Search

● All of Salon.com
○ Only Technology
OK

Directory

Hot Topics
Hackers
Computer Games
E-Books
Media Borg
Napster
Andrew Leonard
Censorship
Neal Stephenson
Artificial Intelligence

Articles by date
All of Salon.com
By department

SALON.COM SITES

Arts & Entertainment
- The Movie Page
Books
Comics
Life
News
People
Politics
Sex
▸ Tech & Business
- Free Software Project
Audio

Letters
Columnists
Corrections

Salon Premium
- Downloads
Salon Plus
Salon Gear
Personals

PARTICIPATE

Table Talk
[Free, spirited

# Complete book outline

Warning to readers: This outline is subject to change at any time. In fact, the outline you are currently reading is not the original outline posted at the launch of the Free Software Project. After the first month, I became frustrated with the delays inherent in waiting until I had complete, 10 to 15,000 word-long chapters before posting them. So I have now broken down the original chapter divisions into shorter chunks. I'm hoping that this will accelerate the process of writing and uploading each section.

Ideally, this will also increase flexibility. The story of free software is ongoing, and as events warrant I would like to incorporate coverage of them into the main narrative. So if for example, a Linux company goes bankrupt, it might then be appropriate to cover that in the context of a section on the perils of going public as a free software enterprise. Or, as is the case with the current (April 20th, 2000) installment, if circumstances allow me to travel to Finland I can then write about the experience, even though Finland wasn't originally scheduled to be covered until Chapter 6, as part of the introduction to Linux and Linus Torvalds.

It is most unlikely that this will be the last change in plans. The order of the chapters may be changed, new chapters may be added, sections may be moved from one chapter to another. I'm also always interested in suggestions as to what should be covered that isn't mentioned here. I consider this outline a plan of action, but I doubt that it will map perfectly to the final product.

If you're curious you can still look at the old outline.
--Andrew Leonard

## Chapter 1: Boot time

[Free, spirited Salon forums]

War for fun and profit Can a military action boost a sagging economy?

Snoop dreams Does anti-terrorism mean the end of civil liberties?

Posts of the week

**The Well** [Pioneering members-only discussions]

Selling to suckers "Antiques Roadshow" meets Judge Judy.

**Sound Off**
E-mail Salon
Send us a Letter to the Editor
Today's letters

WAYS TO GET SALON

**Downloads**
Get Salon.com on your PDA

This introductory chapter examines some of the potential starting points for the story of free software, ranging as far back as the 11th century and as recently as the summer of 1999. The main goal of this chapter is to give readers a sense of just how broad and far-reaching the implications of free software are. The global economy, free speech and censorship, intellectual property, the rise and fall of monopoly power, the history of the Net -- these are just some of the issues that will be explored in future chapters. This chapter is meant to be comprehensible to people who aren't experts in software or computer technology; later chapters will delve deeper. (This section of the outline is unchanged from the original version).

# Chapter 2: Free speech and free software

Part I:

What do free speech and free software have in common? Much has been made by computer historians of the intersection between the counterculture of the Bay Area and the early history of the personal computer. One story that hasn't been told very often is the role the computer science department at Berkeley played in the growth of the Internet, the spread of Unix and the eventual blossoming of free software.

Not everyone at UC Berkeley was a Free Speech Movement veteran or anti-Vietnam War protester, of course. Prodigal programmer Bill Joy didn't pay too much attention to politics in the mid-'70s -- he was too busy rewriting AT&T Unix from top to bottom, and redistributing his changes as the Berkeley Software Distribution, or BSD. BSD became very popular with academics and Internet geeks all over the world, and once Joy's team added networking capabilities to it, BSD evolved into the lingua franca of the Internet.

I will argue that the contribution Berkeley made to networking, Unix and the Net can be seen as just as important as the contribution Berkeley made to radical politics in the 1970s. And indeed, for some of the Berkeley researchers, getting the Net to the people was the biggest contribution to the idea of free speech that they could make.

Part II:

Bill Joy and the other BSD hackers were working in the context of Unix. Linux, too, is in the Unix family tree. Is there something different about Unix that encourages cooperation? Or is there something different about Unix that attracts idiosyncratic independent minded cusses? I'd answer both questions in affirmative. In a technical sense, Unix is all about networking, that's why it has flourished on the Net and has helped the Net flourish. But there's also a culture to Unix that sets it and its related languages and programs apart. Unix is different from Windows in profound cultural ways, just as the MacOS is different from Windows, in quite different, but equally profound ways.

What does it mean to say that Unix has a culture. Part II of Chapter 1 will be The Free Software Project's first stab at exploring the culture inherent in various strains of code. Perl, for example, attracts a certain type of hacker -- I'd like to know why and how? How much credit can we give to Unix's original creators? And how does Unix differ from the culture of Linux?

**Chapter 3: The saint of free software: Richard Stallman goes it alone.**

More than any other single person, Richard Stallman is responsible for incubating both the idea and the reality of free software. He has been profiled countless times, including once by myself, so I'd like to approach the chapter on him from two directions.

Part I: Richard Stallman -- virus or prophet?

Even as the Internet was gaining momentum and the Berkeley researchers were pumping out new versions of Unix, back on the East Coast the "hacker ethic" was under concerted assault. Although MIT is widely credited as the birthplace of hacker culture, by the early 1980s it was beginning to look more like a graveyard. No one felt the chill more than Richard Stallman, who watched in dismay as one colleague after another left the ivory tower to work in the private sector. No longer was he allowed to share the benefits of their work; no longer was software considered the property of all. Now, everything was proprietary.

Today, the Free Software Foundation that Stallman founded in 1984 is credited as the single most important force in resisting the advance of proprietary-only software. During hacking's darkest hour, Stallman kept the light of free software freedom aflame, persevering bullheadedly against all odds in his project to write free software that everyone could use. Without the tools that he created, the development of Linux and countless other free software programs would have been extraordinarily difficult.

Stallman has an opinion on everything, and the current high profile of free software has given him a bigger pulpit from which to declaim from than ever before. Patents, the Microsoft trial, free manuals, copyright, in this section I'll give a comprehensive a look at both the person and the ideas. But I'll do it from a specific angle -- is Richard Stallman a human virus. His GPL license has been criticized as an anti-capitalist virus. What about the person himself. What kind of effect is he having?

Part II: Becoming the virus

Both parts of Chapter 3 will be written using only free software. This will require some learning on my part, and probably won't be as comfortable for me as if I used a proprietary editor. But for Stallman, comfort isn't the issue. Morality is. Some people just feel better using free software. The psychology of the users of free software is as important as the technical construction of that software.

Why do programmers love free software? Why does Linux command such ferocious fandom? The reasons go beyond just the rewards of creating or working with efficient, bug-free software and don't necessarily dovetail with political or moral motivations.

Writer Ellen Ullman calls it the "close to the machine" factor. With free software, you can get your hands dirty, you can fix problems as they arise, and you have the sense that your computer's innards are accessible to you. While this isn't always attractive to everyone -- most average computer users don't really want to muck around with a computer's insides -- it does have a potent appeal to some people who don't consider themselves programmers.

With free software, everyone can become a programmer -- or at the very least a member of a greater programming community. As one learns how to configure a free software program on one's own computer, one naturally turns to the community for help and hints, and in the process of learning, becomes a member of that community who can help others.

There is a psychological attraction to free software. There is a poetry to code, an exhilaration that comes from successful programming, no matter how trivial. Free software programming allows the purest kind of programming satisfaction -- to an almost spiritual degree. This chapter will examine the psychological motivations fueling free software excitement, and will also give the author a chance to detail his own personal journey into the world of free software. Part II will be a plunge into that psychology from a user's perspective.

# Chapter 4: The Internet and Free Software

Part I: Duct tape for the Net: A Perl beyond price

The Internet's growth from a government project that linked university research centers together into one of the most important organisms of society at the close of the 20th century has been told many times. What hasn't been previously recounted, however, is the role that free software hackers played in the Internet's evolution. Chapter IV will show how the Internet and the free software movement are linked together in a positively reinforcing feedback loop: The hackers improve the Net, and the Net enhances the quality of hacker productivity.

One entry point for telling this part of the story is Larry Wall, the creator of the programming language Perl. There would be no Yahoo without Larry Wall -- and no Amazon either, not to mention a million other Web sites. Perl is to the World Wide Web as mortar is to a building made out of bricks. It is the glue language, the thing that stitches everything together. It has made possible a whole new generation of Web-based businesses.

That alone would be enough reason to pay attention to Wall. But that was hardly his first major achievement. In the early 80s, Wall also wrote a little program called "patch." It didn't do much, and wasn't very complicated. Its main purpose was to allow programmers to upgrade their versions of much larger programs without having to get a whole brand new copy of the program delivered over the

Net -- an ordeal that could cost time and money back in the old, low-bandwidth days of the Net. Patch turned out to have huge consequences for the Internet. With patch, programmers could make their own changes to programs and distribute them quickly and efficiently across the Net. Patch made possible the style of collaborative software development that later resulted in the success stories of Linux, Apache, Perl and the rest of the free software pantheon.

Part II: The Web and Free Software

The second section of Chapter 4 will focus on how free software tools have made the new era of Web-based businesses possible. But it will also begin to take a more critical look at some aspects of free software culture. For example, there's the problem of the "benevolent dictatorship" model that most free software projects operate under. Despite the "organized anarchy" that prevails in Net-based free software projects, the most successful usually have one central charismatic leader who makes ultimate decisions. The "community" must support those decisions for them to be effective, but without a strong leader, free software projects often founder. What does this say about the scalability of free software business models -- or their long-term stability?

# Chapter 5: Guns, free software and libertarians

Part I: Eric Raymond

Chronologically, chapter 5 takes the story of free software from Stallman's lonely isolation to the mass movement that is dominating press headlines in the late '90s. Eric Raymond, the self-described "technopagan libertarian" and leading spokesman for free software, is the central focus of the chapter. Raymond, the author of several books and numerous influential articles, is also the premier anthropologist and folk historian of hacker culture. Chapter 5 will continue the exploration of the hacker psyche, with particular emphasis on the libertarian component.

Part II: The politics of software

A consideration of libertarianism is essential to the discussion of free software. Libertarian culture has always thrived on the Net, and the economic model of free software -- which emphasizes grassroots independence -- is psychologically attractive to libertarians. But the "free software movement" is often accused of being a thinly disguised left-wing attack on business. And indeed, a significant number of the software developers who work on free software projects live outside of the United States and are motivated by suspicion and distrust of American-style capitalism.

# Chapter 6: The rise of the penguin: Linux and the plot for world domination

## Part I: The Lore-masters of Finland.

The story of Linux starts in Finland, where an undergraduate at the University of Helsinki first started fooling around with ideas for his own operating system. I will be visiting Finland at the end of March, so this chapter will start there as well. Finland is enormously technologically advanced for a country its size, and Finnish programmers have played a role in the Internet's development that long predates the rise of Linux. Part II: Linus Torvalds and the creation of Linux.

How did it happen? Who is Linus Torvalds? What does he think? How is he different from the other free software hackers. The second section will trace the rise of Linux to its current incarnation as one of the most exciting stories in the computing world today. How did this happen? Linux is the biggest story in free software, the linchpin of the entire movement. In chronological terms, Linux completes the journey that began at Bell Labs.

# Chapter 7: The rise, (and fall?) of the New Linux Economy

Part I: IPO madness!

1999 was the year the stock market went crazy of Linux.

Part II: IPO despair!

2000 was the year the stock market suddenly hated Linux.

These two sections, together, will try to untangle the stormy ups and downs of the market's infatuation and then, subsequently, disgust for Linux. What can we learn about how the stock market, Silicon Valley, Wall Street and the technology industry are feeding off each other from the experience of Linux? Is it even possible that Linux can be commercialized. What are the venture capitalists thinking?

Even long-time advocates of free software were stunned by the events of 1999. Suddenly, as viewed by the investment community, venture capitalists and day-traders, Linux achieved the same buzzword status as earlier concepts like "push" and portals. Companies like Red Hat and VA Linux had huge initial public offerings, giving them stock valuations high enough to purchase other companies, hire scores of programmers, and begin to subsidize the creation of even more free software on a hitherto unprecedented scale.

Is this just another example of dot-com hype gone mad? Is Wall Street being hornswoggled? Or is something deeper happening -- is the free market recognizing the lasting value of free software? Chapter 11 will examine the intersection of the so-called New Economy with the economy of free software, and explore the problems that may arise when the desires of shareholders in publicly traded companies clash with the fiercely held community values of free software hackers.

## Chapter 8: Death to Microsoft

Part I: Microsoft vs. The Gift Economy

The central conflict in the free software saga can be summed up as a showdown between the status quo of Silicon Valley-style capitalism and the new information economy of the Net.

Free software developers operate in a kind of "gift economy." This, of course, is not how Microsoft -- or most other software or computer hardware producing corporations -- plays the game. Instead, it seeks to guard its intellectual property, or, if a competitor appears on the scene, purchase it. Patent applications, non-disclosure agreements, license agreements: the structure of techno-capitalism is built on the control of information.

The success of the free software movement offers hope that there may be new strategies possible for propelling a productive economy -- strategies that don't depend on enforcing artificial limitations on how people are allowed to cooperate or share information. It's possible that this new model may only work in the realm of software, but there may also be applications in other arenas of social endeavor -- one of the sub-themes of this book will be to look for those other arenas.

Will free software topple Microsoft? What does Bill Gates really think about Linux? How will Microsoft attempt to co-opt the movement? Gates has successfully reengineered Microsoft at least once, turning the company around on a dime to "embrace and extend" the Internet. Can he do the same with free software?

A consideration of Microsoft will lead to the wider question of what free software will mean for the entire software industry. In a growing number of cases, free software programs are qualitatively better than their proprietary commercial alternatives -- faster and smarter as well as, obviously, cheaper. Linux has a surging reputation for stability and reliability that many advocates swear puts Microsoft to shame. Apache, a Web server program that operates Web sites, is by far the most popular product of its kind. Through the release of their source code to the general public, free software programs benefit from unparalleled "peer review" -- from having thousands upon thousands of programmers hammer on the code, fix bugs and test it under every possible condition. Software today is becoming unthinkably huge and complex -- Windows NT, Microsoft's industrial-strength operating system, is reputed to consist of a whopping 35 million lines of code. In this era, the distributed resources of the entire Internet constitute the only environment large enough to adequately test all the possible mishaps that might befall so complex a computer program.

Part II: Free software, monopolies, and government action.

The pragmatic benefits offered by free software constitute the biggest threat to Microsoft. How will Bill Gates and co. adapt?

Finally, this chapter will also attempt to place free software in the context of government-business interaction. Since one of the starting points of free software was government restrictions placed on AT&T, and one of the current focal points of free software is Microsoft, it will be useful, in this politically-minded chapter, to consider how monopoly power and government interaction can intersect with and affect the world of software.

(N.B.: As of April, the last three chapters do not yet lend themselves to modularization. But that could well change!)

**Chapter 9: Free to be, you and me**

MP3. DVD. Copyrights, patents, and software piracy: No issue in the current world of software is more divisive, more hotly contested, and more unsettled than the question of ownership of code -- not just in terms of software, but in terms of entertainment and media. It's not an accident that some of the most vocal fans of free software are also busy trading MP3s or reverse engineering encryption protections for DVD players. Music and moves are software, today, and the Internet is, at the very least, the most effective distribution vehicle for such software ever invented.

Push is coming to shove here, and no one knows how the story will play out. Will the defenders of intellectual property be able to hold off the barbarians of the Net? Or will new technological realities force accommodation? Even as corporations race to patent anything and everything they can, and pour hundreds of millions of dollars into defending copyright and attacking software "piracy," the Net is forcing new business models and new ways of thinking on the entire world.

**Chapter 10: Free software bootstrapping the world**

One can argue that every dollar Microsoft spends attacking software piracy in the third world is a dollar of advertising for Linux and free software. Already, countries like India and China are taking a close look at Linux; it's cheap, it's not tied exclusively to an American corporation and it can be adapted to fit every local need. Free software is a great bootstrapping tool for countries short on resources.

One of the most intriguing points about this is that free software is in large part the product of the most privileged classes of the First World. Programmers, by and large, are paid so well that they can afford to take on hobbies like free software projects in their spare time. And in doing so, they are creating an infrastructure of tools that the whole world can and will benefit from.

**Epilogue: The greatest gift**

Even if Microsoft isn't toppled by free software, even if Linux doesn't displace Windows 2000 and even if the fundamental dynamics of the information market economy are not completely transformed by the free software movement, the possibility that we have been offered a different path to take is valuable in its own right. The Internet encourages people to work together, if not for profit, then for fun. In the new gift economy, that might be the greatest gift of all.

The Free Software Project
Read Andrew Leonard's book-in-progress on Linux and open source -- and post your comments.

OLYMPUS
Is it a Digital Voice Recorder that's an MP3 Player,
or an MP3 Player that's a Digital Voice Recorder?
Click here to decide

# Chapter 1: Boot Time

Part 1: Linus Torvalds at the Villa Montalvo

- - - - - - - - - - - -

**By Andrew Leonard**

March 05, 2000 | Nestled in the foothills of the Santa Cruz Mountains, gazing out across the western edge of Silicon Valley, the Villa Montalvo is a grandiose reminder of a different age. Built as a country home at the turn of the 19th century by three-time San Francisco Mayor James Duval Phelan, the Villa sprawls majestically across a landscape that once sprouted apricots, cherries and prunes but today is more likely to nurse Internet start-ups and computer-chip design companies. Saratoga, the small town presided over by the villa, has managed better than most to resist the relentless high-tech mall-ification of the valley, but the imaginary smell of silicon -- a smell of money, progress and greed -- still hangs in the air.

Phelan, the son of a gold rush-era liquor wholesaler who grew up to hate both Prohibition and the invasion of California by Japanese immigrants, decreed in his will that the Villa Montalvo should be dedicated to the "support and encouragement of music, art, literature and architecture." Quite the Renaissance legacy -- though one wonders if he could have dreamed that one day his home would also host glamorous press conferences trumpeting new computer gizmos.

Possibly. Phelan was a man of some imagination. He chose the name Montalvo as homage to the 16th century Spanish writer Garcia Ordonez de Montalvo, who coined the name "California" in his otherwise eminently forgettable novel "Sergas of Esplandian." And since for so much of the 20th century, California has for better or worse represented the future -- of technology, culture, entertainment and even capitalism itself -- what better place to contemplate the cutting edge of Silicon Valley than in a villa dedicated to the man who first dreamed up the state's name?

So perhaps the half-eagle, half-lion stone griffins that guard the narrow winding road up to the Villa Montalvo were not too surprised to see, one rainy morning in January 2000, a horde of journalists, analysts, chip designers, money men and high-tech industry flacks invade their peaceful territory. For this was no ordinary press conference; this was the ultimate Silicon Valley dog-and-pony show. A company named Transmeta -- notorious, on the one hand, for being the most secretive start-up in the valley, and on the other, for employing one of the world's most famous programmers, Linus Torvalds* -- was about to raise the curtain on its tomorrowland product. The next little piece of the mythological Californian future was at hand. Who would dare miss it?

Certainly not me. Like everyone else, I wanted to finally get some answers about what Transmeta was up to. But like most people there, I also wanted to catch a glimpse of Torvalds. In that new universe where the Net, the software industry and the media are colliding, Torvalds is increasingly regarded as a hero of sorts -- a knight in digital armor jousting with the grasping ogres that currently lord it over the high-tech marketplace. Never mind that in real life Torvalds is a staunch pragmatist, a person who displays zero inclination for engaging in crusades or otherwise quixotic adventures. That's immaterial: Torvalds is the primary author of Linux,* a software program that is the core of a free operating system.*

An operating system -- such as Unix,* the Mac OS and, of course, Microsoft Windows -- is the heart and soul of a computer. The phenomenal recent market successes of Linux-based operating systems, which are posing the first real threat to Microsoft's software hegemony in a decade, have thrust Torvalds into the position of being the antithesis to Microsoft chairman Bill Gates.* While Microsoft charges what the market will bear for access to its software, Torvalds gives his code away. And somehow, it works. Indeed, in a seeming paradox, vast fortunes are being generated by corporations specializing in packaging and supporting so-called "free software"* or "open-source software,"* software defined by one fundamental commandment: that the source code* to a program, variously referred to as the underlying blueprints, or recipe, for that program, be freely available to the general public.

