to attack the general case, and I made the algorithm work in a Fortran program. It became the core of my thesis, along with some other special cases. Anyway, I had such a great time at the Labs those two summers that when I finished my thesis early in 1969, I didn't even look for another job--I just went to the Labs. I was lucky to be in the group that did UNIX and C and all of the great things that came with them; that started just after I arrived. In many ways it was the best computer science research group anywhere, part of a large and productive research organization, and it had an enormous influence on the world. I stayed there until 2000, 30 wonderful years with an amazing group of people.

LJ: What is your work these days? Do you like it?

BK: While at the Labs, I spent several semesters teaching, for example, at Princeton and at Harvard. I really liked teaching and spent the academic year of 1999-2000 as a visiting professor in the CS

department at Princeton. Princeton offered me a permanent faculty position, and after months of soul-searching on what should have been an easy decision, I decided to retire from Bell Labs and become a full-time professor. It's a very different job and role than what I was doing at Bell Labs, but I love it, too, and am having the time of my life. Princeton University is one of the best anywhere; the kids are endlessly interesting and rewarding; my colleagues are remarkable; and it's a nice community to be part of. I don't think it would have been the right thing for me to do right after getting out of school, but now it really seems perfect.

LJ: What do you teach your students?

BK: One course is called "Computers in our World". It covers how computers and communications work, for a very non-technical audience; most of the students are majoring in things like literature, politics, history and other humanities disciplines. It's a lot of fun for me, because I can talk about topics that show up in the newspaper every day that have some computer component. (One I did not use this year but could have: Dmitri Sklyarov and ElcomSoft. But I do talk about the DMCA, a US law that definitely has both technical and political components; that's the law that Sklyarov was charged with violating.)

Basically the course covers hardware (how computers work and how they are built); software (algorithms, programming, languages, systems, applications); and communications (Internet, Web, cryptography, compression and the like). There are labs as well, in which they create their own Web pages, do some simple programming and experiment with sound, graphics and spreadsheets.

The other course is called "Advanced Programming Techniques". It's for CS majors and covers a bunch of topics related to how software really is written: scripting languages, object oriented programming in C++ and

Java, user interfaces, network connections, database access, components, patterns, and the like. The students get to define and implement their own multi-person projects, so it's also a taste of software engineering on a small scale, as they worry about design, interfaces, testing, documentation, and even doing demos and giving presentations.

LJ: There are a lot of different areas in today's IT world: platforms, OSes, languages, hardware. In what areas do you consider yourself to be an expert?

BK: I used to be an expert in document preparation systems, such as troff (which ran on UNIX), and in tools for typesetting. I maintained and enhanced troff for a long time, and I wrote a variety of other text processing tools, including eqn, for typesetting mathematics. That was one major piece of my research for a long time. I was also pretty knowledgeable about such things as programming style, especially in C. I'm not an expert in anything now, though. There are too many things to know, and it gets easier to forget as one gets older.

LJ: What was your part in the birth and destiny of the C language?

BK: I had no part in the birth of C, period. It's entirely Dennis Ritchie's work. I wrote a tutorial on how to use C for people at Bell Labs, and I twisted Dennis's arm into writing a book with me. But, if he had been so motivated, he certainly could have done it without help. He's a superb writer, as one can tell from the C reference manuals, which are his prose, untouched. I've profited a great deal from being part of the book, and I treasure Dennis as a friend, but I didn't have anything to do with C.

LJ: What do you think: is C a high level language?

BK: C is perhaps the best balance of expressiveness and efficiency that has ever been seen in programming languages. At the time it was

developed, efficiency mattered a great deal: machines were slow and had small memories, so one had to get close to the efficiency of assembler. C did this for system programming tasks--writing compilers, operating systems and tools. It was so close to the machine that you could see what the code would be (and it wasn't hard to write a good compiler), but it still was safely above the instruction level and a good enough match to all machines that one didn't think about specific tricks for specific machines. Once C came along, there no longer was any reason for any normal programmer to use assembly language. It's still my favorite language; if I were marooned on a desert island with only one compiler, it would have to be for C.

LJ: You called C "the best balance of expressiveness and efficiency". What about Pascal? There are legions of Pascal programmers in the world. Is it less expressive or less efficient than C?