To the uninitiated, free software sounds like a joke, a late-night psychic friends TV come-on or, at best, a fad for geeks and nerds who have nothing better to do than play with computer code all night long. But to a growing number of technology watchers, free software means much, much more: Its success points toward a possible future in which the simple act of sharing constitutes the bedrock of a new strain of capitalism. By early 2000, talk of initial public offerings, billion-dollar market capitalizations and venture-capitalist shenanigans had become increasingly common wherever free-software hackers hung out. A healthy and growing number of computing cognoscenti were even arguing that, in a truly free market, free software would inevitably dominate.

Together with thousands of other programmers scattered across the world, Torvalds is demonstrating the astonishing potential of what can be achieved when volunteers collaborate with each other via the Internet, sharing code across corporate, geographic, cultural and linguistic boundary lines. As a byproduct, Torvalds can lay claim to what is quite possibly the fastest growing cult of personality in the world of technology. For Transmeta, the public relations benefits alone are well worth his salary; just by being his employer, Transmeta ensures that some of the most keen eyes on the Net will obsess over the company's every move.

But what about the pack of photographers, the satellite trucks from CNN and ABC, the audience members calling their editors or their friends with live updates at every break in the action? Would they have come clamoring to Saratoga if not to contemplate Torvalds in the flesh? Perhaps not -- even with the lure of valet parking and a fancy lunch.

And yet there certainly was plenty of real meat to chew on at this particular chip demo. After years of hard work and an estimated $100 million or so, Transmeta had cooked up two tiny microprocessor* chips, dubbed "Crusoe," that may well usher in a new era of ubiquitous, low-cost, mobile computing. On display at the left edge of the stage was an array of gadgets that any self-respecting early adopter would have a difficult time not slavering over, including a "Web slate" designed to be an ultra-portable interface to the Net and a laptop with battery staying power three or four times the current norm.

So not only was this a gathering that demanded attendance, but it also unabashedly encouraged an exuberant display of high-tech fetishism. The case for mobile computing did not need to be made to this wannabe cyborg audience. Hardly a visitor came near who was not equipped with at least one cellphone, personal digital assistant, digital camera and/or wireless modem-equipped laptop.

I was no exception. Splayed out across my lap, as I sat in the small theater where Transmeta execs, grinning from ear to ear, declaimed upon their unique "code morphing"* software and the astonishingly low power consumption of their chips, lay my own cherished gadget, a brand-spanking-new Sony Vaio laptop computer of which I was inordinately proud.

It wasn't just the sleek, burnished design or the feather-like weight that pleased me about my laptop. My laptop made me happy because, in microcosm, it exemplified some of the changes sweeping through the software industry that were personified, on a much larger scale, by Transmeta's products and Torvalds' code. When I bought the machine, it came installed with Windows 98. But with surprisingly little trouble, I was able to transform it into a "dual-boot" system: Depending upon my whim, I could choose which operating system the computer loaded, or "booted up,"* first -- in this case, Windows 98 or Red Hat Linux 6.1.

Dual-booting is not for everyone. It's a geeky thing. Most people don't need two operating systems on their computers. But for me, it represented the possibility of choice in a dangerously monopolistic environment. The vast majority of computer users accept, without much demurral, that if they purchase a personal computer it will most likely come pre-installed with Windows (let's put aside, for the moment, the question of the Macintosh minority). Microsoft's control over that opening screen gives the company great power, as has been demonstrated in the Microsoft antitrust trial. By taking control of the boot-up sequence, I was rejecting Microsoft's claim to preeminence on my computer and reducing it to just another contender. It was up to me if I wanted the slick ease-of-use of Windows or the powerful flexibility of a Linux-based operating system -- if I wanted to be comforted by supposedly idiot-proof proprietary software that held my hand, or teased with the uncertainties of the free-software way of life.

Transmeta was also pursuing a dual-boot strategy. The company was placing its bets, or chips, as it were, on two operating systems: One chip was designed to work with Windows, the other one with Linux-based operating systems. To demonstrate this strategy graphically, Torvalds and another Transmeta employee, Dave Taylor, trotted out to battle each other in a networked bout of the exquisitely violent first-person shooter* video game Quake III. Taylor fought from a Crusoe-powered computer running Windows, while Torvalds wielded his weapons on a Crusoe-powered computer running Linux.

The showdown launched a photographic frenzy, as the assembled corps of camera-toting journalists surged toward the stage. Valley veterans must have been shaking their heads. Transmeta, a proud aspirant to the glorious chip-making heritage of the valley, was showing off chips that it believed could change the world, just as, much earlier, Intel's microprocessor chip had set the stage for the personal computer explosion. But the real excitement of the day was the sight of a jeans-wearing, sandal-clad young man attempting, without any success at all, to avoid being blown to bits by his opponent, who was sporting leather pants and tails.

Torvalds died, early and often, to the amused dismay of his fans. But it wasn't really his fault, nor, as he was quick to claim, could his pathetic showing be blamed on defects in Linux. Taylor, his opponent, was one of the original authors of Quake; heaven only knew how many hours he had logged hunting down foes in garish labyrinths that he had helped to create.

A 10-minute break followed Torvalds' unseemly demise. As I typed in some notes, a ponytailed man sitting beside me gave me some friendly grief for using a Microsoft product, Word, to write about Linus Torvalds. A fair criticism, I conceded. But I was planning to file my copy directly from the Villa Montalvo, I told him, and I wasn't going to take any chances with my still feeble Linux mastery when operating under a tight deadline in competition with every other technology reporter in Northern California.

As the break wore on, it occurred to me that this was a golden opportunity to ask for advice. My amiable critic was a reasonably well-known Linux advocate for one of the more high-profile Linux companies. I had been having a slight problem with the boot-up sequence on my laptop; maybe he could help.

The problem was a silly little thing. After powering on, the laptop presented me with a "boot prompt": If I typed in "dos," Windows 98 started up; if "linux," then Linux. But if I didn't type anything, the machine defaulted to Linux in just three seconds. I wanted it to wait longer. Theoretically, a small change to a simple configuration file should have solved the problem.

Theoretically. Life with Linux is one long learning curve, and for some reason I couldn't get it to work. But my Linux guru friend beside me smiled with the confidence of a veteran power user. "I can fix that," he assured me. "Want me to take a look?"

Sure, I said, and handed him my machine. His fingers flew across the keyboard, making changes faster than I could process. Then he handed it back to me, still running Linux. I had to reboot to start Windows so I could continue taking notes. But my computer wouldn't let me boot back into Windows. Something was wrong.

For a moment I suspected sabotage. I complained. My companion was embarrassed. His fingers flew again. And suddenly the machine would not boot, period. No Windows, no Linux, no nothing. A classic example of how a little freedom can be a dangerous thing. Microsoft Windows attempts, not always successfully, to hide its inner workings from you, the better to prevent you from amputating your own head; in Linux, self-mutilation is a snap.

I had to drive back to San Francisco to write my story. As I fought my way through a Pacific storm pummeling the coastal mountain range, I mulled over how best to start the piece. For a moment I even contemplated mentioning my boot-up misadventures.

Booting up, is, of course, a great place to start. The term is derived from the word bootstrap -- as in, to pull yourself up by your own bootstraps. The boot-up sequence is the first set of orders a computer receives upon awakening. Those orders initiate other orders -- other programs -- which in turn bring the computer to full possession of its senses.

The whole free-software movement, I realized, is itself a tremendously successful bootstrapping project. Starting with the simplest of objects -- the ones and zeroes that are the basic building blocks of code -- programmers have hacked together increasingly elaborate structures: programming languages, operating systems, the Internet itself.

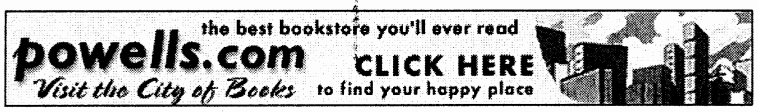
I put aside the question of how best to introduce a hurriedly written account of a press conference. I decided that if I could just find the initial boot-up moment for the whole story of free software -- a story to which I'd been devoting my reporting career -- the rest of the narrative would no doubt unfold in logically pleasing order, like a row of falling dominoes or a sequence of coded subroutines* snapping efficiently into action.

But that raised the obvious question: Just where should I look for free software's original boot-up moment?

- - - - - - - - - - - -

**Sound Off**
Send us a Letter to the Editor

GO TO Salon.com >> Technology



salon.com > Free Software Project March 6, 2000
URL:
http://www.salon.com/tech/fsp/2000/03/06/chapter_one_part_2

# Chapter one: Boot time

## Part 2: Starting points

------------

Near the close of the 11th century, an Italian jurist named Irnerius founded a school of law in the town of Bologna. We are told by Odofredus, a 13th century professor of Roman law, that Irnerius was the first "to pass on his research through his teaching." This assertion may be questionable -- no doubt there have been countless other scholars who taught what they had learned, long before Irnerius. (Aristotle and Confucius, to pick just two, spring to mind.) But the contention is intriguing. A central tenet of open-source faith is the belief that source code is an intellectual good that should be shared with as wide an audience as possible.

Free software is free speech. Bill Joy, a programmer* extraordinaire who co-founded the computer workstation manufacturer Sun Microsystems, suggests that that belief is an outgrowth of the academic tradition of sharing research results with others. And that tradition, he observes, is at least 1,000 years old, going back to the founding of what is generally considered to be the first modern European university -- Irnerius's University of Bologna.

University researchers, from computing's earliest days, have long spearheaded research and development in both computer hardware and software, so it should come as little surprise that academic customs influence how some of them view their work. But did free software really begin at Bologna, nearly a millenium before the invention of the computer?

Not, certainly, in any literal sense. And yet it is still worthwhile to think about free software in the context of nearly 1,000 years of intellectual curiosity and academic freedom. To many programmers, code is a means of expression; a form of speech; a way of seeing, understanding and interacting with the world. To put into place proprietary restraints restricting that speech is a repugnant act of censorship. Sharing source code is not just a way of creating software -- it is a way of life, a passion and a faith.

But free software is also fundamentally a software development methodology. As such, it owes much of its vigor to the efforts of programmers constantly looking for more effective ways of getting their work done. The drive for efficiency is no recent development in the software world, either. From the very beginning of the commercial computing era, programmers have realized that working together -- even across corporate lines -- makes eminently good sense.

----------------------------

In December 1952, IBM rolled out its first commercially sold electronic computer, the 701 -- also known by the quaint name "the Defense Calculator," since nearly every one of the 19 701s manufactured was rented out (for a pricey $15,000 a month) either to the United States Defense Department or to aerospace companies living off of Defense largesse.

Writing software for the 701 was a slow, painful process, made even more difficult by the lack of programming tools that today's hackers take for granted. The most pressing need was for a "compiler"* -- a program that would translate other software programs into instructions that the 701 could understand. Each company that rented a 701 needed a compiler, but writing one from scratch would be a time consuming and expensive task.

There had to be a better way. A group of West Coast aerospace companies -- pillars of the Cold War economy like Lockheed, Douglas and North American Aviation -- joined together to pool their resources. Thus was born PACT -- the Project for the Advancement of Coding Techniques. Possibly the first example of programmers who worked for directly competing companies sharing source code, PACT set an example that today's open-source start-ups are vigorously imitating.

"All parties concerned recognized that the days of 'going it alone' had to end," recalls Irwin Greenwald, a programmer who worked at the RAND Corporation think tank, which was actively involved in facilitating the collaboration. "It was too expensive both in dollars and time to completion."

Greenwald's memory is that the urge to collaborate came from technical staff at the separate companies who then sold management on the idea. If so, this is a point worth noting: Programmers know that one of the best ways to write code is to collaborate, and if that means sharing notes with your competitors, so be it. The point is to get the job done. As another participant in the project, Wesley S. Melahn, reported at the time, "The first few months of experience seem to indicate that the co-operating computer groups will be hansomely repaid for the small investment in PACT I by the savings in coding and machine time. Perhaps the greatest dividends will come from the demonstration that co-operative undertakings by groups with diversified interests can succeed and can speed up the development of the art of machine computation."

Never mind all that hifalutin stuff about academic freedom and code as "expression"; sharing source code is simply a technically superior strategy for software development. The true roots of free software can be traced back to the bald desire of the military-industrial complex to operate more efficiently.

PACT was a conscious attempt to solve programming problems that ended up providing a template for open-source business cooperation. But as we search for the wellsprings of free software, it's important to realize that software evolution is buffeted by the winds of fortune as much as it is planned by programmers. Accidents will happen.

----------------------------

In 1956, for example, the United States federal government enjoined AT&T to abide by the terms of a consent decree that forbade the government-regulated monopoly from entering non-telephony markets such as computing -- and, even more importantly, required AT&T to license its patents. So, a little less than two decades later, when Dennis Ritchie* and Ken Thompson* invented the Unix operating system, AT&T lawyers, mindful of the consent decree, initially prohibited AT&T from commercializing the software. As a result, the source code to Unix was made available to universities and research laboratories at a nominal fee.

Ken Thompson had received his Ph.D. from the University of California at Berkeley, and in 1975 he took a year's sabbatical there. Later that same year, Bill Joy arrived as a graduate student. The combination of Thompson, Joy and cheap access to the Unix source code led to an explosion of creativity every bit as radical and world-changing as the political explosion that Berkeley helped unleash in the '60s. Joy and a team of talented programmers rewrote and enhanced Unix and redistributed their changes to other Unix enthusiasts, who, in turn, often contributed their own new features and improvements -- modeling, in striking fashion, the strategy of open-source software development that is embodied today by Linux and other free-software projects.

The Berkeley programmers also added the networking capabilities to Unix that made it the ideal lingua franca for the Arpanet,* the Internet's predecessor -- a feat that held immense implications for the software industry and the evolution of the Internet. To this day, the most vigorous arena for free-software/open-source development occurs in the friendly ecological habitat of Unix. Linux is a clone of Unix, and many other free-software mainstays are most comfortable in a Unix/Linux environment. A generation of programmers have grown up with the ability, if they cared to, to upgrade Unix as they see fit. Through doing so, they created a flourishing culture indigenous to the Net.

Did the consent decree of 1956, then, kick off open source? Is it the tap root supporting free software's mighty flowering? Maybe. According to Ritchie, "the consent decree, to the extent that it prevented AT&T from thinking at all about a Unix business, certainly influenced the licensing and distribution policy, which most likely would have been more closed without the restriction." At the very least, writes Peter Salus, author of "A Quarter Century of Unix," "the decree resulted in a much more rapid dissemination of technology than would otherwise have been possible."

But before we get too excited about the potential for government intervention to aid the growth of the free-software industry, it must be recalled that the consequences of the consent decree were entirely unintended. At the time, in fact, the decree was widely viewed as a painless slap on the wrist that implied "no real injury" to AT&T. Certainly, in 1956, very few people foresaw that the eventual linking of computers together via telecommunication networks would become one of the defining technology advances of the 20th century.

The consent decree was one kind of accident, albeit on a huge scale involving the classic thrust/counterthrust of government and business interaction. But accidents on a much smaller scale have also exerted profound influences. Consider, for example, the case of the printer paper jam.

----------------------------

In 1979, the hackers who populated M.I.T's Artificial Intelligence Laboratory received a laser printer. Not just any laser printer, but a Dover laser printer from Xerox -- the very first laser printer, one of the results of the groundbreaking research conducted at the Xerox PARC laboratories in Palo Alto.

The Dover was a big clunky thing that had a tendency, as is common with printers, to get fouled up with paper jams and other mechanical problems. But not to worry. With a previous printer, the M.I.T. hackers had done what they did best: hacked the printer's software so that if it ran out of paper or jammed, a message would leap across the Lab's computer network announcing the fact. And somebody would get up and load in some new paper, or unjam the printer.

Now that they had their spiffy new laser printer, the hackers saw no reason why they shouldn't incorporate their modifications in the software of the Dover, too. But they didn't have the source code to the printer's software. As Richard Stallman,* one of the lab's preeminent hackers, recalls, Xerox would not give the Lab a copy. He couldn't see it, couldn't fix it, couldn't upgrade it.

In 1984, Stallman, a legendary and controversial figure in the programming world, founded the Free Software Foundation, an institution dedicated to promoting the notion that source code should be freely available. The incident with the Xerox printer, Stallman remembers, was one catalyst that launched him along his path.

"At the time, I hadn't reached the conclusion that non-free software is unethical -- I just observed it was a pain in the neck," recalls Stallman. "If they had offered us a copy with a copyright notice on it, and said we couldn't republish it, at the time I might have been content with the arrangement. I might even have signed a nondisclosure agreement for the source code, for all I can tell today. It was after later reflection that I concluded nondisclosure agreements were wrong on principle."

Principle. Richard Stallman is a controversial figure in the programmer community. He is stubborn and unyielding -- tact is not one of the weapons in his formidable arsenal. He's been called a communist and a crank, and some of the more business-oriented open-source entrepreneurs probably wish that the man would simply shut up and go away. In person, Stallman's long flowing hair, his steely green-eyed gaze and his tendency to take strong moral positions and hold them with unmovable tenacity make him seem more like an Old Testament prophet than a technological visionary. But his influence is undeniable and his contributions are enormous.

Richard Stallman represents the ideological core of the free-software movement. His moral fervor for sharing source code is what drives many programmers forward. More than any other person, Stallman is responsible for promulgating the notion that free software is more than just a technique -- it is an ethical imperative.

----------------------------

In what has to be one of the most ironic twists in the path to free software's origins, it can be argued that Microsoft's blistering campaign to win the so-called Web browser wars in the late '90s had the entirely unexpected result of boosting the free-software movement out of the recesses of the Net into the glare of public view. Microsoft's not inconsequential role in the free-software movement is all the more tantalizing when one considers that Bill Gates, as a young man of 19, personally played an instrumental role in turning the production of software into a proprietary profit center.

In January 1998, Netscape, the company that kick-started the Web explosion with the release of its Netscape Navigator Web browser, announced its plans to join the free-software movement by declaring that it would release the source code to Netscape Navigator. Executives at Netscape declared that their thinking had been influenced in part by the online publication of a seminal paper describing the logistics of the open source software development methodology written by hacker Eric Raymond. But there were more factors driving the decision than simply some fortuitous Web surfing. Netscape had been battered by Microsoft, and the trade press portrayed the move as both a last gasp of desperation and a sign that free software was a force to be reckoned with. The Netscape announcement marked a sea change in how the technology press covered free software: Hitherto it had been either ignored, marginalized or treated as a fading remnant of long gone days of hacker idealism; now, suddenly, reporters began paying attention to the growth of Linux-based operating systems.

Prior to Netscape's move, most software companies generally kept their source code private from all outsiders. After Netscape, the free-software gold rush began, and company after company rushed to announce their support for Linux and other open source success stories.

The Netscape announcement was more than a brilliant public relations move. Although we don't yet know the results of the experiment -- the open source version of Netscape Navigator is still not ready for public consumption -- Netscape's action illuminated the broader lines of a fundamental conflict for mastery of the software marketplace: one between proprietary control, represented by the centralized, monopolistic Microsoft, and decentralized freedom, represented by the near-anarchic open source movement. Anti-Microsoft sentiment plays a key role in energizing some authors of free software, particularly those in Europe or Asia who wish to avoid dependence on what they see as a rapacious American corporation.

The battle is far from over -- indeed, it has hardly even begun in earnest. But Microsoft itself, through its ceaseless efforts to dominate new markets, is responsible in part for setting off the hostilities, for fanning the flames of free-software passion.

---------------------------

If Netscape's 1998 announcement attracted the attention of the technology sector, then in 1999 Red Hat grabbed the whole world by the throat with its successful stock market debut. And while it might seem peculiar to consider a point in time as recent as the summer of 1999 a potential starting point for the free-software story, the watershed importance of the Red Hat public offering is undeniable. At the end of the first day of trading, Red Hat, a company with barely $10 million in revenue and no profits that specialized in assembling and distributing packages of free software, was worth $8 billion. And Wall Street suddenly started paying attention to free software.

The frenzy to buy shares in free software's premier brand name marked the moment when free software went from a programmer fad to a "new economy" phenomenon. Dot-com madness and open-source software proved to be fast friends; in the year since, a long line of Linux-related startups has gone public or announced plans to do so. One of those companies, VA Linux, even stunned itself with the highest first-day trading bounce in initial-public-offering history. Free software had become an economic tidal wave. Before Red Hat, debates about the future of free software tended to be arcane online exchanges of programmer jargon. After Red Hat, every twist and turn in the open source world would become front page news.

The Red Hat IPO is the flagship example of how a movement built largely by volunteers is becoming a significant force in the global economy. It also epitomizes the bizarre world of the techno-economy at the turn of the 21st century -- a world where buzzwords can turn into billion-dollar market capitalizations at the flip of a day-trader's switch, and free software is Big Business.

But would either Red Hat's IPO or Netscape's bold decision to release its source code ever have happened had it not been for a young Finnish programmer with hacker chops and a Net connection?

----------------------------

On Oct. 5, 1991, Linus Torvalds, then a 21-year-old undergraduate at the University of Helsinki in Finland, posted a message to "comp.os.minix" -- an online bulletin board accessible via the Internet. A few geeks noticed the post. Some even responded by sending e-mail to Torvalds. But it is safe to say that no one recognized the moment as an epochal event in the history of computing. While not quite a backwater, neither was "comp.os.minix" a gathering place for the rich and powerful. The forum was just an online hangout for devotees of an experimental computer operating system known as Minix.*

His message began:

```
Do you pine for the nice days of
minix-1.1, when men were men and wrote
```

```
their own device drivers?* Are you
without a nice project and just dying
to cut your teeth on a OS you can try
to modify for your needs? Are you
finding it frustrating when everything
works on minix? No more all-nighters
to get a nifty program working? Then
this post might be just for you :-)
```

Torvalds aimed his message at curious hackers in search of a new challenge; he called the code he was working on "a program for hackers written by a hacker." In his world, the term hacker implied respect -- definitely not to be confused with the slur mistakenly employed by the popular media to describe criminals intent on breaking into computer systems. A hacker, in the historically accurate sense of the word, is a programmer who enjoys writing code, solving problems, taking things apart to see how they work and fixing them if they are broken. Hackers are creative, unconventional and generally unwilling to pay lip service to any particular party line.

Above all, hackers are a restless breed. While they may jump at the chance to inspect or install some new agglomeration of code, they will also speedily become disenchanted if the code doesn't do what they want -- if it lacks some desirable feature or suffers from a debilitating bug.

So it was with Torvalds. He had initially enjoyed installing Minix on his home computer and testing its capabilities. But he soon became dissatisfied. He disagreed with some of the design choices made by Minix's creator, Andrew Tanenbaum,* a professor of computer science who specialized in research on operating systems. Ultimately, he thought he could do better.

As a rule, hackers are generously endowed with strong egos, and Torvalds is no exception. But even as judged by hacker standards, for a 21-year-old undergraduate to embark on the mission of writing his own operating system qualified as hubristic. Operating systems are enormously complex software programs -- the single most essential piece of software on a computer. Without an operating system, a computer is a pile of useless silicon, metal, and plastic: like a house without electricity, plumbing or heat.

However, Torvalds didn't actually need to write an entire operating system. Return, for a moment, to Torvalds' 1991 post to comp.os.minix. Take a close look at the digital infrastructure that made it possible: It positively reeked of free software. Comp.os.minix was a "newsgroup" on the online bulletin-board system "Usenet* News." Usenet News was (and is) an excellent example of the free software development model. Starting in the early '80s, programmers seeking an efficient way to share information on a vast number of different topics crafted the system together during their spare time -- without any thought of financial gain. They made their source code public, so generations of future programmers were free to add their own improvements.

Similarly, when Torvalds and the respondents to his post exchanged private e-mail, their notes were routed across the Internet by another program, Sendmail,* that was also free software -- and that, to this day, is responsible for moving a hefty majority of the world's e-mail across the Internet. Even more significantly, the tools that Torvalds used to get his code up and running -- the compilers and debuggers* that are to a programmer what lathes and radial arm saws are to a carpenter -- were themselves explicit products of the organized wing of the free software movement. Their creation had been masterminded, subsidized and, in large part, authored by members of the Free Software Foundation led by Richard Stallman -- an institution whose explict goal was to create a completely free operating system.

The story of free software, it turns out, is the story of the Net itself. Not only have programmers working in the tradition of free software been the primary architects of the guts of the Net, but as the Net has grown, it in turn increasingly facilitated the kind of large-scale collaboration -- across company lines and national boundaries, time zones and war zones -- that makes ever more complex and ambitious free software projects possible.

So Torvalds wasn't operating in a vacuum when he announced he had put some code on a publicly accessible computer. When Torvalds made his post, nearly all of the key pieces of the free software puzzle -- an infrastructure that would allow programmers to hack to their heart's content without ever having to taint their hard drives with the stain of proprietary software -- were already in place. Torvalds just finished the job by providing the last piece of the puzzle.

Torvalds focused his efforts on the creation of an operating system "kernel." A kernel,* in programmer lingo, is the core of an operating system, the all-essential code that ensures that the different parts of a computer can successfully communicate with each other. In 1991, the Free Software Foundation had yet to complete its own kernel. Torvalds' Linux kernel completed the circle.

For that achievement alone, a world of programmers is fanatically grateful. But sheer coding skill can only explain part of Torvalds' achievement. The secret to Torvalds' success lies not just in his ability to string ones and zeros together in imaginative and effective ways. More than any other single programmer before him, Torvalds exploited the Net's facility for bringing people together. Using e-mail and Usenet, he nurtured a worldwide community of freely collaborating programmers. The universe of software engineering is an environment where egos tend to run rampant and patience for fools is in short supply; Torvalds' ability to welcome newcomers into the fold soon became a Linux calling card -- and provided volatile fuel for future growth. As Torvalds noted later, "The power of Linux is as much about the community of cooperation behind it as the code itself."

----------------------------

A paper jam, a post to Usenet, a consent decree in 1956; an intellectual tradition dating back centuries combined with the common sense of programmers eager to solve problems and a moral imperative to share; the threat of Microsoft domination and the lure of dot-com profits -- tracing the roots of the free software movement back through these conflicting and competing motivations and historical accidents is like navigating a particularly twisty Borgesian labyrinth. There is no single boot-up moment.

But that's as it should be. The free software movement is anarchic and decentralized, rife with internal contradictions, competing ideologies, dissension and sometimes disarray. That may, to some observers, seem a weakness. But it is also a profound strength.

**salon.com** | March 6, 2000

salon.com > Free Software Project March 6, 2000
URL:
http://www.salon.com/tech/fsp/2000/03/06/chapter_one_part_3

# Chapter 1: Boot Time

## Part 3: The Bamboo Forest

------------

**O**ne spring day in 1997, two elderly Chinese women appeared at the front door of my house in the flatlands of Berkeley, Calif. Their English was minimal, but I speak some Chinese, and after a few false starts I grasped that the women were seeking permission to harvest my bamboo grove. Bamboo shoots, best picked just before they begin to poke their insistent heads up through the earth, are a delicacy in Chinese cuisine. My bamboo grove, which lined one fence of my back yard, separating my house from a neighboring three-story apartment building, was just beginning to sprout.

I acceded to their request, but grudgingly. I was new to the neighborhood -- it was my first spring in my new home -- and I wanted more bamboo, not less. The bamboo had been planted by a previous owner who wanted a barricade blocking his view of the apartment building; I was of the opinion that there was still plenty left that could benefit from obscurement. I told the women they could pick just a few shoots. But I felt selfish, and the quizzical look in their eyes, as if they couldn't comprehend how anyone could be stupid enough not to want their bamboo harvested, failed to improve my spirits.

The rainy season faltered two months early that spring, and the bamboo shoots that escaped the clutches of the scavengers withered and died. I felt sorry for the poor bamboo, so obviously unfit to flourish in the harsh Bay Area climate. I watered the rest of the grove throughout the summer, and idly wondered whether I should dose the bamboo with a mass treatment of fertilizer.

But the bamboo spirits had just been biding their time. The next winter, one of the wettest El Niño deluges of the entire century pummeled California. One day, when the torrent paused to grab a breath, I strolled through the yard and noted, with a sense of surprise quickly graduating to alarm, that 50 or 60 new shoots had erupted out of the moist earth, some as far as 6 feet away from the main thicket. Most of them were an inch and a half to 2 inches in diameter at the base, significantly thicker than the average already-existing full-grown stalks, or "culms." Two weeks later, I was shocked to see that a wicker chair left sitting near the grove had suddenly been hoisted several feet in the air by a fast-moving culm. After a few more days had passed I looked again: The culms were growing at a rate of several feet a week.

"It is a most impressive sight to see the new sprouts of a bamboo grove, shooting spike-like out of the ground like Cadmus' crop of dragon's teeth," wrote one bamboo-fascinated Westerner who lived in China around the turn of the 19th century. I could not agree more -- especially after I learned that bamboo grows to its full height (in this case, 30 feet) in a single season. (One species has even been clocked at 47.6 inches of growth in a single 24-hour period.) I suddenly wanted to call my Chinese visitors back. The bamboo, in the space of a few weeks, had transformed itself from a pleasing, decorative and useful adornment into an invading army.

The metaphor was more apt than I knew. Bamboo falls under a subclass of grasses that display "rhizome" reproductive habits; they are plants that propagate primarily through their root structure, rather than by seed or pollen. There are two main types of bamboo: clumping bamboo, which stays close to home, and so-called running bamboo, which botanists describe with no apparent sense of humor as "rampantly invasive."

My bamboo was running bamboo. The main patch sent rhizome roots 1 inch thick in diameter shooting out in every direction, each capable of launching new culms every few inches. A bamboo patch has no central tap root to decapitate, and any shred of rhizome left undemolished can relaunch the entire patch. No wonder Li Khan, the 13th century author of one of China's greatest treatises on bamboo, tells us that the proper word to describe the extending rhizomes of the running bamboo is xingbian -- "on the march."

My running bamboo was advancing on the foundation of my home. The prospect of rhizomes ripping my basement apart did not thrill me. I was even less delighted to overhear the neighbors discussing my bamboo. The rhizomes had penetrated the fence between us, scooted under 4 feet of concrete and started sending troops of sprouts up through cracks in their pavement.

Shortly after discovering how quickly the bamboo was spreading, I retrieved a spade from my basement and began to dig at the base of one of the culms. But before I reached the rhizome I stumbled into a spaghettilike intertwining network of much smaller roots, or rootlets, that originated in the culm and also radiated out from the rhizomes. Together, the rhizomes and rootlets were replacing the uppermost foot of topsoil in that part of the garden with a woody, impenetrable mass easily capable of denting my spade.

I paused in reflection. The wind ruffled the bamboo leaves -- a gentle rustle that in Chinese culture has long been considered an indicator of elegance and gracious living but to me seemed a most sinister susurrus. Still, even as my fear began to mount, I found it difficult not to admire the ornery, gnarly survival of the bamboo. Praised throughout millenniums in Asia for combining strength with flexibility, bamboo is clearly one of nature's great achievements. If I had known then what I now know -- that a bamboo grove was one of the only living things to survive the atomic blast on Hiroshima, or that a bamboo forest is considered one of the safest places to be in an earthquake (because the interlocking rhizomes hold the ground together) -- I would no doubt have quailed at the prospect of ever overcoming the graceful intruder. But at the time, I was just impressed with how tough the plant was.

I considered a backhoe. I contemplated poison. I wondered about dynamite. But the first option struck me as unmanly and the second as ecologically unwise. The third would have delighted my children but probably not helped my property's value. So at the advice of a friend, who, not coincidentally, is the editor of this book, I obtained a mattock: a heavy-duty peasant implement of destruction, a combination of ax and pick, designed for breaking up inhospitable terrain. The mattock is an altogether pleasing tool, and it sliced right through the bamboo roots -- not quite like a knife through butter, but still with undeniable confidence.

I enjoyed swinging the mattock. I am a technology reporter for an online magazine, which means most of my waking life is spent sitting in front of my computer writing words meant to be read by other people sitting in front of their computers. The virtual life is deficient in visceral fulfillment; after pushing e-mail back and forth all week, the prospect of repeatedly hurling a heavy chunk of iron and fiberglass into the dirt offered welcome satisfaction. As I raised the mattock over my back and let it fall with a sweet thwack into the bamboo, I fell into a nearly unthinking rhythm -- hoist the mattock, let it fall, hoist, fall, hoist, fall. Every so often I would grab a 4-foot crowbar and crack another section of rhizome out of the ground, lifting the grotesquely beautiful twisted mass of roots on high to flaunt at my family, as if I had just snared a 20-pound bass or brought down a charging 12-point stag with a bow and arrow.

But try as I might, I found myself unable to sever my workaday life from my backyard labor. In between hoists, my thoughts swung back to the cornerstones of my daily reporting. For two years, my attention had increasingly gravitated toward one particular set of stories: the free-software movement. The story of free software, it seemed to me, set forth a grand narrative about technology that put the entire world of computing into sharp, intriguing focus.

As far as I was concerned, thumping away at my runaway running bamboo, the story of the free-software movement was equal parts political revolution, cultural upheaval and economic tidal wave. It was the most interesting and most important narrative to be told in the computing universe. My editors at Salon.com agreed with me, their enthusiasm fueled by the circulation figures our traffic tabulating software registered whenever I wrote an article on the topic. For nearly two years they had been encouraging me to follow the story, to attempt to answer the many questions that rippled off free software's wake. How was it possible that free-software projects could battle Microsoft and Netscape for market share? How could a ragtag band of hackers dotted across the world -- from Finland to Fremont, Calif. -- be collaborating with such efficiency? What did the concept of free software mean for the protection of intellectual property? And what would happen when the big guns of corporate capitalism finally trained their sights on this upstart? Could free software actually win in the long run? Or would Microsoft annihilate it, as it had demolished so many opponents before?

My editors weren't the only people paying attention. As I labored away in my garden and at my computer, mighty Microsoft was casting its Sauron-like eye upon these meddlesome hacker hobbits. Microsoft is an arrogant company, but it is not stupid. Several influential Microsoft executives were justly alarmed at the fast growth of a competitor that might possibly, in the long run, be even more dangerous to Microsoft's stock price and quarterly profits than the trustbusting Department of Justice, or any gaggle of Bill Gates-hating Silicon Valley CEOs.

The free-software movement poses a unique challenge to Microsoft. Microsoft's traditional strategy, when faced by a threat from another company, is simple: Buy out, crush or subvert the enemy. Yet even though there are a smattering of corporations boasting particularly high profiles in the free-software world, there is no single company that symbolizes or controls the movement. The code itself is common property, the product of a collaborative effort midwifed by the Internet. Microsoft would be able to squash free software about as easily at it could squelch the Net itself.

One particular morning I swung my mattock at yet another square inch of gleaming rhizome. Thwack. I stared with despair at how little progress I had made after several hours of backbreaking work. Theoretically, this was fun, a relaxing change of pace. But I was really getting next to nowhere, and I had to face the fact that if I didn't uproot every last square inch, some tiny rhizome splinter would start it all up again. That roiling mass of roots -- the woody rhizome, the slender rootlets -- all twisted together in incredible complexity ... how dared I imagine I could defeat it? Who did I think I was -- Microsoft?

Rhizome power. How obvious could a metaphor be? My bamboo offered me a clear and simple demonstration of just what kind of foe Microsoft and the rest of the proprietary-software world faced from the free-software challenge.

The Internet is a rhizome: It has no central trunk, no main axis, no single point of entry or exit. It spreads everywhere, connects everything. The Internet even does Mother Nature one better: It's a superrhizome. Bamboo roots interlock and intertwine, but they don't actually interconnect; if you sever one rhizome, you create two distinct patches. But the Internet is built on the principle of multiply redundant interconnection. There's always another way through, another "workaround," as programmers like to say.

Recall Torvalds' "community of cooperation." That community is a social reflection of the power unleashed by the Net -- the fuel for unquenchable grass-roots excitement. It makes sense, even if the Oxford English Dictionary tells us that the original meaning of "grass roots" refers only to the lowest, or most fundamental, level of a thing. Rhizomes, after all, are grass roots; they are the interconnecting substrate that stitches an endless prairie into one living organism. They emblematize the lack of bureaucratic hierarchy that makes a successful grass-roots campaign an unstoppable phenomenon.

I am far from the first person to have latched onto the grass-roots potential of the Internet's rhizomelike characteristics. Activists of every political and ideological persuasion are wont to seize upon the Net, imagining it the perfect organizational tool for energizing do-it-yourself campaigns. Whether the cause is Tibetan independence, the right to keep and bear arms, pro-choice or pro-life fanaticism makes no difference. Only the willfully blind fail to recognize how fast e-mail and the Net can transmit information and rally the faithful.

Even if the Net's early pioneers didn't label it with botanic precision, they knew what they were seeing. As John Gilmore,* a well-known programmer, "cypherpunk"* and free-software advocate has often been quoted as saying: "The Internet interprets censorship as damage and routes around it." The description sums up more than just the resilience of modern telecommunication networks -- it's a dandy summary of how a rhizome survives in a hostile environment.

Libertarians like Gilmore have long viewed the Net's decentralization as a welcome refuge from Big Government control. As they zip their most private, confidential information from hidden node to hidden node in cryptographically encoded sheaths, they imagine that they may elude the heavy hand of the tax man or the censor. Their version of rhizome liberation is escapist in a most practical sense. But they aren't the only people placing their bets for transcendent change on the Internet. Philosophers, idealists and visionaries of all stripes employ the Net as a magic mirror to reflect the object of their most ardent desire.

The libertarians crave escape, while other revolutionaries cry for rebellion. For some philosophers the metaphor of the rhizome seduces with a promise to discombobulate all normal power relations and dynamics, to enable "resistance" to the status quo, however that status quo is defined. For their online-savvy disciples, the Internet is an attractive embodiment of such theory. Unmappable, inchoate, ever changing, ever growing, contemptuous of geographical borders or legal restrictions -- what better home could there be than the Net for a myriad of "temporary autonomous zones," "pirate oases" welcoming dissidents of every description?

There's just one nagging problem. A close review of the Net's impact on society doesn't automatically prove that the Net is actually accomplishing significant political or social change, despite its obvious potential for enabling grass-roots campaigns. One reason, of course, is that the tool has no inherent bias -- anyone can use it. If, for example, both Republicans and Democrats take to the Net in a U.S. Senate election, the net advantage to either side, so to speak, is negligible, a wash. Virtual organizing doesn't always alleviate flesh-and-blood oppression. What difference has e-mail yet made to religious freedom in China?

But the Net is making a difference in the arena of software development. And, unlikely as it may seem, there is a connection between the arcane intricacies of how best to write complex, powerful code and the question of whether the Net will or can change society for the vast nonprogramming majority.

Free software is the leading edge of what is to come, the first product of the indigenous culture of cyberspace. That shouldn't be a surprise. Programmers built the Net and were its first inhabitants; naturally, they would be the first to understand how to most fully exploit its potential. What is really eye-opening, however, is how different the culture that those programmers created online is from the culture that dominates the offline world. One can even argue that this new programmer culture -- the culture of free software, the so-called gift economy -- has grown up in resistance to the standard operating procedures of the technoeconomy.

The "gift economy" is a phrase originally coined by the French anthropologist Marcel Mauss to describe certain practices of exchange he observed in tribal peoples of America's Northwest and the Southeast Pacific. In a gift-economy society, people volunteer their services or goods to others and in turn benefit from similar volunteer efforts. For example, the Chinook Indians regularly held "potlatch" gatherings in which all participants contributed their own offerings -- the ancestor to today's potlucks. By giving to others, you expressed your own status and also incurred in the recipients of your gifts a reciprocal obligation. Many years later, digital anthropologists seeking to explain how the Internet grew in its early years began to use the "gift economy" phrase to describe how programmers contributed their own software tools to the Net community without expecting direct recompense, but nonetheless, by their own example, encouraging others to also give freely.

Microsoft does not operate according to the gift economy, nor do most icons of modern capitalism. The gift economy is an oddity, a culture in which ways of living that, to put it bluntly, simply feel good pay off. The gift economy, based on sharing, collaboration and openness, is only now translating into an economic windfall, and none of it could have happened without the spread of internationally linked computer networks. One lesson of free software is that highly complex projects can be undertaken on an essentially volunteer basis, if there is an infrastructure available that seamlessly enables tapping the resources and abilities of a large enough group.

The Linux explosion is the gift economy's greatest single success, save for, of course, the Internet itself. Linux has proved that certain truths heretofore held self-evident about how to become commercially successful don't necessarily hold water. To succeed, you don't need brute force, a $100 million-dollar marketing campaign or a ruthless determination to own or crush any competitor. Even more fundamentally, you don't necessarily need to spend money to make money.