BK: I wrote a paper long ago called "Why Pascal Is Not My Favorite Programming Language"--that says it all. Pascal was perhaps okay as a teaching language, but in its official standard form is not appropriate for writing real programs.

LJ: What were the AWK and AMPL languages designed for? What is your part in their design?

BK: AWK was a joint effort among Al Aho, Peter Weinberger and myself; the name is our initials. I think it's fair to say we were pretty equal in our contributions. Al knew all about regular expressions and the pattern-action paradigm; Peter knew about report generation and database issues; and I had a very clear idea of wanting to be able to handle string and numeric values and conversions between them as easily as possible. I'm pretty sure that Peter did the first implementation (which only took a couple of days), aside from regular expressions, which Al did; I have maintained and modified it on my own since about

1980. We wrote the AWK book together in 1987.

AMPL is a language for specifying optimization problems such as linear programming. It acts as a sort of compiler, converting a natural and convenient mathematical notation into whatever a particular solver program needs. AMPL is joint work with Bob Fourer and David Gay. Bob is in the Industrial Engineering and Management Science department at Northwestern University; Dave was a colleague in Computer Science at Bell Labs until he retired a year or two ago. Bob had been interested in modeling languages for specifying optimization problems for a long time. He spent a sabbatical year at Bell Labs around 1984. Because I was interested in special purpose languages (like AWK), he, Dave and I worked out the initial design of AMPL and I wrote the prototype implementation. It was my first C++ program, so although it was instructive, it probably wasn't good. In any case, it did show that the language was useful to a lot of people. Dave took over the implementation, and he has completely owned that ever since. He and Bob are experts on optimization; I am not. Essentially all of the current form of AMPL is their work; I have been a member of the team only by courtesy for a long time. (We did publish a second edition of the AMPL book a couple of months ago; I worked on that with them.)

LJ: Brian, what do you think of UNIX? Is it a good and reliable platform for development?

BK: I'm used to UNIX systems that run for months or even years without crashing. If I were developing UNIX software, there isn't any other choice. If I were developing Windows software, then I would undoubtedly use Windows if there were a graphical component or if it cared about the operating system; otherwise I would use UNIX and port the program. When I do Java, I often do a mixture, because the tools I prefer [to use] run on Unix, but the graphical interfaces are more

responsive on Windows than through an X interface.

LJ: What UNIX OSes do you like? Linux? BSD?

BK: The way I use them, which is as a casual programmer, it doesn't matter--they are all the same. If I encounter some difference, it only makes me mad, because there really isn't any reason for things to be different most of the time. I use Solaris at Princeton, Irix when I visit Bell Labs, and FreeBSD on my Mac; I also have Cygwin on several PCs so that standard tools are readily available.

LJ: Is it true that you suggested the name "UNIX" for the long ago OS, Multics? What does that word mean?

BK: Yes, long ago. Multics was an acronym for something like Multiplexed Information and Computing Service, and it was big and complicated because it had many of everything. I suggested Unics for Ken's new system, because it was small and had at most one of anything. (Multi and uni are both Latin roots, so it was a very weak pun.) Someone else spelled it with the letter X; no one can remember who.

LJ: How do you find the current situation in the world of IT monopolists? How do you feel Microsoft's politics and products?

BK: Like many people, I have mixed feelings about Microsoft. They have done much good for the world, producing a common environment that has enabled a lot of creative people to build new software and hardware and sell it at reasonable prices. Microsoft's work has made computing accessible to a huge population who would otherwise not be able to use computers. At the same time, I am unhappy with some of their products. An operating system should not crash very often, if at all, and the sheer complexity of both using and programming the Windows environment is daunting.

LJ: You are a well-known expert in practical programming. Does it differ

from theoretical and research programming?

BK: As the great American philosopher Yogi Berra is reputed to have said, "In theory, there is no difference between theory and practice. In practice, there is." I'm not sure what theoretical programming might be, but code that can't be executed on a computer is unlikely to work and thus isn't terribly useful except as a thought exercise.

Research programming might mean software written as a prototype or [used] to verify that some concept can be made to work. There, the difference is that one can cut lots of corners: don't worry about errors, ignore potential hazards, provide no user interface, skip documentation and, of course, do no maintenance. In that sense, research programming is vastly easier than writing a program that will be used by many people over a long period of time. Someone (Fred Brooks, in *The Mythical Man Month*, perhaps) once said that it is at least an order of magnitude more work to do production software than a prototype. I think he's wrong by at least an order of magnitude.