We live in a world where software increasingly underlies every aspect of human existence. Good software is a necessary tool for survival. Linux, and a vast toolbox of other useful items provided to the world by the free-software movement, can be used to build efficient organizations, run companies, allocate resources and, wherever there is a bootstrapping need, fill it. Whether they know it or not, the free-software programmers are helping to change the way the world does business, by empowering the little and afflicting the big.

I had contemplated, in general terms, the free-ranging nature of the Internet's infrastructure many times before, but until that moment when I stood, leaning on my mattock, staring into the heart of the rhizome, I hadn't made the connection between the Internet's fundamental characteristics and the vigor of free software. Free software programs are the shoots springing up from the Internet's multitudinous nodes. Wherever there is a crack in the software industry's pavement, they'll squeeze through and grow like mad. Chop one off, and a hundred more will spring up -- just like bamboo shoots after a spring rain. And like bamboo, the Net's decentralized structure makes it resistant to nuclear devastation and to the digital equivalents of poison or rampaging backhoes.

Chinese poets have long compared the snapping cracks of fast-growing new bamboo shoots to the sound of thunder and lightning. In the 11th century, Ou-yang Hsiu wrote:

As startling thunder cracks a maddening whip,
So their misty sheaths unfold from patterned stem.
Humble of heart, they yet tower high themselves.

As I swung my mattock with renewed energy, excited by the insight proffered me by the bamboo, that startling thunder cracked its maddening whip on my own reporter's soul. How high would free software tower?
**salon.com** | March 6, 2000

salon.com > Free Software Project May 16, 2000
URL:
http://www.salon.com/tech/fsp/2000/05/16/chapter_2_part_one

# BSD Unix: Power to the people, from the code

How Berkeley hackers built the Net's most fabled free operating system on the ashes of the '60s -- and then lost the lead to Linux.

------------
BY ANDREW LEONARD

**B**y the time Bill Joy arrived in Berkeley, Calif., in 1975 to attend graduate school, the fabled capital of leftist radicalism was a bit ragged around the edges. If the 21-year-old programming wunderkind had glanced at the headlines blasting out of the local alternative weeklies, he might have wondered just what kind of insane mess he had gotten himself into. In San Francisco, Patty Hearst was on trial for a bank robbery committed while the newspaper heiress was toting machine guns for the Symbionese Liberation Army. In Oakland, the Weather Underground botched a bombing of a Defense Department building. Even the reliable bugaboo of CIA recruitment on the University of California's Berkeley campus failed to generate more than a token protest.

Berkeley was burned out, its radical energy wasting away in infantile terrorism, conspiracy theorizing and drug overdoses. The Free Speech Movement that had galvanized the university in the '60s belonged to another geological age. Ken Thompson, co-creator of the Unix operating system, graduated from Berkeley in 1966 with a degree in electrical engineering. He returned to the university from Bell Labs for a sabbatical in 1975. But the campus on which he had once walked to class through clouds of tear gas had changed. That year, says Thompson, Berkeley "had turned into the most politically apathetic place I'd seen."

But it was the right place for Joy. "He never looked at those [alternative] papers," says John Gage, a close friend of Joy's during the Berkeley years and later at Sun Microsystems, a company co-founded by Joy. Today, Joy calls himself a "staunch Democrat" and has recently carved out a new niche as a techno-skeptical doomsayer, but in the '70s he was, by his own description, "not an activist." Joy chose to attend UC-Berkeley instead of

Stanford or MIT not because he was attracted by its politics or countercultural reputation but because the computer science department's hardware was so obsolete that he figured he'd have no choice but to confine his research efforts to studying computing theory -- which was exactly what he wanted to do.

But theory turned out not to be Joy's forte. He started hacking code and never stopped. "His goal was to build something that *worked,"* recalls Gage. And so he did. During his seven years at Berkeley, Joy and a few other graduate students and staff researchers spearheaded an intensive software development effort that culminated, most famously, in a radically improved version of AT&T's Unix, known simply as Berkeley Unix or, more commonly, as BSD,* for Berkeley Software Distribution.

Talk about your killer apps! Berkeley Unix worked so well that DARPA* chose it to be the preferred "universal computing environment" linking together Arpanet* research nodes, thus setting in place an essential piece of infrastructure for the later growth of the Internet. An entire generation of computer scientists cut their teeth on Berkeley Unix. Without it, the Net might well have evolved into a shape similar to what it is today, but with it, the Net exploded.

How did the small group of Berkeley programmers pull off such a feat? Well, for one thing, there was Joy, a programmer around whom legends accrue like so many iron filings stuck to a magnet. He could read at age 3, play chess at 4 and, during his oral exams, invented a "sorting algorithm"* on the fly that so stunned his examiners, one of them later compared the experience to "Jesus confounding his elders."

But too much focus on Joy, a favorite target for business magazine hagiography, obscures the larger picture. Berkeley's most important contribution was not software; it was the way Berkeley created software. At Berkeley, a small core group -- never more than four people at any one time -- coordinated the contributions of an ever-growing network of far-flung, mostly volunteer programmers into progressive releases of steadily improving software. In so doing, they codified a template for what is now referred to as the "open-source software development methodology." Put more simply, the Berkeley hackers set up a system for creating free software.

BSD itself wasn't originally free software. Joy sold it, with the University of California's blessing, at a nominal cost only to people or institutions that had already purchased licenses permitting them access to the source code of AT&T Unix (although, in practice, Joy's efforts to verify whether would-be buyers really did own licenses may not have been overly vigorous). But in spirit, Berkeley Unix was indeed free: As Dennis Ritchie, Thompson's collaborator in creating Unix, observes, anyone who wanted to hack on Unix usually had access to the source code, one way or another. And if those hackers sent their modifications to Berkeley, and they were deemed good enough, they became part of a code base maintained by programmers who wanted nothing more than for their software to be widely used, for as low a cost as possible.

Berkeley Unix has morphed through multiple phase shifts since its inception some 20 years ago, from the Joy-dominated era of the late '70s and early '80s to the more collaborative period that began after Joy's departure to Sun in 1982. But in the early '90s, after a bitter confrontation with AT&T, BSD finally did become "freely redistributable," and descendants of BSD -- led by FreeBSD,* but also including OpenBSD* and NetBSD* -- are vigorous participants in the contemporary battle for operating-system supremacy. Yahoo, arguably the world's busiest Web site, runs on FreeBSD. And yet, despite its proud heritage, BSD's current status doesn't quite match up to its early fame. A victim of schisms within its own developer community, bruised by the battle with AT&T and wounded by the defection of Joy to Sun, BSD is currently a small player, especially as compared with Linux. Linux-based operating systems have seized the public imagination.

BSD patriots argue that the battle is far from over, that BSD is technically superior and will therefore win in the end. That's for the future to determine. What's indisputable is BSD's contribution in the past. Even if, by 1975, Berkeley's Free Speech Movement was a relic belonging to a fast-fading generation, on the fourth floor of Evans Hall, where Joy shared an office, the free-software movement was just beginning.

The connection between the two movements is clear, if not direct. By demonstrating the power of cooperative software development, and by strengthening the software backbone of the Internet so it could further nurture such development, BSD helped enable the creation of a medium that will do more to spread free speech than anything hitherto constructed. Power to the people, from the code.

Many nice, upper-middle-class Berkeley backyards boast a redwood patio, possibly a hot tub, perhaps a vegetable garden complete with thriving rosemary bushes and marauding raccoons. Bob Fabry's backyard has a radio tower that wouldn't look out of place at a major Air Force base. Capable of telescoping upward to a height of 100 feet, and built mostly by Fabry himself -- the hub that rotates the antenna was scavenged from a 1940s airplane propeller -- the radio tower looms beside Fabry's home like a not-so-miniature Eiffel Tower. One look at it and you realize you are in the presence of a very dedicated geek.

Fabry takes his ham radio "hobby" seriously. He once even helped organize a trip to the uninhabited wilderness of Heard Island, about 1,000 miles north of Antarctica, just to set up a ham radio station for a few days so amateur-radio enthusiasts all over the world could enjoy the pleasure of exchanging radio signals with the faraway station.

But Fabry's most impressive achievement scales far beyond his tower or his expeditions. He is the Berkeley computer science professor who orchestrated the creation of Berkeley Unix. Not that he wrote a lot of code -- that honor belonged primarily to Joy and the other members of Fabry's Computer Science Research Group, an all-star band of programmers whose roster included names like Sam Leffler, Kirk McKusick, Mike Karels and Keith Bostic. But while Joy and others were hacking for 36 hours at a stretch, improving file systems,* networking performance, memory utilization and a hundred other arcane but crucial elements of Unix, Fabry was running interference -- maneuvering through the formidable bureaucracies of the University of California and AT&T, dealing with departmental politics and backbiting and, most important, writing the grant proposals that brought a steady flood of DARPA money into Berkeley.

Fabry was personally responsible for bringing Unix to Berkeley. His reasons were simple, and offer an early example of the pragmatist bent that has characterized BSD development ever since.

Unix was cheap. AT&T had been forced to practically give it away for free by government order. But Unix was also, fundamentally, a hack designed to work on cheap hardware. Back in 1969, Thompson wanted to get a computer game called Space Travel working on a castoff PDP-7. So what did he do? He wrote an entire operating system that made it possible. Kind of like using a nuclear missile to hammer a nail -- but then that's often standard operating procedure for obsessed hackers.

Fabry was entranced by Unix's affordability, along with the ease with which it could be adapted to different computer hardware. In addition to his academic research focus on operating systems, he was involved in setting up computing resources for the UC-Berkeley student body. In the mid-'70s, this could be an expensive proposition. Typically, operating systems that would allow multiple users of a mainframe* to work at individual terminals were designed only for extremely expensive computers. The costs per user could easily reach $50,000 a *terminal,* which made such systems impractical for pedagogic purposes. But Unix, used in combination with a relatively inexpensive PDP-11 from DEC, ended up costing closer to $5,000 "per seat."

Even better, a $99 license fee bought you access to the Unix source code -- to the blueprints, the magic recipe, the key that unlocked all hidden mysteries. For researchers, teachers and students, this was priceless. Researchers working on cutting-edge operating system technology could experiment with already existing source code and modify it for their needs; students who wanted to learn how an operating system really worked could find out by getting their hands dirty with the code. Duane Adams, the DARPA contract "monitor" who administered the Berkeley Unix contracts, notes that the availability of source code was an explicit reason why DARPA chose Berkeley Unix instead of contending aspirants like DEC's* VMS.* Never mind that VMS had been designed from the bottom up for the DEC VAX computers that were the most popular hardware for Arpanet nodes; VMS was a closed, proprietary system. You couldn't get in and muck about, so it just wasn't attractive to researchers.

DARPA was also recognizing reality. Prominent researchers, hungering for the magnetic tapes carrying Berkeley's latest distributions like so many desperate junkies, demanded that DARPA adopt Berkeley Unix because that's what they were already using.

"What was driving DARPA," says Fabry, "was that almost all of their contractors were telling them that they were running Berkeley Unix and it was superior to anything else available."

As one popular explanation has it, Unix's source code became widely available through a lucky accident -- as an unanticipated consequence of the consent decree that forbade AT&T from commercializing its non-telephony-related inventions. But that's only a part of the story. Unix was always more than just a bit player in a showdown between the world's largest government and the world's biggest corporation. Unix was, at least in the mind's eye of scientists like Fabry, "a thing of beauty." And from the very beginning, Unix benefited from a communal vibe that spread directly from its creators, Ritchie and Thompson.

Fabry recalls grasping the hidden wonders of Unix one week in 1975 when Thompson conducted a "reading" of Unix over several successive nights.

"The first meeting of the West Coast Unix User's Group had about 12 or 15 people," recalls Fabry, a mild man, now 60 years old, who clearly delights in his 25-year-old memories. "We all sat around in Cory Hall and Ken Thompson read code with us. We went through the kernel* line by line in a series of evening meetings; he just explained what everything did ... It was wonderful."

The reading of the code: Thompson's primeval act of deconstruction was an initiation into the Unix cabala, a ritual passing down of code lore. Fabry may have brought the first physical manifestation of Unix to Berkeley, but Thompson's reading embedded it in Berkeley's soul. Eric Allman, *" who was later to write sendmail,* the open-source-software mail transport program that still shuttles the vast majority of Internet mail across the Net, was an undergraduate at Berkeley when he attended the readings. He still has his marked-up "listings," reams of cheap, flimsy computer paper with notes scribbled on them, detailing the obscurities of the C programming language and other Unix arcana.

"The really bizarre thing is that Ken Thompson did a free tutorial on Unix kernel internals," recalls Allman, "and everyone fit into a rather tiny room." Today, you'd need to rent a ballroom.

Fabry marched against the Vietnam War while he was a graduate student in Chicago, and notes proudly that in his entire 12-year tenure at Berkeley he never once wore a tie. But although some historians have later described the Berkeley hackers as freedom fighters -- especially in the context of their battle with AT&T (which came well after Fabry had retired) -- neither Fabry nor the hackers themselves saw what they were doing in such explicit ideological terms. But when I ask Fabry if there was ever a moment when the goal crystallized in his head that software *should* be free, he turns the question around:

"Where did it come from that code should cost money? I think that's the fair question," says Fabry. In the mid-'70s, most programmers had grown up in an era where software was usually included with hardware and not considered a separate revenue source or proprietary intellectual property. Joy saw his work on Unix as research, to be shared with the rest of the academic research community the same way professors had been sharing the fruits of their labors for thousands of years. Ritchie and Thompson wanted their software to be used -- they did everything in their power to help the Berkeley programmers fix bugs and make improvements.

None of them saw himself as a crusader. But a trickle of idealism still occasionally leaked out.

"I think the spirit in which we were putting this all together was much the spirit that was picked up later by the Free Software Foundation and the various people who were trying to build 'software for the people,'" says Fabry. "The idea is that there is no duplication cost for software, so it ought to be basically free, and we were all working together to try to produce this ideal system that we would all love to have, and love to be able to use ourselves. That was the goal of a lot of people, and of course that was the original goal of Ken Thompson and Dennis Ritchie in starting Unix."

Fabry retired at age 43, tired of the DARPA treadmill and eager to focus his energy on his ham-radio tinkering. But Berkeley Unix's record of success still thrills him.

"Berkeley Unix was clearly the most successful university software project that has ever gone on," says Fabry in a rare moment of assertiveness, before backtracking slightly. "I don't know, I haven't been keeping up since 1983 and maybe there's been something since then, but I believe that that was true at the time. We had literally thousands and thousands of installations, and a whole generation of computer science students all around the world grew up on Berkeley Unix. It set a standard for operating systems that people are still having trouble doing better than. It was also the first efficient networking solution; for years it was the only game in town, the basis of Internet development. It really was one of the things that the people who made the Internet what it is today built on. There were battles that had been solved that didn't have to be solved again in order to do whatever new part that they wanted to do."

"Bill codes like a demon." -- John Gage

"His code was ugly." -- Kirk McKusick

"Bill Joy was a fabulous marketer." -- Eric Allman

"Bill was superb. He was the epitome of what one would like to see in a graduate student. -- Bob Fabry

"He had an infectious enthusiasm about him, where he would just get the people around him to do stuff. And he had an incredible drive to get his software out so that other people would use it." -- Kirk McKusick

"Berkeley Unix was the work of many talented developers. Bill Joy's particular genius was in integrating the work of these many contributors from many different organizations." -- Rob Gurwitz

"Bill's really smart." -- Sam Leffler

--------------------

Gage's eyes twinkle as he recalls one of his favorite Bill Joy stories. The scene, he says, is in a boardroom high up in a building overlooking Washington, D.C. The time is the early 1980s. In attendance are some representatives from DARPA, some employees of BBN* (a Boston company that received the original DARPA contract to build the Arpanet) and a few Berkeley hackers, including Joy.

At issue was an annoying problem that had been bothering DARPA. DARPA had given Berkeley a major contract to enhance Unix so that it would be suitable for DARPA's network of research sites. It had also decided that Berkeley Unix should incorporate TCP/IP,* a specification for how Arpanet machines would interconnect. Devised by Vinton Cerf and Bob Kahn, TCP/IP (Transmission Control Protocol/Internet Protocol) was -- and still is today -- the basic method by which computers talk to each other across the Internet.

But DARPA had given BBN the contract to implement the TCP/IP protocol,* to write the all-important TCP/IP "stack."* Joy had been instructed to plug BBN's stack into Berkeley Unix. But Joy refused to do so. In his opinion, BBN's TCP/IP wasn't good enough. So he wrote his own high-performance TCP/IP stack.

As Gage tells it, "BBN had a big contract to implement TCP/IP, but their stuff didn't work, and Joy's grad student stuff worked. So they had this big meeting and this grad student in a T-shirt shows up, and they said, 'How did you do this?' And Bill said, 'It's very simple -- you read the protocol and write the code.'"

"That really frosted the BBN guys."

Well, sure. In programming lingo, a flat statement like "Read the protocol and write the code" is, to borrow some modern slang, a major dis. But did Joy really say the words? And did BBN's code really not work?

No and no, says Rob Gurwitz, the BBN programmer who wrote BBN's implementation of TCP/IP. Gurwitz says he was at all the DARPA steering committee meetings that handled TCP/IP matters during that era, and he doesn't remember ever hearing Joy make such a statement. Gurwitz, who says he worked closely with Joy and Sam Leffler on the integration of TCP/IP into Berkeley Unix, also says Joy's version of TCP/IP was not a direct replacement for BBN's code. Joy's stack was designed to maximize performance over local area networks that had wide bandwidth connectivity -- like an Ethernet* network designed to serve an entire university campus, for example. Gurwitz's version was built to operate on the much narrower 56Kbps telecommunication lines that made up the Arpanet's backbone.

Nonetheless, the Berkeley hackers were (and are) convinced that their implementation was superior, and they continued to resist all attempts by DARPA to force them to include the BBN version in Berkeley Unix, even though, according to Gurwitz, several DARPA sites continued to use BBN's version for years. As is the case with most programming disputes, the deeper one delves into the TCP/IP spat, the more "Rashomon"-like the search for truth becomes.

So why is the squabble important? For at least three reasons.

First, the incorporation of TCP/IP into Berkeley Unix can be, and often is, singled out as the most important innovation that made the Internet function efficiently. Joy and other Computer Science Research Group* veterans argue that their version of TCP/IP was crucial, because only it was technically good enough to satisfy researchers who wanted to communicate with each other and get work done on their local networks as well as on the Internet.

The TCP/IP stack, one could argue, was the original Promethean gift of fire to the mortals of the Net. And when the Internet suddenly boomed in the '90s, Berkeley Unix scaled up right along with it -- a testament, says Kirk McKusick, to the quality of its design. To this day, BSD advocates contend that the networking performance of BSD, which can still be traced all the way back to Joy's TCP/IP code, outclasses the best that Linux-based operating systems can do.

Second, the TCP/IP stack played a deciding role in settling the legal battle between AT&T and the University of California. The breakup of the AT&T monopoly in 1984 finally permitted AT&T to commercialize Unix. For years, as Unix became the preferred language of the Net and academia, AT&T had steadily increased licensing fees from the original $99 all the way up to $250,000. But AT&T wasn't the only interested party. In the early '90s, BSDi,* a spinoff of Berkeley's CSRG, started selling its own version of Berkeley Unix, and the University of California had been selling its version for years. In 1992 AT&T sued both the University of California and BSDi, claiming that BSD Unix included proprietary AT&T code.

Unfortunately for AT&T, the version of Unix that the company was then pushing, System 5, turned out to incorporate large chunks of code originally written by BSD hackers -- including the TCP/IP stack. Berkeley released all its code under an extraordinarily liberal license -- basically, users could do anything they wanted with BSD code as long as they retained the University of California copyright. But AT&T had stripped the UC copyrights and begun marketing the software as its own. Hackers like McKusick were peeved.

"We had written this code and they were claiming it was theirs," says McKusick, "and that they had the rights to it, and we just flat out didn't believe it. And it pissed us off that they were basically taking our work, that they hadn't paid a penny for, but that they had made money off of because of all the damn licenses that they had sold, and they were now trying to claim it was theirs."

The University of California's lawyers seized upon the opening, countersuing AT&T for copyright violation. After the requisite legal scurrying, the two sides came to a settlement, the terms of which both sides are forbidden to comment on. That settlement, along with a concerted effort by BSD hackers to rid their code of any AT&T "taint," freed the operating system of its last proprietary vestiges.

Third, even if Joy did not piss off his fellow programmers by saying "Read the protocol and write the code," no one who knows him well will deny that it is the kind of thing he *could* easily have said. Joy's colleagues and professors are unanimous in describing him as a fundamentally nice guy. But like so many great hackers, Joy is also almost unconsciously arrogant. And that arrogance has been, historically, a key part of the BSD legend. As a general rule, programmers tend to have a high opinion of themselves. And as a class, Unix programmers are well known for demonstrating their own special blend of high-priest orneriness. But BSD Unix hackers, with some notable exceptions, are especially virulent in their self-assuredness. They aren't wrong very often, and when they are, convincing them of that fact requires several armies and quite a bit of heavy artillery. Indeed, the easiest explanation for why BSD hackers watch in dismay while Linux-based operating systems sweep the world is that, for years, subsections of the BSD community have been endlessly imitating the mulishness that marked Joy's original reluctance to compromise on TCP/IP.

Of course, if anyone ever had a right to be arrogant, it would be Bill Joy. When the University of California received new computer terminals advanced enough to allow a cursor to be mapped to a particular point on the screen, Joy promptly, and speedily, wrote a text editor, vi,* that took advantage of the new capabilities. Vi is still widely used today, standard equipment on nearly all Unix installations.

If the compiler Joy was using didn't satisfy him, he wrote a new one. If the backspace key didn't work correctly in Unix, he rewrote the source code. And so on.

"Bill's very good at taking something," says McKusick, "saying, 'OK, this is what I have, this is where I want to get to, what's the shortest path from here to there?' His code was ugly, unmaintainable, incomprehensible, but by golly it only took him two weeks to do an incredible amount of functionality. Someone asked me once to compare myself to Bill Joy, and I said, "You know, there's really nothing that Bill's done that I couldn't have done, but what Bill did in a year would take me 10.'"

"He was just very good at reading a large body of code and wrapping his mind around it," recalls Fabry. "So he could do major reorganizations of code in a single weekend. I saw him do major reorganizations of the kernel several different times. In the beginning it took him just a few days, while later on it might take him a month. It was wonderful."

Joy called me once on his cellphone. It was Feb. 14, and he was in a grocery store in Monterey, Calif., stocking up on food before heading over to the annual TED (Technology, Entertainment, and Design) conference. Never one to waste a spare second, he decided to combine two chores -- my questions about his Berkeley days and his own need for sustenance. The result gave me a close glimpse at one of Joy's more famous qualities -- his ability to multitask. While answering a question from me, he would simultaneously talk to the cashier or a counter person without skipping a beat. In the middle of a sentence, out would pop the words "fruit salad" or "yogurt with raisins."

Like Unix itself, famous for its ability to perform simultaneous tasks, Joy could allocate portions of his brain to separate jobs at the same time, without appearing to shortchange any of them.

I learned later that his performance wasn't as awe-inspiring as I first thought. Joy, I was told, decides what he thinks about something and then rarely changes his mind; instead, he stores away his thoughts on the subject and is able to regurgitate them on demand, without wasting any fresh brain cells.

People who've worked with Joy say his ability to compartmentalize made him difficult to deal with on occasion. The stories of shouted arguments ringing through the hallways of Berkeley's computer science department are legion. But Joy's stubborn single-mindedness also made him an excellent leader. Berkeley Unix thrived in large part because Joy held his code to the highest possible standard and refused to compromise. Leffler, Joy's second-in-command for most of the heavy lifting involved in pushing out the first versions of Berkeley Unix, says he and Joy had a *responsibility* not to compromise. The U.S. Department of Defense was paying for BSD, and its prospective users encompassed the cream of the computing-science crop.

Intriguingly, Joy doesn't subscribe to the fundamental credo of the Linux movement -- the belief that the strength of open-source software is its ability to tap the energy and enthusiasm of a vast network of volunteer programmers. Linux is built on an egalitarian ethic: Perhaps not every programmer will write great code, but together, all those eyeballs and all those keyboard-pounding fingers will incrementally make their way toward greatness.

But Joy doesn't believe that having more programmers equals better code.

"Most people are bad programmers," says Joy. "The honest truth is that having a lot of people staring at the code does not find the really nasty bugs. The really nasty bugs are found by a couple of really smart people who just kill themselves. Most people looking at the code won't see anything ... You can't have thousands of people contributing and achieve a high standard."

Joy even disputes the commonly held conception that BSD set the model for demonstrating how a large software project could be built via contributions by a widely distributed network of programmers.

"Almost no fixes came in from anywhere else," says Joy, referring to Berkeley Unix. "In fact, most of the stuff I got back was not that great. Remember, there wasn't a real swift network at that time. In later years you got stuff back, but I didn't get much stuff back during that time that I was doing."

Joy may be overstating the case. Leffler and McKusick, while conceding that 90 percent of outside contributions did not meet Berkeley's standards, state authoritatively that significant portions of BSD code came from outside Berkeley. And after Joy's 1982 departure to Sun, the percentage of outside contributions began to rise, in tandem with the rise of the Net. Joy may be overreacting against the new generation of open-source hackers, many of whom frequently fail to acknowledge (or even be aware of) Joy's contributions to the software ecology that underlies the entire free-software movement. Joy says that when he boots* Red Hat Linux, he sees boot-up messages scroll by that he personally wrote 20 years earlier. Joy would much rather talk about his current passions, Sun's Java and Jini, and when he's asked about Linux, he sometimes lets traces of annoyance slip through. Who are these punk hackers, some of whom weren't even alive the first time Joy rewrote the Unix kernel?

"If I had to rewrite Unix from scratch, I could do it in a summer, easily," says Joy. "And it would be much better. A much, much better job. The ideas are old."

But if an increase in the number of programmers doesn't ensure an increase in the quality of code, then why do Linux-based operating systems dominate the market today? And if Joy rejected most contributions from outside Berkeley, why does BSD enjoy a reputation for pioneering the model for collaborative open-source software development?

"BSD was Bill Joy, initially," says McKusick. "He did the distributions and talked about them and pushed them out. He would give talks, and, inevitably, at the end of the talk he would say, 'And if you have any cool stuff, come talk to me.'"

But, Eric Allman hastens to interject, Joy did not invent the concept of freely redistributing software at Berkeley. That "seemed to be an ethos that permeated Berkeley in general," recalls Allman, who had worked in the early '70s on a database project called INGRES that was widely redistributed. And after Joy was gone, that ethos, says Allman, continued.

It is a gorgeous spring Sunday in March. McKusick, Allman and I are sitting around an outdoor table in the backyard of "Chez Oxford," the north Berkeley cottage that Allman and McKusick have lived in for nearly 20 years. The first warm sun after a month of nearly continual rain is beaming down upon us, combining happily with a stream of selections from Chez Oxford's copious wine cellar.

Allman and McKusick can use some relaxing. Allman's company, Sendmail, is in the midst of a hectic round of financing and has just released a major upgrade. McKusick, meanwhile, has sent the entire BSD community into a tizzy by orchestrating a merger between BSDi, a spinoff from the CSRG that sells a proprietary version of BSD, and Walnut Creek CD-ROM, the largest distributor of the FreeBSD distribution. Plans are even afoot for the migration of some code from BSDi's proprietary BSD/OS into the completely free FreeBSD. Interpretations of the merger's significance vary wildly. To some, it's a concession that BSDi's proprietary offerings are making no headway against the Linux onslaught. To others, the merger is a hopeful sign that the days of BSD splintering are over: The community is re-forming again, readying itself for a new round of sparring with other operating-system upstarts.

If Joy was the heart of BSD, then McKusick is its soul, the keeper of the Berkeley Unix flame. After Joy's 1982 departure, Leffler lingered around Berkeley long enough to complete the delivery of BSD 4.2 to DARPA, and then also left academia for the more lucrative embrace of Lucasfilm and, later, Silicon Graphics. McKusick, who had until then been juggling his dissertation with various BSD tasks that Joy talked him into doing, picked up the reins.

To this day, McKusick says proudly, he is one of the only original BSD developers who subscribes to developer mailing lists for *all* the major BSD spinoff distributions. Every night at 2 a.m., one of his computers downloads that day's modifications of FreeBSD and recompiles the system, keeping him excruciatingly up to date. If anyone can claim a comprehensive perspective on BSD's evolution over time, it is McKusick.

Like his fellow BSD coders, McKusick disavows revolutionary fervor or Berkeley radical ambitions.

"The contribution that we made, ultimately," says McKusick, "was in developing a model for doing open-source software ... We figured out how you could take a small group of people and coordinate a software project where you have several hundred people working on it."

McKusick outlines an organizational model that grew up in the wake of the departure of Joy and Leffler. At the center, there is a "core group" -- a set of programmers who control access to the code, by granting or revoking the right to modify or "commit" new code to the code base. Spreading out from them are the "committers" who have that right. Extending out beyond the committers are the general community of developers who submit changes, bug reports and fixes to the committers. Most of today's high-profile open-source projects, such as the Apache Web server and the GNU project, use a similar form of organization. Linux is the major exception. There is no core for Linux -- just Linus Torvalds, followed by a tier of trusted "lieutenants."

"The committers," says McKusick, "were a group of people we trusted to commit stuff that were responsible for things. The notion was that you didn't have all these autocratic controls ... Now, you could have snuck in and committed something to the kernel, some kind of trapdoor even. I won't say we wouldn't have been none the wiser, because we did in fact keep close tabs on what was going on in the kernel, but we didn't need to tell people not to do that; we didn't have to administratively keep them from doing things they shouldn't be doing. We had set up a culture as well as a structure."

Still, 90 percent of the contributions were thrown away; the rest, as McKusick likes to say, "were peed upon to make them smell like Berkeley.

"The trick is that ultimately you find that small nugget of people who are the really good ones, and that's why we have this whole hierarchy, that's why we still have a hierarchy today," says McKusick. "Yes, there are thousands of developers, most of whom honestly couldn't paint themselves out of a wet paper bag if their life depended on it ... But you still give them a place; you don't just dismiss them."

The circle has widened constantly, McKusick notes. Today, FreeBSD has a core of 16, surrounded by nearly 180 committers and thousands of developers. Even so, FreeBSD is dwarfed by the development community contributing to Linux-based operating systems. Which raises the obvious question, one that McKusick has heard hundreds of times over in recent years: How did Linux-based operating systems overtake BSD?

There are some obvious answers. In the early '90s, as the power of personal computers grew steadily, many Unix aficionados began seeking a way to run Unix on their PCs. There were two contenders at the time, 386BSD, a version of BSD created by William and Lynne Jolitz for computers built around Intel "x86" microchips, and Linux-based operating systems. But the AT&T suit, combined with the slow pace of development on 386BSD, placed the whole BSD effort under a cloud. No one knew if AT&T would succeed in quashing BSD altogether. Linux, in combination with the GNU utilities, was protected by the ironclad GNU General Public License -- all the code was free and always would be free.

By the time the AT&T suit was resolved, the snowball ride to Linux was underway. And the future development of BSD after 386BSD did little to persuade hackers to change their minds. Developers dissatisfied with the pace at which 386BSD incorporated new patches split off and founded FreeBSD and NetBSD in 1993. Not long after, an internal dispute within the NetBSD core resulted in the spawning of OpenBSD. Meanwhile, McKusick and other members of the CSRG founded BSDi.

McKusick shrugs off the widely held perception that the BSD community is irreparably shattered. In addition to the just-merged FreeBSD and BSDi, he declares, there are only two other major distributions of BSD, each of which has its own particular focus. NetBSD specializes in porting BSD to different computer architectures. OpenBSD concentrates on security issues.

And how many Linux distributions are there? asks McKusick, rolling his eyes. Fifteen, 20?

I point out that all the Linux distributions share the same kernel, which is overseen by the strong, and generally impartial, centralizing presence of Torvalds. BSD has no center.

McKusick looks a little wistful.

"This is somewhat egotistical," he says. "But I believe that had I been willing to act as the figurehead ... I could have kept the community from splitting. I had basically been in that position for 10 years, maybe 15, and I had really felt that the time had come to pass the mantle on to other people. I had a certain view of the way things ought to be done, and at some point you want to get new blood. Honestly, what I really thought was going to happen was, I knew that things would explode and there would be a lot of different [distributions], and I figured most of them would die off, and one would become the evident new one. And in some ways that really has happened. I mean FreeBSD has, at least by seats, 80 percent of the market, and the other two are interesting, but they are really out in the noise."

Depending too much on a centrally unifying force, suggests McKusick, isn't necessarily a strength. "What happens when Linus Torvalds either dies, gets tired or otherwise steps away from it all? It's a huge weight. He's done, in my mind, a terrible job of building something that lives beyond him. If I got hit by a bus, BSD wouldn't really be affected a lot."

As proof of his assertion, he notes that Torvalds does not rely on a source code control software* program to administer changes to the Linux kernel. Instead, Torvalds reviews each major patch by plugging it into his own development system. But what happens if something goes wrong? How do you roll back to before the changes? Where is the institutional memory, as embodied by software, that will keep a project going when the original leader leaves?

When I e-mailed Torvalds after this discussion for a response, he dismissed the problem.

"The purely technical side of keeping track of the sources can be handled by source control packages," says Torvalds, "but at least, in my opinion, they actually tend to favor the approach of 'Let's put this in now; if it turns out to be a mistake, we can always revert it because we have source control.' And of course, nobody ever actually does clean up anything. Or hardly ever. So I think the *real* problem in computer science is to have quality control before it even hits the distribution, and so far there isn't any other package than the human brain that can do that job."

Torvalds' answer is interesting, if only because the confidence that it reveals echoes the strength of Joy's belief in his own abilities. Perhaps Linux will not be able to develop BSD-like organizational structures until after Torvalds leaves the scene.

Ultimately, the disagreements over organizational strategy between the BSD and Linux camps are petty when compared with the similarities. As McKusick is quick to stress, "The important thing is to be open source, not whether Linux or BSD wins out."

And in that context, one can indeed see BSD, from the Joy era through the McKusick era and beyond, as part of a continuum feeding energy and enthusiasm into the current Linux upswing. Even during the earliest days, Joy, as McKusick notes, wanted other people to contribute "cool things." Ever since then, the trend line has been one in which the circle of contributors has widened steadily. BSD demonstrated how to bring people into the process -- by making them part of the process and by giving them credit for their contributions. As Keith Bostic,*, the leader of the drive to create a version of BSD that included no proprietary AT&T code at all, likes to say, "There is no end to the world of good you can do by giving people credit."

Because what happens? You end up creating tools that everyone can use to be even more productive, to create even-greater structures that encourage collaboration, which in turn unlock even more creativity. That's the essence of free software, and it goes beyond what kind of license protects the software, or whether source control software is employed, or even how arrogant the top developers might be. By opening up the code, Berkeley widened the pool of software possibility.

--------------------

Gage* didn't contribute to the code in Berkeley Unix, but he spent plenty of time hanging around the computer rooms in Evans Hall, gabbing with Joy and pondering the political implications of the computer revolution. Gage, a mathematical statistics graduate student at Berkeley, was also a hippie radical from way back -- involved in both the Free Speech Movement and the antiwar protests. He was a delegate for Bobby Kennedy at the Democratic Convention in 1968 and deputy press secretary for presidential candidate George McGovern in 1972.

After three hours in Gage's favorite Berkeley coffee shop exploring what the Internet means for individual liberty and what role Berkeley Unix played in catalyzing the Internet's growth, Gage rolls his memories back to perhaps the single most-famous moment in Berkeley's history of activism.

He starts quoting the speech Mario Savio gave on the steps of the administration building overlooking Sproul Plaza. He hunches over the table, his eyes blazing with a sudden visceral intensity:

"'There is a time when the operation of the machine becomes so odious,'" declaims Gage, "'makes you so sick at heart, that you can't take part; you can't even passively take part, and you've got to put your bodies upon the gears and upon the wheels, upon the levers, upon all the apparatus, and you've got to make it stop. And you've got to indicate to the people who run it, to the people who own it, that unless you're free, the machine will be prevented from working at all!'"

Gage grins. Berkeley Unix, he proposes, offered a different way forward from the painful agony of hurling oneself into the operation of a demonic crankshaft. Berkeley Unix, with its source code available to all who wanted it, *was* the "gears and levers" of the machine. By promoting access to the source code, to the inner workings of that machine, the free-software/open-source movement empowered people to place their hands on the gears and levers, to take control of their computers, their Internet, their entire technological infrastructure.

"The open-source movement is a free speech movement," says Gage. "Source code looks like poetry, but it's also a machine -- words that *do*. Unix opens up the discourse in the machinery because the words in Unix literally cause action, and those actions will cause other actions."

Savio is dead. The Free Speech Movement is half-forgotten. Few, if any, of its participants would have predicted at the time that a network of computers might prove to be free speech's greatest friend and best weapon. Indeed, Savio's "machine" was in part a metaphor for what he saw as the dehumanization inherent in information technology: The University of California was IBM, the students were punch cards, both literally and figuratively, fed into the machine, not to be folded, spindled or mutilated.

The Berkeley Unix hackers, by helping to unleash the power of the Internet, rehumanized the "machine." Those "words that do" instigated connectivity and provoked communication. Somewhere, Savio is smiling.

**salon.com** | May 16, 2000

**About the writer**
Andrew Leonard is a senior writer for Salon Technology and author of Salon's Free Software Project, an online book-in-progress exploring the history and culture of the free software movement.

# Do-it-yourself giant brains!

From punch cards to Linux, hackers love to tinker and share. Even Bill Gates can't stop them.

- - - - - - - - - - - -

**By Andrew Leonard**

Jun. 22, 2000 | Godless Russians and Communist film directors weren't the only bogeymen who gave Cold War-era Los Angeles the heebie-jeebies. In the early '50s, the Los Angeles Police Department confronted a truly destabilizing threat: pinball.

City authorities, considering pinball machines to be implements of vice and corruption, would break them up for spare parts, consigning the debris to bins in the police department's electronics shop at downtown's Lincoln Jail. Among the parts stockpiled at the jail were heaps of solenoids -- electromagnetic coils used for initiating pinball plunger and bumper action.

Solenoids are really, really good at facilitating on/off flip-flops, mechanically opening and closing circuits depending upon the flow of electric current. During World War II the solenoids were a vital military resource material. But afterwards, they sat unused -- until a couple of teenaged proto-hackers named Phil Cramer and Bill Fletcher came along.

Cramer and Fletcher knew just what to do with leftover solenoids. They would be perfect for constructing Giant Brains!

Like many other aspiring electronics geeks in the immediate postwar era, Cramer and Fletcher's imaginations had been enticed by a book published in 1949 called "Giant Brains, or Machines That Think." Written by Edmund C. Berkeley, an expert in the then-infant field of computing, "Giant Brains" was both a primer and a manifesto.

In language that managed the delicate trick of being exquisitely clear *and* uncompromisingly evangelistic, Berkeley described how a computer works, step by step, instruction by instruction. Employing numerous diagrams, and painstakingly explaining every underlying concept (like "binary" or "register" or "input/output") as if it had never been explained before, Berkeley demonstrated how it was possible to *move* digital information from one "place" to another -- and how a set of on/off switches, if wired correctly, could perform operations on that information, handling such extraordinary feats as the addition of two plus two.

In 1952, Berkeley walked the walk. He built his own computer, Simon, considered by some historians to be the *first* "personal computer," and documented the process in a series of 13 articles for Radio Electronics magazine. Cramer and Fletcher, demonstrating a cavalier attitude towards proprietary information that would become a calling card for do-it-yourself hackers in generations to come, ripped the pages of schematics right out of copies of the magazines at their local library. (As Cramer noted shamefacedly, 50 years later, "We had no duplicators in those days!")

Fletcher had a contact within the police department who let them rummage through the bins of electronics. Following Berkeley's instructions, the two teenagers built a simple solenoid relay-driven computer. It didn't work exactly as planned, but it did *something,* and that was enough. Enough to get Cramer's father, an accountant, to give the youngsters $50 to buy more parts. Enough to encourage them to try again, to build another simple machine that *did* work. Enough, in the case of Phil Cramer, to launch him into a life spent tending Giant Brains, a career of computer programming that continues to this day.