LJ: How often have you written code in the last few years?

BK: Too infrequently, unfortunately, except for small experiments, examples for my courses and occasional maintenance of AWK. I did spend a fair amount of time building various user interfaces for the AMPL language--in Java, Tcl/Tk and Visual Basic--but none of these are very big, and none are very satisfactory either. Last summer I spent most of my time finishing off the second edition of our book on AMPL, which didn't involve any programming either. So I'm hoping to get back to doing more.

LJ: What are your hobbies? Reading? Sports?

BK: Mostly reading; I read a lot, mostly history and sometimes detective stories and occasionally biographies. I used to ski a bit, played squash

and racquetball, and I once got a black belt in karate. But that was very long ago indeed. Today my sports are restricted to taking long walks.

LJ: Could you say that you love computers (IT)?

BK: No. There was a time when they were incredible fun to work with, and I really enjoyed programming and getting the machine to do things, but it was never my whole life. And modern systems are so messy and complicated that they are more frustrating than rewarding most of the time. It's still pretty easy to get completely wrapped up in trying to write a program, though; that will always be fun.

LJ: You have worked in Bell Labs, alongside Bjarne Stroustrup, Ken Thompson and Dennis Ritchie. What kind of relations do you have with them? Were you like a big, wise family?

BK: We were all friends and close colleagues for many years, all in the same small group at Bell Labs. Ken, Dennis and I are all about the same age, and we all came to the Labs about the same time; Bjarne came 10 years later. I wouldn't call it family, but it was definitely good friends, and I miss seeing them all every day, which is the way it was for many years.

LJ: Could you make any predictions about IT in the future? What programming languages will we be using?

BK: There are only two real problems in computing: computers are too hard to use and too hard to program. We've made enormous progress on both of these over the past fifty years, but they are still the real problems. And I predict they still will be problems 50 years from now. Of course, we will be using machines far more powerful than today's, and our languages undoubtedly will be more expressive. But we will be undertaking far more complicated tasks, so the progress will not be completely evident.

I expect that much of the real progress will be in mechanization: getting the machine to do more of the work for us. There are many examples today--compilers, parser-generators, application-specific languages, wizards, interface builders--all of which create code for us more easily than we could do it manually. This will keep getting better: as we understand some area so well that it becomes almost mechanical to program for it, we will mechanize the process. And, of course, the level of language will continue to rise, as languages become more declarative ("do what I want", rather than "do these particular steps") and as efficiency is less of a concern for any particular aspect of a computation.

I'm less sure what will happen on the "easier to use" side, however. Here the trend for the past 10 or 15 years has been unsatisfactory. Computers are hard to use, even with ostensibly friendly GUIs and assistants and the like. This is a real problem, because computers are pervasive, and more and more all of us have to deal with them in all kinds of settings, some critical (think of flying a plane, where the "blue screen of death" takes on a whole new meaning). We simply have to make better interfaces to machines.

LJ: Brian, thank you very much. Good luck!

Aleksey Dolya is a Russian C/C++ programmer interested in network security and software protection.
email: tanat@hotmail.ru (mailto:tanat@hotmail.ru)

You May Like

(/content/fsfs-libreplanet-2021-free-software-conference-weekend-online-only)

FSF's LibrePlanet 2021 Free Software Conference Is Next Weekend, Online Only (/content/fsfs-libreplanet-2021-free-software-conference-weekend-online-only)

Logan Abbott (/users/logan-abbott)

Loadsharers: Funding the Load-Bearing Internet Person (/content/loadsharers-funding-load-bearing-internet-person)

Eric S. Raymond (/users/eric-s-raymond)

Open Source Is Good, but How Can It Do Good? (/content/open-source-good-how-can-it-do-good)

Glyn Moody (/users/glyn-moody)

When Choosing Your Commercial Linux, Choose Wisely! (/content/when-choosing-your-commercial-linux-choose-wisely)

Vince Calandra (/users/vince-calandra)

Connect With Us

 (https://youtube.com/linuxjournalonline)  (https://www.facebook.com /linuxjournal/)  (https://twitter.com/linuxjournal)

Linux Journal, representing 25+ years of publication, is the original magazine of the global Open Source community.