Dig under the surface of your average computer geek and you will find a person in love with the idea of having a Giant Brain of one's own to play with. As computer scientist Dick Karpinski observes, computers, or to be precise, *the act of programming* computers, "is the only way to have socially acceptable slaves."

Over the decades, the opportunity to harness the power of a computer to one's own selfish purposes, whatever those may be, has proven irresistibly seductive. From the '50s kids who gravitated inexorably to IBM mainframes to the Homebrew Computer Club tinkerers who built the first personal computers in the '70s to the Linux hackers exchanging tips and tricks in their user's groups in the '90s, the underlying passion is identical: It is a whole lot of *fun* to be the master of a Giant Brain, down to the very last binary one or zero.

And anything that hinders that mastery is resented. Abhorred. Reviled. Detested. It is no accident that the hacker of the '50s and '60s despised IBM while his counterpart in the '80s and '90s denounced Microsoft. They got in the way! The love affair that so many programmers have with free software isn't reducible to mere respect for an efficient software development methodology. It is also an expression of the programmer's age-old craving to be in intimate control of every aspect of the machine -- and unwillingness to allow any barriers to block the Source.

Of course, when programmers like Cramer got started, there was no real difference between hardware and software. It was all just stuff that made the machine *go*. You might be inputting instructions by floppy disk, or magnetic tape, or punch cards, or paper tape, or even by wiring plugs together -- the medium simply didn't matter: The point was to get the machine to work.

Hackers have always understood this -- and indeed, in the earliest days, there was little need to worry about any separation. Hardware came with software, and you fiddled with both until you got the machine to do what you wanted it to do. But over the decades since the '50s the growth of the commercial software industry, combined with the increasing complexity of software, has worked to divide programmers from the object of their passions. As programmers began to be consigned to smaller and smaller pieces of a larger and larger pie, the job became less and less fun. And when a 19-year-old Bill Gates appeared on the scene in the mid-'70s, admonishing the Homebrew hackers to stop "stealing" his BASIC programming language, the end of hacker happiness seemed nigh.

But true hackers don't let little things like monopolies or near-infinite complexity stop them. As their first line of defense, they have always sought strength in numbers. In the '50s, programmers like Phil Cramer joined a powerful IBM computer user's "club" called SHARE, determined to pool their resources in order to get their machines working better -- and make IBM dance to their tune. In the '70s, the Homebrew hackers likewise came together, in garages and living rooms, to share their expertise and code in the service of their new, desk-sized, power-to-the-people machines. And from the '90s right on through to a new century, free software hackers have also flocked to one another. The programmer-computer relationship may be an inescapably solitary interaction, but the fight for control over every last bit of digital and silicon granularity requires collective effort.

The free software movement has often been described as being by hackers, for hackers, with the rest of us just lucky beneficiaries of the byproduct of hacker obsessions. Even as the free software movement has been organized and corporatized, at root it's still the same as it ever was -- a movement fueled by tinkerers who are constitutionally unable to allow anything to stand between them and their machines.

It doesn't matter whether the tools of their trade are piles of solenoids or copylefted compiler and debugger programs. Hackers will stop at nothing in their drive to play with their Giant Brains. If that means that along the way they'll build the Internet, unleash the personal computer industry and topple Microsoft, well, so be it: They just want to have fun.

The year was 1950. Barry Gordon sat at a desk, a mechanical Friden calculator to his left, punching numbers. He wasn't alone. The "math section" at the Mutual of New York insurance company included rows of similar desks, all featuring Fridens, clerks and actuarial tables. The job was mind-numbing: Punch a number into the Friden. Multiply it with a number from the actuarial table, read out the result, write it down and multiply it by another number. Over and over and over again.

"I get called in by my boss, one day," says Gordon, a fast-talking New Yorker who grins as he starts into a story he has obviously told many times before. "He was the head of the math section, and he says, 'We have a new IBM 604 electronic calculator, and we'd like one of the actuarial students to learn how to use it. What do you think?' I said, 'Sounds terrific!' I didn't know what the hell he was talking about -- I'm 23 years old, on my first job -- but I said it sounded great."

Just by studying the manual, Gordon mastered the essentials of the 604 well enough that he was able to fix it when it malfunctioned a couple of weeks later, even without ever having previously laid eyes on the machine. Within days, he was transferred into the "tabulating division." He felt blessed.

"We used to sit in the math section," says Gordon, "churning out rates and forfeiture values and dividends and whatnot, and now I'd come across this fabulous electronic machine that does stuff miraculously at lightning speeds, and I'm thinking, I'm in on a revolution! I'm going to free mankind from the drudgery of sitting at calculators!"

There was just one little problem. After setting up the 604 to do whatever particular operation was required, the machine would spit out "reams and reams" of punch cards that in turn would be fed into the "tabulator" which would then promptly produce printouts of neat columns of numbers. But only numbers -- there were no headings, no indications of what the numbers referred to. The tabulator could only handle numbers. To make the information "camera-ready" for later copying and distribution, those headings had to be added.

"So one day I'm wondering, where do those headings come from?" says Gordon. "And one evening, on a break during overtime, I go down to the math section where all the people had been freed from the drudgery of hitting the calculators. And they are sitting there, cutting and pasting headings on the pages to make camera-ready copy! And I thought, this is what I have freed them for? There is something terribly wrong with this whole system! At that point I almost quit computing to do something else."

Mort Bernstein's living room explodes into laughter. Barry Gordon is sharing his story with a roomful of aged programmers -- veterans of the industry who all started out in the early '50s. The breadth of experience shared by the nine men and one woman gathered together in this Santa Monica suburb is impressive. They have literally seen it all.

Today, they trade tips on avoiding Microsoft Outlook viruses, or dual-booting their personal computers with Linux. But their experiences are grounded in the biggest of "big iron" -- cumbersome IBM mainframes like the 701 "Defense Calculator" and its successor, the 704. Most were employed by Southern California aerospace companies or think tanks: RAND, Lockheed, Douglas, North American Aviation. Notwithstanding Gordon's insurance tales, these programmers were accustomed to solving problems of slightly more significance than, say, handling Web site traffic or calculating life expectancies. Irwin Greenwald simulated the H-bomb explosion on Eniwetok at RAND, for example, while Westinghouse's Frank Engel modeled the possible deadly malfunction of a Nautilus submarine nuclear reactor.

Some, like Phil Cramer, are still working -- he runs a company that specializes in software for auto dealerships. Others, like Frank Wagner, the patriarch at this meeting, have been retired for several decades. Still others, like the host, Mort Bernstein, amuse themselves with hobbies: Bernstein is trying to program an "emulator" for RAND's Johnniac computer on his PC, but is finding it to be quite a task. Not only does he have to emulate the Johnniac's processor, but he also has to emulate the punch cards that fed it, and the punch card reader, and so on.

The youngest doesn't look a day over 65, but that isn't slowing anyone down. Jokes, anecdotes about long-forgotten computers and disputes about ancient programming lore zing back and forth. They delight in insulting each other, their long-dead colleagues, and most of all, IBM. Programmers of any age appreciate the art of the deftly delivered sarcastic jab, or, in the parlance of these coders, the "cut-down." When Gordon recalls a going-away party at IBM for a manager who appears to have been spectacularly nondescript, and quotes the parting toast "John, your departure will fill the void that was created when you first got here," the laughter is hardly polite -- it's an uproar.

The occasion is a reunion for former members of SHARE, an IBM user's club. Founded in 1955, SHARE may have been the first official computer user's group *ever*. Certainly, for a time, it was one of the most powerful.

SHARE was an outgrowth of an earlier collaboration between Southern California aerospace companies called PACT -- the Project for the Advancement of Coding Technologies. PACT's goal was to write a compiler for the 701, IBM's first commercial digital computer. SHARE started out as a proactive measure to prepare for the arrival of the 704. Although the name predated a later reverse-engineered acronym -- "The Society to Help Alleviate Redundant Engineering" -- from the very beginning SHARE aimed to save individual programmers from the sorry fate of writing basic code for essential tools that had already been written by someone else.

"We wanted to do something about this silly business of everybody programming their own square root routine," recalls Wagner, who managed programmers at the Mustang fighter plane manufacturer North American Aviation. The 704, which filled a large, specially built room with its card punch, card reader, printer, CRT, magnetic tape reels, magnetic drum, magnetic core storage, central processing unit and operator console (each of which was a separate machine), was not only hard to use, but also didn't come equipped with much in the way of pre-installed "software." Instead, the 704 arrived with a 103-page "Principles of Operation" manual, an assembler and some very basic utility programs, such as a single punch card "bootstrap loader" to get the machine started.

These huge mainframes were more than welcome to the aerospace companies -- they were essential. In the tense days of the Korean War, the pressure was on the defense industry to keep up with every move made by the Soviets and Communist Chinese. But, as Wagner recalls, it was getting harder and harder to tote up the necessary numbers involved in high-tech aeronautic design. "We were drowning in arithmetic," says Wagner, a genial, albeit occasionally sharp-tongued man treated by the other programmers with a mixture of respect and friendly deference. "Whenever an aircraft design changed, you had to go way back and start all over again."

Cooperation, for the purpose of the elimination of "redundant" effort, was the order of the day. The companies didn't share all their software -- they kept their structural analysis programs to themselves. But they did share the tools that they used to build such programs. Such cooperation only made sense to programmers who hated wasting their time, and who, according to Gordon, "tended not to be company loyal or commercially oriented."

"They were loyal to their profession," says Wagner.

And their drinking buddies. Certainly, as an example of programmer pragmatism prefiguring such open-source standbys as the Apache Web server, or Linux itself, by decades, SHARE is historically noteworthy. But SHARE wasn't just about saving money and time -- it was also about having fun with your community. Going to SHARE meetings was a blast, and not least because every night there was an open bar "SCIDS" meeting: The "SHARE Committee for Imbibers, Drinkers and Sots." SCIDS was where the action was, where the "technical" people gathered to lubricate themselves on alcohol and algorithms. As John Backus, the principal author of IBM's Fortran programming language, noted during a 1980 commemoration marking 25 years of SHARE, "There were two principal pleasures of SHARE: Blasting IBM -- giving them hell for fouling up and not giving them what they wanted -- and the second activity, known as SCIDS."

Programmers labor under a stereotype that maligns them as anti-social shut-ins. But while it is true that the act of programming is solitary toil, programmers are also intensely social. It's hardly an exaggeration to suggest that the Internet was built mainly so that programmers would have a place where they could get together and chat about their favorite science fiction novels; but before the Internet, you had to gather at conventions, or conferences. That's where you shared notes with your colleagues, that's where you learned your craft and honed your programming chops. That's where you figured out how to fix your Giant Brains.

Because even if building your own Giant Brain is fun, it sure isn't easy. It's hard work, and you're going to need help. And you're not going to care if the guy you're knocking back whiskeys with, discussing the finer points of matrix inversion, works for your competitor. The point is to get the machine to function properly, to get the buzz that comes from observing the translation of your will -- your computer program -- into action.

These early programmers were insatiably curious tinkerers and inventors eager to dissect every function of their machines, and to learn every clever workaround or deft coding maneuver that would help them get their job done. The advent of proprietary code that could not be tinkered with, could not be taken apart and reconfigured like a bunch of Lego blocks, was extraordinarily annoying. When they started to hack, whether as teenagers wiring together solenoids, or as defense contractors responsible for fending off the Russkis, hardware and software were both just means to an end. Often, in order to change the programming in one of the early computers or tabulating machines, you had to rewire the darn thing yourself. Typically, this was done by means of a "plug board" -- a panel much like the telephone operator's switchboard with which Lily Tomlin once wrought havoc. If you wanted to change the sequence of operations, you pulled out a plug connecting a wire to another point and plugged it in somewhere else.

It was only later, as software became increasingly less *physical,* that it was even possible to obstruct programmers from handy access to digital innards. And for the first decade or so after the introduction of the commercially sold computer, few people even realized why that would be advantageous.

"There was no economic value in software," notes Bernstein. "No one recognized software as an economic entity at the time."

IBM's domination of the computer market partially explained the failure to see software as a revenue generator. IBM was opposed to selling *anything.* Its business model was based on leasing hardware. Leasing made it easier to plan for the future and calculate ongoing revenues. Hardware, software, support personnel -- it was all leased, all bundled together in one monthly package. The inclusion of the software was a carefully thought-out strategic measure that locked customers into IBM. If you used IBM hardware, then you used the software that came with it. And if you spent thousands of hours mastering that software, then when it came time to renew your lease, you stayed with IBM -- why on earth would you want to invest thousands more hours getting up to speed on someone else's software, even if new hardware from another vendor might be technically superior?

Bundled software didn't automatically mean bundled source code -- which was one reason that a group like SHARE was necessary. In addition to providing a means for IBM users to share the code that they wrote individually, SHARE was also an important lobbying tool for getting IBM to make changes, add features and otherwise respond to customer concerns. Especially during the early years, SHARE's membership represented a majority of IBM's most important customers. When it complained, IBM listened. It was a user's "club," explains Frank Wagner -- in the sense of the "club" being a large stick useful for beating IBM about the head with.

During their reunion, the SHARE programmers talked about IBM in precisely the same way that hackers today rail against Microsoft. IBM was the eight-gazillion-pound gorilla. IBM pioneered FUD and the practice of "vaporware," forced bad technology down customer's throats and got away with it all because it owned the market. Even the SHARE members who at one time worked for IBM -- such as Gordon, who put in 25 years at Big Blue, and Bernstein, who did a consulting stint there -- joined the IBM derision. IBM got in the hackers' way.

But in one of the great unpredictable ironies embedded in the history of software, IBM's decision to *stop* bundling its software with its hardware turned out to be the catalyst that really launched the commercial software industry -- and made proprietary control of source code start making sense to the corporate computing world. In 1969, says Mort Bernstein, as the Department of Justice began to prepare its epochal antitrust suit against IBM, IBM decided to engage in some proactive defensive maneuvers. By unbundling IBM's software offerings, and charging for them separately, IBM hoped to avoid the accusation that it was unfairly leveraging its monopoly control of the market for mainframes.

And in a flash, the commercial software industry was born.

"There was no software industry to speak of until that moment, and then it began to burgeon," says Bernstein.

IBM's act of unbundling presents an intriguing contradiction. Opening up competition undoubtedly contributed to faster growth and greater opportunity for companies looking to make a profit in the world of computing, with a consequent increase in the number of jobs for programmers. But individual programmers ended up becoming further divorced from the hands-on finagling with bits and bytes that made their jobs fun. Not only were they denied access to the source code of commercial programs, but their own responsibilities were steadily being curtailed.

Fun, for programmers, was on the run. As programming tasks became larger, employing thousands of programmers at a time, programming duties became more and more tightly segmented and regimented.

"Programming went from an individual craft to a 'professional' activity that had to be managed," says Bernstein. "As computing became a fundamental necessity in every aspect of commerce and industry, more needs arose. Programmers were asked to provide reasonably precise estimates of the time and resources that would be needed for each development. Based on this, budgets and schedules were produced. Management expected the result to match the estimates within reason. This is a real sea change, from a freewheeling craft performed by 'artistes' to a tightly managed activity supposedly performed by 'professionals.'"

To some observers, the changes in the industry could be placed in the broader, and more suspicious, context of the move to "scientifically manage" office work of all kinds. Following principles established by Frederick Winslow Taylor, an inventor who influenced the creation of assembly-line manufacturing, management began treating programmers as if *they* were the plugs to be moved around the plug boards, rather than creative visionaries with minds of their own. As Joan Greenbaum, a former IBM programmer, writes in "Windows on the Workplace":

> "The first step that management took to gain control over the programming workforce was to divide the conceptual work of programming from the more physical tasks of computer operations. Although this division was put into effect in the aerospace industry in the mid-1950s and subsequently used by companies that had defense contracts, it wasn't until the mid-1960s that it spread elsewhere. By 1965, when IBM began installing the general-purpose System 360, both the more expensive hardware (a large mainframe computer) and the easier to use software (an operating system that could be controlled through commands rather than operators working switches), gave upper and middle managers room to begin enforcing the separation of programming from operations. Operators were to stay in the 'machine room' tending the computer, while programmers were to sit upstairs and write the instructions. Those of us in the field at the time remember feeling that a firm division of labor had been introduced almost overnight."

Proprietary code. Tight divisions of labor. An end to the freedom and fun of the golden age. By the late '70s, things had become pretty bleak for the corporate programmer.

But help was on the way.

"Then the PC came along and the ball game changed again," says Bernstein. "Who has to account for the time they use on their PC? No one! It's an appliance on my desk that is of the same nature as my telephone. And the world of the programmer had come full circle. The PC is what makes the open-source community as freewheeling as it is. Linux and open source could never exist in the world of tightly controlled corporate computing."

It is not every day that you meet someone who has an antique rubidium atomic clock ticking away the minutes in his study. What's even more unusual is when the clock -- a fairly large timepiece, about the size of an electric dishwasher -- doesn't even stand out. Among the old DEC minicomputers, teletype machines and obsolete tape drives that line the walls of Bob Lash's back room, the atomic clock, which was once a fixture at California's Vandenberg Airforce Base and is practically indestructible, is almost an afterthought. Compared to the home-made computer that Lash built while going to high school in the '70s in Palo Alto, it is hardly remarkable at all.

Bob Lash enjoys the honor of being one of the youngest attendees at the first meeting of the legendary Homebrew Computer Club in Gordon French's garage on March 5th, 1975. That alone is testament to some sublime geekiness. Silicon Valley's Homebrew Computer Club is a staple in the annals of computing history. Two of the best books about the history of the personal computer, Steven Levy's "Hackers" and "Fire in the Valley," by Paul Freiberger and Michael Swaine, devote hundreds of pages to Homebrew. Steven Wozniak, co-founder of Apple, is a Homebrew alumnus, as is Lee Felsenstein, the designer of two earl, and beloved personal computers, the Sol and the Osborne. The role of the Homebrew hackers in creating the personal computer, in delivering the power of a Giant Brain to you and me, is difficult to overestimate.

But Bob Lash's study takes personal geekiness to new heights. The study could be mistaken for a museum of computing, although it is actually anything but. The oscilloscope, soldering iron and bins of electronic parts offer a clue. These relics, the DEC PDP-11/23 and PDP-8/I, are in working order. This room is a living shrine, proof that Lash, a programmer who is very much alive in the present -- his luxurious home in the hills above Redwood City was paid for largely by the sale of a Web chat software company that he and another Homebrew alumnus founded -- hasn't lost sight of his past.

Some of the artifacts in this room are part of an attempt to re-create Lash's very first encounter with a computer, when he was only 6 years old. He was taking part in an experiment at Stanford aimed at studying the teaching of mathematics to young children. After one of the sessions, he was allowed a peek into a nearby machine room -- "Basically, a room full of cabinets, with a console filled with blinking white lights, and there was a tape drive with a pair of reels sitting on it: a DEC tape drive. It made a huge impact on me. There was a fellow there and he pointed to a key on a console and told me to push it, so I did, and when I did, suddenly a pair of reels started spinning. The thought that you could hit a key on the console and then make something else happen, that you could control something remotely by a computer, was to me an entirely new idea."

I observe to Lash that it is not uncommon for 6-year-olds to be confronted with "entirely new ideas." He smiles, but it is nonetheless clear that the encounter was life-transforming. Right next to Lash, as he speaks, sits a DEC tape drive, not quite the same model as the one he encountered as a youth, but close enough. The teletype machine connected to the PDP-8/I is also nearly identical to the one he recalls. More, perhaps, than any programmer I have ever encountered, Lash has concentrated his intellect and his skills on capturing the reality of his own human-computer interaction. Barring cyborg surgery, he is a man who has gotten as close to the machine as possible.

And like his fellow Homebrew hackers, he's convinced that everyone should share the joy. In his study there are also four personal computers. One old 486 IBM clone runs Linux and is hosting a Web server. It is also connected to the teletype machine that feeds instructions to the PDP-8/I. Once Lash straightens all the bugs out, Web surfers will be able to run their own old PDP code on Lash's machine, if they still have any, or test out any new code that they might want to take for a spin.

But why would anyone want to do that? The PDP-8/I was popular in its time, but compared to today's computers, it's a joke -- a three-year-old personal computer is about a thousand times as powerful. Sure, it looks cool, with its flashing lights and switches and knobs, but it's not really what one would call a high-powered productivity tool.

But Lash has his reasons: "What I'd like to do is give younger programmers the chance to get some hands-on experience at communing with the machine at its deepest level," he says calmly. "With today's processors you really don't have a feeling for what's going on inside, they're like black boxes. But in a machine like the [PDP] 8 everything that is happening inside is shown on the indicator lamps, all the registers are displayed and you can actually get right down to the register level and understand what's really happening step by step. Nowadays there is almost no ability for people to do that any more. They've lost touch with that."

*Right down to the register level* -- the places within a computer's central processing unit where individual bits of information (or the location of those bits of information) are stored. Berkeley's "Giant Brains" book laid out the workings of the computing machine; Lash wants to make those workings transparent to a Web-enabled public. The benefit, he argues, is better software.

"I think that programmers who can see what is happening all the way through down to the deepest levels of the machine have the best understanding of what's really happening," says Lash, "and that has a big influence on how they think about what they are going to do and how they are going to do it. In this age of code bloat and inefficient systems and sluggish Windows machines that are shimmying and shaking and smoking, I think that if today's programmers had a better understanding of what's happening under the hood they could and would build more efficient systems."

But again, it's not just about creating technically superior software, it's also about connecting to the fun that's at the heart of the computing experience.

"I think having an appreciation and an understanding of what's happening is helpful, but even more important than professional benefits, I think that for young programmers, it fires up a sense of enthusiasm and wonder for what they're doing. It isn't just a dull boring job -- 'Let's crank out another application because we have a deadline.' It's really a miracle when you see what happens, and it's a wonder. A kind of spark and spirit can be of enormous benefit to the work that you are trying to accomplish. I guess it's more about motivation than any real technical point."

Lash recalls attending Homebrew meetings with exactly the same mixture of nostalgia and glee that the SHARE programmers remembered experiencing at their SCIDS meetings. His desire to understand his computers, and his belief that everyone would benefit from a similar understanding, is also directly connected to the explicit free software ideology that has sprung up in computing in more recent years. Lash is a believer in Linux-based operating systems -- as is his 11-year-old son Elliot, who, when he isn't playing Starcraft, is running a Linux-based operating system called Phat-Linux.

For Lash, who reverently shows me a loose-leaf binder filled with hundreds of pages of all the lines of code he wrote two decades ago for his home-made computer, the code *is* the machine.

Most of the Homebrew hackers felt the same way. Which might just explain why they got their dander up when Bill Gates wrote them a nasty letter in 1976.

On February 3, 1976, less than a year after the founding of the Homebrew Computer Club, a 19-year-old named Bill Gates, who titled himself General Partner of a then little-known company called "Micro-Soft," wrote a screed titled "An Open Letter to Hobbyists."

Bill Gates and his partner Paul Allen had written a version of the BASIC programming language for one of the first mass-produced personal computers, the Altair. Without it, there wasn't a whole lot that you could do with an Altair. But to Gates' dismay, he had discovered that less than 10 percent of all Altair owners were paying for a copy of his BASIC -- instead, hackers were making their own copies and giving them away. To Gates, this was outright thievery.

"As the majority of hobbyists must be aware," wrote Gates, "most of you steal your software. Hardware must be paid for, but software is something to share. Who cares if the people who worked on it get paid?"

"Who can afford to do professional work for nothing?" continued Gates. "What hobbyist can put three man years into programming, finding all bugs, documenting his product and distribute for free? The fact is, no one besides us has invested a lot of money in hobby software ... Most directly, the thing you do is theft."

Like the Homebrew Computer Club, Gates' letter is an indelible icon of computer history. For some, it marks the birth of a billionaire, outlining in bald terms the psychology of one of the most successful businessmen of the 20th century. For others, it foretold the death of the hacker dream that information should be free. And to still others, it marks the line drawn in the sand between the world of free software and the empire of Microsoft. Want to know why so many hackers despise Microsoft? Read the letter.

But didn't Gates have a reason to be angry? Weren't the Homebrew hobbyists stealing BASIC? Well, in a sense, sure.

"We would say 'bring back more copies than you took,'" recalls Lee Felsenstein, remembering Homebrew Club meetings held in the Stanford Linear Accelerator Center (SLAC) auditorium. "Altair BASIC showed up first as a paper tape that had been ripped off or liberated in late '75 and was being passed around or copied -- a teletype could copy it. Sometimes I would hold up the black board pointer and say 'put them here,' and people would skewer their rolls of tape on the pointer."

I meet Lee Felsenstein, designer of the Sol and Osborne personal computers, in a nearly unfurnished office in Palo Alto which he is using as a temporary work space while he looks for a new job. Just a few weeks earlier, his previous employer, the Paul Allen-funded Interval Research think tank, closed its doors. While he negotiates a purchase involving his credit card information on his cellphone, I examine the only part of the room that shows signs of life. On a large workbench stretching along one wall are the tools of his inventor's trade: a fancy digital oscilloscope, a heat gun, a power supply, and trays upon trays of silicon chips: the basic building blocks of the modern computer.

By now I have read "Giant Brains" and have learned the basics of how information can be moved from register to register, and what kind of operations can be enacted on that information. At Bob Lash's home, I have had the chance to examine both his homemade computer and the reams of code he wrote to make that computer work. Now, I'm looking at a more contemporary version of the same concept, the silicon-based chips that embody those operations. And I am realizing: hardware, software, chips, wires -- for me too, it is becoming difficult to make any clear distinctions. And as I probe Felsenstein's memory of the letter from Bill Gates, I begin to see exactly why the Homebrew hackers got so mad at being called thieves even as they blithely admitted that they were copying someone else's copyrighted software.

Did Felsenstein remember the moment he read that letter?

He rolls his eyes.

"Yes, absolutely," says Felsenstein. "I read it aloud from the floor of the Homebrew Computer Club. To great derision. I read it to the multitudes assembled in the SLAC auditorium. Everybody thought that was hilarious, and they were damned if they were going to send them $500."

Felsenstein draws a detailed picture of the moment. The SLAC auditorium holds 275 people, he says, but it was only about two-thirds full.

"The people involved were not at the top of any stratum," says Felsenstein. "They were second string and below ... They were people who wanted desperately to have access to computers. The majority of them worked in the electronics industry in Silicon Valley, and some in the computer industry, but they were not permitted access to computers at their work."

So when MITS, a company in Albuquerque, New Mexico, announced the arrival of the Altair, one of the very first inexpensive personal computers, they jumped at the chance to buy it. And then were almost immediately disappointed. The Altair was little more than a box of parts that barely worked. For one thing, it came with no devices for getting information into and out of it, which meant that the Homebrew hackers were faced with quite a bit of extra tinkering, at extra cost.

"At the time when the Altair personal computer was being delivered it was found to be difficult to get it running," says Felsenstein. "It was poorly designed in several ways. And once you finally got an Altair hooked up together, it didn't do anything."

"The machine is not complete until the software defines what it is doing," says Felsenstein. "I view software as another component of the system. Our view of this was, this is like the last part of the machine that everybody sweated bullets to not only buy but to learn how to put it together and so forth, and what the hell, what business do they have trying to jack us up for another $500? This makes the thing go and I want to make it go ... It seemed like a kind of a bait-and-switch at the time to say 'You can buy a computer for $297 but oh, sorry, all you get is a bunch of parts that don't give you any I/O [input/output] and oh, sorry, once you get that set up at great cost to yourself you don't get any software that lets you do anything but you can pay more for that."

"We didn't really know who Bill Gates was," continues Felsenstein. "He seemed to be involved with MITS. He started out as a MITS employee ... As far as we could tell that's all that he was. Somebody had come along with a BASIC and attached themselves to MITS and said 'Now we can really clean up because these bozos don't realize they need software for their boxes.'"

"And then he comes in with this letter," says Felsenstein, "saying 'we haven't gotten the kind of money we wanted and you guys are all crooks -- most hobbyists steal their software.' Well, you know, for people who were out about ten times what they thought they were going to be out when they answered the ad for the Altair, the concept of thievery is a little different. [We said] 'You guys in effect stole our money, and now you want another $500? I'm sorry, we want our computers to work.' The general feeling was, let them ask nicely, don't call us crooks to begin with, we see who the real crooks are."

The letter, says Felsenstein, drew a line that could never be erased. Once and for all, Bill Gates declared that he was not a hacker -- that, on the contrary, hackers were his enemy.

"The oppositional stance kind of presaged everything else," says Felsenstein. "We wanted somebody to say, 'Look, I am one of you and here is what is going on.' He removed himself from our society and our culture with that letter and with his subsequent actions."

I ask Felsenstein if he recalls the reaction of the Homebrew audience the moment he read the line "most of you steal your software."

"There was much hooting," says Felsenstein. "We had a good time laughing at that letter. And as I like to say now, what a shame that Bill Gates didn't get his money. He could have been a contender. "

Pandemonium reigns at the Four Seas Restaurant in San Francisco's Chinatown on the evening of June 15, 1999. The line to get in stretches down the stairs from the second floor, out the front door, and spills onto the sidewalk. The buzz of conversation is half-anxious, half-excited -- it's as if the assembled crowd is waiting to see some hot new band, but isn't sure that there's enough room for everybody.

They haven't come for the food. They have come to attend, of all things, a Bay Area Linux User's Group [BALUG] meeting. Of course, it's not your average BALUG event. While the meetings have become steadily more crowded over the past few years, in close correspondence to the rise in popularity of Linux-based operating systems, they aren't usually this kind of a madhouse. But tonight, Linus Torvalds has come to speak, and in June of 1999 in San Francisco, that means a major techno-cultural media event is at hand.

By the spring of 1999, Linux fever is reaching hitherto unimaginable heights. One distributor of Linux-based operating systems, Red Hat, has just filed to go public, and others are soon to follow. Everyone is in love with Linux -- Wall Street, the press, the pundits. On this particular evening, 400 people will stuff themselves into the Four Seas Restaurant, in part to hear Torvalds, but also to celebrate their own great good fortune. Programming is fun again, and free software is a big reason why.

It's actually kind of a shame that Torvalds' appearance has skewed the BALUG meeting from its normal, less media-friendly fare. These meetings follow directly in the tradition of both the Homebrew Computer Club and even the SHARE IBM user's club -- although few, if any, of the attendees have ever heard of SHARE. It's time to get together, drink some beer, chow some middling Chinese food, and talk shop. Figure out how to get your system working, how to get your giant brain tuned and optimized the way you want it, without any taint of proprietary software obscuring the machine's inner mysteries. At closely related "installfests" organized by various LUGs, you can even bring your computer and get experts to walk you through the often torturous process of getting a free software operating system running correctly on it.

This is the kind of forum in which Torvalds excels, where he can engage in his favorite interactive question-and-answer format with technically knowledgeable people. I have seen him speak to crowds numbering 20,000 and higher and field reporter's queries at press conferences with aplomb, but he is most comfortable when being asked detailed questions about arcane aspects of software and hardware. Torvalds has a way of being both self-deprecating and utterly sure of himself that works well with audiences. When he jokes that "I am basically lazy" or that "if I don't understand something, I think it is bad," he gets a big laugh, but no one underestimates him.

Sitting at one of the big round tables that dot the banquet floor, listening to a smattering of hackers exchange jokes and chatter about the various Linux happenings of the day, I'm impressed at the general sense of good feeling that flows through the room. People are happy to be here. They're happy to be talking about obscure matters of symmetric multiprocessing performance and source control management. They're especially happy to be doing what they do, which, more and more, means hacking on Linux-based operating systems. A year later, when I hear Lee Felsenstein muse about what makes open-source software fun, I immediately think back to the night at the Four Seas.

"The open-source movement is a direct descendant of the Homebrew Computing Club," says Felsenstein, "motivated by the same things -- that very seductive goal of creating what never has been before, and doing it in a sharing community where coercion is absent and the joy and the beauty of the creativity is manifest in everyone. It's very, very powerful."

It's the same fun that is recalled by SHARE programmers when they reminisce about SCIDS meetings. But there's one big difference.

Torvalds gets one of his biggest laughs of the night when he makes a glancing reference to the commercialization of Linux. Referring to Linux hackers, he says, "They can laugh at the stupid company who will pay them for what they would do [anyway] for free."

Few audiences could be more receptive to his quip than the hackers assembled in front of him. The evening is being sponsored by two companies, VA Linux and Linuxcare, which have for the past six months been hiring every member of any Linux User's Group they can get their hands on. Not many of the hackers may have imagined when they first started coming to the meetings that their hobby would suddenly make them highly sought-after professionals, but that's certainly the case now. Linux hackers are hot. Who can afford to do professional work for nothing? Right now, a good many companies are trying to figure out new answers to that exact question.

The fun is back in programming. The industry has indeed come full circle. Once again, hackers are being paid to do just what they want to do -- only this time around, everything they produce is made freely available to the general public.

In June of 1999, or June of 2000, for that matter, it's impossible to say how long this hacker-blessed happenstance will continue. Perhaps the present period is a new golden age of hacking doomed to decline or morph into some less blissful stage. And of course, not all jobs at Linux-related companies are ideal, nor is there any certainty that giving software away will wrest control of the computing industry from Microsoft. But for the hackers twiddling their chopsticks, happily continuing a cherished tradition of community and mutual self-help, such quibbling about what has not yet come to pass is a waste of time.

"There are a great many things that all of us could do much better if we could only apply what the wisest of us knows," wrote Edmund C. Berkeley in "Giant Brains, or Machines That Think." His point was an argument for the worth of the computer as a concentration of human knowledge made available to all humans. But it also can be taken to represent the value of the people who create those computers, who work together to share their wisdom. All of us could do much better, he seems to be implying, if we are given the freedom to SHARE.

--------------------

**Read Chapter One of the Free Software Project**

**Join the discussion on this chapter**

------------

**Sound Off**
Send us a Letter to the Editor

GO TO: Salon.com >> Technology

Salon Search About Salon Table Talk Newsletters Advertise in Salon Investor Relations

Arts & Entertainment | Books | Business | Comics | Health | Mothers Who Think | News
People | Politics | Sex | Technology and The Free Software Project
Letters | Columnists | Salon Plus | Salon Shop

salon.com > Free Software Project April 20, 2000
URL: http://www.salon.com/tech/fsp/2000/04/20/chapter_six_part_1

# Finland -- the open-source society

In the icy, cellphone-mad birthplace of Linux, networks rule. It's a matter of survival.

------------
BY ANDREW LEONARD

**Author's Note:** *Writing an online book presents some unique challenges -- and opportunities. With this installment, I decided to break with boring linear order and go straight to the first half of Chapter 6, rather than the expected Chapter 2. Why? Because three weeks ago I spent a week in Finland, the birthplace of Linux, and I wanted to write about it before the memories faded. And if I write it, why wait to publish it?*

Back in the summer of 1993, I went to Finland for all my software needs. I never questioned why. I just knew that if I wanted a free copy of Tetris, or an image viewer for looking at jpegs* of Madonna, or an application that would make Chinese characters readable in my e-mail, I headed to Finland -- or rather, to the Internet address "nic.funet.fi." It was just another one of the lovable eccentricities of the old Internet. For some reason, one of the world's largest repositories of freely redistributable software could be found in a small Northern European country previously most famous for sauna baths and Sibelius.

Today, Finland is famous for other reasons -- notably, for being the original home of both Nokia, the world's largest and most profitable manufacturer of mobile phones, and Linus Torvalds, the creator of Linux. Finland is also now widely hailed as one of the most "wired" nations on the planet (as judged by mobile phone and Internet usage). Once known mostly for exports of pulp and paper products from its vast forests, Finland now enjoys the unexpected honor of being acclaimed throughout Europe as a role model for the so-called Information Society.

Nokia receives the lion's share of the credit. An aggressive, fast-growing, fully global company that makes Microsoft look like an old fuddy-duddy, Nokia is hiring new employees at the rate of 1,000 per month. The company so dominates the economy of Finland that a sudden drop in its stock price sends jitters through the entire nation. But which came first, the Nokia chicken or the Finnish egg? Is Nokia the reason that glued-to-their-phones Finns often seem like some strange new cyborg beast -- *homo mobilis telefonicus?* Or does the much-touted Finnish openness to new technology explain Nokia's surge to the forefront of the global economy?

Finland's love affair with high technology runs deep. The closer you look, the less remarkable it seems that a 21-year-old undergraduate at the University of Helsinki cooked up some code that ended up throwing the entire software industry into turmoil. For Linux is far from Finland's only contribution to Internet culture: To an extent way out of proportion to its size, Finland has bequeathed unto the Net a valuable and culturally rich set of essential tools.

In addition to the software library at nic.funet.fi, there is also the much beloved, albeit now somewhat archaic, Internet Relay Chat, or IRC* -- one of the first popular open-source programs to enable real-time online conversations between globally dispersed Internet users. There's also ssh,* a program hugely popular with hackers and geeks that helps ensure secure online transmission of data. And, perhaps most notoriously, there's that Net icon of the early '90s, Johan Helsingius'* "anon.penet.fi" anonymous remailer* -- a tool that, until the Church of Scientology convinced Finnish authorities to shut it down, allowed the paranoid or privacy-conscious to post to newsgroups and send mail in complete, cryptographically protected anonymity. These contributions, and even Linux itself, may be just a drop in the bucket of the hundreds of thousands of software programs hackers have uploaded to the Net. But the Finns' predilection for creating such tools reveals an acute understanding of the nature of a networked, open-source society.

Finland's contributions to the Net pose a conundrum. When Finns asked me why I had come to their out-of-the-way nation, I gave them two reasons. The first was obvious -- I had come to dig up background information on Linus Torvalds. So I visited the university where he first started hacking on Linux. I talked with people who had studied under Torvalds' maternal grandfather, a well-known professor of statistics, and who were used to watching his father, a television reporter, deliver dispatches from the war in Chechnya. I even hung out in the neighborhood bar his mother is known to frequent.

And everywhere I went, people were eager to gossip. Did I know what his mother said about Linus' love life in last Saturday's afternoon newspaper? Was I aware that his parents had been members of the Communist Party? What did I think about the fact that in the late '60s his student radical father, Nils Torvalds, had infuriated his other grandfather, a conservative newspaper editor, by posing on the cover of a magazine holding a machine gun? And could I please tell them how much Linus was worth? A hundred million? A billion?

But Linus wasn't the whole story. I also sought the answer to a question I must have been subconsciously mulling over ever since I waited for that first software program from 10,000 miles away to creep across my 2400 baud modem in 1993. Why Finland? In the 21st century, there's hardly a nation in the world that *doesn't* want to be a role model for the information society. What made Finland so special? Was it an accident of history, the luck of the draw, or some more complex intersection of cultural evolution and the activist will of an entire people? More to the point, was it possible that the deep structure of Finnish civilization encourages an open-source way of life?

Harri K. Salminen points at a nondescript PC half hidden under a rack of shelves, practically invisible in a room full of much larger computers. It's possible, says Salminen, pursing his lips in a geekily confident way that suggests total familiarity with the millions of dollars of hardware surrounding him, that the hard drive on this computer served the files I downloaded from nic.funet.fi seven years earlier. It's not much to look at now -- it isn't even connected to the Net. Instead, an impressive array of state-of-the-art SGI Crays and DEC Alphas hum contentedly. In a high-tech country, this, the central server headquarters of the Center for Scientific Computing, is one of the highest-tech rooms -- the root node of the Finnish Internet.

It's also the room in which Linux was first made available to the general public, which makes it one of the original source points for open source -- and as close as you can get to a holy shrine for free software. For years, says Salminen, the demand from outside Finland for downloads of free software from nic.funet.fi required the imposition of bandwidth transfer "speed limits" to keep the network usable for Finns. As I strolled among the computers, half-listening to Salminen, the chief "coordinator" for nic.funet.fi, I could almost see the world-spanning network, an infinitely tangled spider web of connectivity, spiraling out from this one node, delivering one of the Net's most infectious packages of software to countless other nodes. I wondered what a real-time look at the scrolling log files of nic.funet.fi might have revealed back in August 1991, as hackers from all over the globe arrived, downloaded, left, and then used Linux and all the other free software tools that make up a Linux-based operating system to build their own nodes from which to spread the digital word. A room full of computers is hardly a romantic sight, but here, at Linux's original launching point, I felt as physically close to the soul of the Internet as I had ever been.

Salminen seemed bemused at my sincere intensity. A typical Finn and a typical geek -- fluent in six languages, an expert in C* and Perl* programming -- he chewed over my questions as if he wasn't quite sure they were worth asking. He had no problem providing the nuts and bolts of the history of the Internet in Finland: In 1988 Salminen was personally in charge of setting up the link between the FUNET network and the NSFNET backbone of the Internet, in conjunction with four other Scandinavian nations He could tell me exactly who had first uploaded Linux to nic.funet.fi -- a student named Ari Lemmke -- and on what date commercial sales of Internet connections began in Finland -- 1993.

But why was Finland so wired? Why had Finns made so many contributions to the Internet? Why was the country so gaga over all forms of telecommunication -- beginning with the phone?

There is no single answer. But there are some telling data points. First, the Finnish infatuation with the telephone is no new phenomenon, no mere byproduct of Nokia's dramatic rise to prominence. Finns have been crazy about phones from practically the first moment they could get their hands on them. In 1896, Mrs. Alex-Tweedie, an English travel writer, noted that "Finland is full of phones." Angel Ganivet, the Spanish consul in Finland in 1896-97, observed that phones were almost as common as kitchenware, and devoted an entire chapter of his book on Finland to the "excessive" interest Finns had in technology. It also has become an inordinately popular national obsession (at least among the telecom-literate people I interviewed) to mention at least once a day how there were more than 800 separate telephone companies in the country during the 1920s and '30s.

Finland is a sparsely settled country -- a little over 5 million people are sprinkled across a land mass 1,000 kilometers long from north to south. An attraction to phones is therefore an understandable outgrowth of local geography. But a historical misstep by the Russian tsar also played a crucial role. During the 19th century Finland was an "autonomous Grand Duchy" under the rule of the Russian Empire. (Prior to that, for seven centuries Finland had been ruled by its neighbor, Sweden.) Finland's multitude of phone companies was a legacy of the Tsar's decision to declare the telegraph a militarily essential device -- and the telephone, on the other hand, little more than a toy.

Wary of the possibility that the Tsar might change his mind, the Finnish government chose to grant licenses to operate telephone companies to all applicants -- in marked contrast to the practice of most other nations, who ensured that telephone operation was a tightly controlled state monopoly. The reasoning of the Finnish government was as follows: It would be much easier for the tsar to renege on his decision if all he had to do was simply close down or otherwise take control of one state enterprise, rather than hunt down hundreds of independent companies.

When you have 800 telephone companies in a country that, in the 1920s, only had a population of 2 million to 3 million people, you are forced to become expert in interconnection technologies. As a result, Finns understand networking.

I was reminded of this constantly during my week in Helsinki, in both small ways and large. The Ministry of Foreign Affairs, which was helping to coordinate my visit, (and which paid for my airfare to Finland) gave everyone I interviewed a copy of my schedule, so everyone knew who I had talked to already and who I would be talking to next -- they even used me as a conduit for messages between each other. I was absorbed into their network as effortlessly as a well-configured Web server handles a newly arrived connection request.

In Finland, the mobile phone has evolved into much more than just a symbol of Nokia's corporate power -- it is now a vehicle for the etiquette of personal encounters. Examining a new acquaintance's phone -- for new features, for style or just for the heck of it -- is as natural as shaking hands.

And provision of cutting-edge wireless services isn't just future hype, it's a cornerstone of the national economy. Near the end of my stay, I had dinner with Jarkko Oikarinen,* the inventor of IRC. He told me that that very day he had decided to quit his job as a programmer in the University of Oulu's medical school in favor of joining a startup to work on wireless applications for mobile phones. I hardly blinked. Join the crowd, Jarkko. The question, in Finland, isn't "who is doing the interesting work in wireless networking?" but rather "who isn't?"

So that most telling stat about Finland -- 5 million people, 4 million mobile phones -- begins to make sense. But what about the Net? Where's the built-in connection to programming?

Salminen shrugs. It's the long winter, he says. Finland's the northernmost country in Europe -- nearly a third of the nation is within the Arctic Circle. There's just not much else to do besides hack.

In Finland, all roads lead to winter. There are, in fact, no fewer than three winters in Finland: autumn winter, high winter and spring winter. I arrived in Helsinki at the end of March, smack in the middle of spring winter. During the week I visited, the temperature rarely rose above freezing; the bays and inlets that snake into and through Helsinki were clogged with ice, and remnants of high winter snow still survived in parks and by the sides of roads. But it was sunny, and people were cheerful -- because in Finland, when the temperature gets as high as freezing, spring is at hand.

Finns dote on their winter; it is built in to the national psyche, a point of both pride and misery. Not for nothing is Finland the world leader in naval ice-breaking technology. Finns will sniff, slightly annoyed, if you dare even to question whether their winter really is appreciably worse than that of their Scandinavian neighbors. Most Swedes and Danes live further south, they note, while Norway's long coastline is warmed by the Gulf Stream. It's no accident that Finland invented the sauna -- keeping warm is a national pastime.

Winter, says Risto Linturi,* explains everything about Finland.

Linturi is Finland's leading candidate for national digital visionary, though at first listen he doesn't sound much like the smooth snake-oil salesmen that pass for digital evangelists in the West. Instead, he creaks like a glacier, ponderously, crunching granite outcrops of speech into gravel as he moves forward, contemplating each newly spoken word as if it were some kind of bizarre mutation. Formerly the chief technology strategist for the Helsinki Telephone Company, Linturi is currently a venture capitalist whose company provides modest seed capital for high tech start-ups in Finland.

But a visionary he is: A voracious reader of science fiction (his favorite author is Robert Heinlein), he lives in a high tech "smart house" whose doors and appliances he can control with his mobile phone.

"As long as we have been living in Finland," says Linturi, "we have been very interested in staying alive. And that is way more high tech than anyone today can realize."

Linturi says that the key to surviving Finland's long, dark winter is the efficient optimization of information. How many cows do you intend to keep alive through the long dark months? At what point do you kill the cows you won't keep alive in order to maximize your remaining food stocks? How will you then keep the meat from spoiling? How much time do you devote to chopping wood? What are the most energy efficient techniques for insulation and cooking?

Finland was no home to Vikings, observes Linturi, raging across the rest of Europe in search of easy plunder. Death came not from war, but from winter.

"You did not get killed because you could not defend yourself," says Linturi. "You got killed because you could not supply yourself."

As proof, Linturi points to Finnish folk tales. In the Kalevala, a compendium of myths and legends assembled by budding Finnish nationalists in the 19th century, you find no helmeted Valkyries or hammer-swinging Thunder Gods. Instead, Finnish folk tales, asserts Linturi, revere the "lore master." The protagonists of the Kalevala are Ilmarinen, the smith, and Väinämöinen, the lore singer. The antagonist is Louhi, the black witch of the North.

"All of these heroes [and villains] are characters whose main capability is information -- storing or utilizing information," says Linturi. "To survive through the winter in a country like Finland, you don't need heroes and you don't need power. You need information. You need lore."

The image of the lore master instantly conjures up a vision quite at home in the world of programming -- that bearded, long-haired, Unix guru who is equally comfortable in the midst of reams of C code or a game of "Dungeons & Dragons." Programming is all about lore, and all about optimization. Indeed, one of the criticisms of Linux and other open-source/free software programs is that they do not represent innovation, i.e. the creation of something wholly new, but merely optimization, the tuning of something old.

But there's another, more significant correspondence between the survival lore of ancient Finns and the nature of information in the digital era. Survival lore doesn't automatically lend itself to a proprietary model of information acquisition. In other words, unlike warrior lore, survival lore does not diminish in value if other people acquire it. You might want to keep a better design for a longbow or sword to yourself or your clan, hoping to gain an arms-race advantage over your competitors. But you gain relatively little by keeping to yourself a better food preparation technique or algorithm* for calculating the proper ratio of wood-chopping to hay-gathering to livestock-slaughtering. Quite the contrary: If you share your winter survival optimization techniques with others, they may well be more likely to share their information with you.

Not for nothing has Finland been dubbed "a nation of cooperators." The term is not always interpreted favorably -- centuries of existence as a buffer state between East and West have forced Finland to always watch its step, particularly in the Cold War era, when conservative Americans dismissed Finland as a Soviet lackey, while the equally suspicious Russians glared at every Finnish gesture of accommodation with the capitalist world. In such a historical context, Finns have excelled politically at offending no one, or "kissing both asses," as one young hacker put it. But cooperation does not automatically imply quisling-style collaboration. A talent for cooperation is also an implicit recognition that in numbers, there is strength -- that collective action can achieve mighty things.

The same sense of cooperation feeds into Finland's pride at being a successful welfare state, although in this it is not significantly different from other Scandinavian nations. But the fact that Finns are generally willing to pay the high taxes necessary to provide free child care, health care and schooling (through the university level) certainly hasn't hurt the development of a high technology infrastructure. Torvalds himself notes that Finland simply isn't as cut-throatishly competitive a society as the United States -- there's more of a sense that everyone benefits from a comprehensive safety net.

"You must visit Marshall Mannerheim's grave," Marja Erola told me. Erola, a program manager at TEKES, the National Technology Agency of the Finnish government, gazed at me with a quintessentially Finnish stare, at once direct and earnest. "It will help you to understand Finnish society."

I followed her advice. My last day in Helsinki, as the sun was setting and the ice that had thawed during the day was just beginning to harden again, I walked from my hotel across the peninsula straddled by Helsinki to the western side, adjoining the Gulf of Finland. There, amid stately rows of fir and birch, a large graveyard stretches along the shoreline. It is a cemetery intended to be visited and very much alive -- more a park than a place of death. As I strolled along the manicured graves, contemplating the Finnish and Swedish family names, I spotted couples walking hand in hand, or staring out at the sea.

At the western-most edge the cemetery opens up into a broad field, broken up by arcing lines of graves. Each grave is marked by a 1-foot-square marble plaque lying flat on the ground. The graves are for soldiers who died in World War II fighting the Soviets. It's a bleak and bloody reminder of the last century.

In the center of a field stands the tomb of C.G.E. Mannerheim, aka Marshall Mannerheim, the general who led the wartime defense of Finland. Elected president of Finland shortly after the war, Mannerheim is considered one of Finland's greatest heroes. His tomb, amid the men he led, is much larger than those of his soldiers, but the only real difference is in scale: it too, is another flat, square, solid block of marble.

Marja Erola told me that Mannerheim's insistence on being buried among his men was both proof and symbol of what she termed the relative "lack of hierarchy" in Finnish society. It's a cultural trait the Finns are inordinately proud of. Finland is a phenomenally homogeneous nation, both in terms of ethnicity and class; the only significant minority is Swedish-speaking Finns, who comprise about 6 percent of the population (and whose number include Linus Torvalds -- although as one Finnish free software hacker told me, "He's still a good guy, even if he is a Swedish-speaker").

The absence of hierarchy is partially explained by the legacy of Swedish rule -- for centuries the Swedes provided most of what passed for an aristocracy, while the Finns were nearly all one class of quasi-peasants. Another explanation points to the individualist ethic of those Finnish peasants -- the Ostrobothnian lumberjack, for example, carving his livelihood out of the forest, owing fealty to no lord, is an icon of independence.

Whatever its origin, a disrespect for hierarchical divisions has now been enshrined as a Finnish value. And no Finnish entity demonstrates the power of that trait better than Nokia, the company with the nicest coat racks in all of Helsinki.

Imagine a country where all the new buildings built in the previous year are offices for one corporation; where all the computer science graduates are hired to work at that same corporation, and where half the gross national product is produced by that one corporation. That's only a slightly exaggerated vision of the role played by Nokia in Finland.

Not all Finns are overjoyed by Nokia's overwhelming presence in Finnish life. Nokia's corporate motto is "connecting people." But to entrepreneurs unable to hire quality engineering talent for their own firms, and computer science department chairs worried about the fact that all their students are focusing on research areas related to wireless communications, the more accurate slogan is "collecting people." And to employees who work ever-longer hours, struggling to maintain a competitive edge against fearsome rivals like Motorola, Ericsson and Siemens, the bitter joke is that Nokia is actually in the business of "disconnecting families."

It's tough to stay ahead in the global economy; long hours are part of the price. But Nokia executives are convinced that their company has other advantages. One of the cherished tropes of Nokian corporate folklore is the idea that any lower-level employee can pull out his or her mobile phone and dial up the boss, all the way up to the CEO. If the CEO doesn't answer, no problem: you are then empowered to make your own decisions, to act upon your own initiative. This corporate mind-set is codified in Nokia's own internal communication practices. At Nokia, says Erkki Ormala, a director of technology policy, "We don't ask who is your boss, we ask to whom you are reporting."

Ormala is my last interview in Helsinki, in a plush conference room at Nokia House, Nokia's 4-year-old headquarters, a dazzlingly blue building coated in a sheath of sparkling glass. Finland is good at meetings -- I've been in a great many conference rooms during my week and I've almost always been impressed. The coffee is always fresh, a variety of pastries and sweet breads are invariably laid out on the table, the lightest of taps on an intercom button summons help near-instantaneously. But Nokia was in a class by itself: The plate glass windows looking out at the bay offer the most sublime view, the catered lunch is the tastiest, the electronic conferencing equipment is the most state-of-the-art.

Nokia House even has the nicest coat racks! In Finland, spacious coat racks are an integral part of the architecture of every building I visit. At larger corporations, they occupy a considerable amount of real estate adjoining the reception area. Finnish design esthetics are famous for pleasingly matching up form and function -- by the time I hang up my coat at Nokia, I'm not surprised to notice that the coat hangers are the coolest I've ever seen -- angled bars of steel that wouldn't look out of place as construction elements in a nuclear power plant.

Like his company's coat hangers, Ormala's presentation is flawless. He has a ready, polished answer to every question except one: When I ask him whether Nokia's emphasis on open standards for communications technologies is an example of a more progressive approach to flourishing in today's global information economy than Microsoft's strategy of controlling standards, he quickly pleads no comment. In today's world, Microsoft and Nokia are both competitors and cooperators. In Finland, the Internet is already integral to every new model mobile phone -- the software that runs that Net-to-phone interface will be the battleground of the next generation of operating systems and wireless hardware.

Listening to Ormala field my questions in an English that is more precise than my own, it's easy to see how Nokia has made such huge strides over the past half decade. But when Ormala quantifies some of his company's growth, telling me that the corporation is hiring 1,000 employees a month, that the average age of all Nokia's employees is a little over 30, and their average tenure with the company is just under three years, I boggle. Ormala himself has only been with the company for a year, after stints in the Finnish government and as chairman of an OECD working group on innovation and technology.

How is it possible for a company hiring 1,000 20-somethings a month to even pretend to itself that it is effectively managing its own growth? Nokia, I say to Ormala, sounds like a runaway train. But Ormala just smiles.

"Necessity has created the need to learn how to integrate people," says Ormala. "I was integrated in a process where I learned to know the organization and the organization learned to know me."

There is no better way to compete in a fast-moving, global economy than by decentralizing operations and depending on local initiative. Nokia, says Ormala, is better suited than most companies to succeed in this economy because of its anti-hierarchical culture. One of the reasons Nokia is growing so fast, he suggests, is precisely because of that decentralization -- more than half of Nokia's employees work outside of Finland, responding to local conditions as they see fit.

As I listen to Ormala, I am struck by the similarities between Nokia and Linux. Although both have a clear center, Nokia's CEO Jorma Ollila and Linux's Linus, both also depend on subordinates to be able to solve their own problems -- apply their own patches, as it were, to the bugs that turn up in their everyday activities. Both are fundamentally global enterprises, taking advantage of advanced telecommunication structures to create new ways of doing business -- or creating code. And both operate according to values that may well be rooted in the deep structure of Finnish culture.

Lack of rigid hierarchy, respect for the value of shared information, an openness to new technology: what do all these qualities have in common? They nicely complement the task of flourishing in a networked environment. Ultimately, it doesn't matter whether Finnish folk tales or Swedish rule or long winters really constitute some kind of deep cultural programming. Finns aren't automatons, required by their history to act in specific ways. What is indisputable is that Finns have convinced *themselves* that they like to play with new gadgets and distrust hierarchies. And that becomes a self-fulfilling prophecy. In a world where new things tumble out one after another in an ever-accelerating rush, having convinced yourself that you thrive on newness is an amazing tactical advantage.

Of course, all these qualities require one more magic ingredient to make them meld perfectly together -- self-confidence. And intriguingly, although from my perspective Finland seemed to be overflowing with confidence, many Finns told me that the country had actually long suffered from a serious self-esteem problem.

In Helsinki, Russia is never far away, physically or psychologically. The apartment building in which Linus Torvalds grew up is on a street named St. Petersburg -- that Russian city founded by Peter the Great was once (before WW II) only about 25 kilometers distant from the southeastern border of Finland. The old Russian embassy, a huge, classically designed building with an imposing stone-carved hammer-and-sickle presiding over all who come near, is just a few blocks away.

Torvalds' own parents were both members of the Finnish Communist Party. It's one of the amusing paradoxes of free software: Linus Torvalds, a paragon of pragmatism, currently working in the heart of Silicon Valley for a highly capitalized start-up that epitomizes the way business is done in the free market global economy, grew up in atmosphere drenched in socialist practice and rhetoric.

Neither of Torvalds' parents are communists any longer; both are journalists, his father for television and radio, his mother as a translator. And it certainly wasn't out of the ordinary for upper-middle class Finns to be communists in the 1960s. At the time, at least as far as the West was concerned, Finland was clearly part of the Soviet sphere of influence; in Finland itself, there was always a nagging worry as to whether the country would be the next Hungary or Czechoslovakia -- doomed to watch Soviet tanks roll through the capital city. Finns, who as far back as the 19th century had a reputation for stoic resignation, kept quiet and worried about their image.

Risto Linturi likes to tell a joke -- "A Finn, a Russian, and an American go to the zoo, and see a huge elephant. The American thinks, 'I could sell this elephant for a lot of money.' The Russian thinks, 'This elephant could feed a lot of people.' But the Finn wonders, 'What does the elephant think about me?'"

By the end of my stay in Finland, the Finns were asking me as many questions as I asked them. I got the feeling, sometimes, that I was the elephant. They would rather know what I thought about them than explain themselves to me. But when I told them that the country struck me as a pretty happy place, that everyone was exuding self-confidence from every pore, they acted surprised. Hannu Puttonen, a filmmaker working on a documentary about Linux, was positively perplexed -- Finland, he said, has always seen itself as the "sad country." Even the very first page of the Kalevala refers to the Finnish homeland as "the luckless lands of the North."

Perhaps my impressions were skewed, he suggested, by my selection of interview subjects among the movers and shakers in Finland's information society. Nokia scientists and computer programmers were bound to be complacent, given their current success. But the country still has an unemployment rate of almost 10 percent, noted Puttonen, and memories of a deep recession at the beginning of the 1990s are still sharp.

That recession was caused, in large part, by the end of the Cold War and the breakup of the Soviet Union, which until the early 1990s accounted for 25 percent of Finland's exports. Ever since then, Finland appears to be exhaling a huge sigh of relief -- relief that may be easier to see from the outside looking in.

Mato Valtonen is an aging rocker who now runs a company called WAPit, which specializes in wireless application services for mobile phones. Until quite recently, Valtonen was the lead singer and front man for the Leningrad Cowboys, a Finnish rock band with a reputation for punk/postmodern troublemaking. In 1993, Valtonen recalled, the Leningrad Cowboys hired Russia's Red Army Choir to go on tour with them, performing American pop songs. At an outdoor concert in central Helsinki, where 200,000 people attempted to force themselves into a space that could fit only 70,000, one could hear, says Valtonen, the sound of Finland relaxing. The sight of the Red Army Choir singing Lynyrd Skynyrd's "Sweet Home Alabama" suggested that Russian tanks were no longer threatening the border.

Two years later, Finland beat Sweden in the ice hockey world championships. (Remember 1980 -- when the Americans beat the Soviets during the Lake Placid Olympics? Multiply that by about a thousand orders of magnitude. Sweden ruled Finland for 700 years! Naked men were dancing on top of police cars in downtown Helsinki!) Esa Tihala, director of e-business at ICL, a one-time computer manufacturer moving rapidly into Web-only e-commerce solutions, cited that moment as another psychological breakthrough point. "We never had won anything, before," said Tihala, his face glowing. "We didn't think we *could* win anything."

But now Finland is winning everything. Red Army Choirs, ice hockey champions, mobile phone megacorporations and open source avatars -- Finland's psyche is in pretty good shape. Of course, the global economy is nothing if not fickle. When stock prices drop on the NASDAQ exchange, stock markets all over the world react in kind, not excepting Finland.

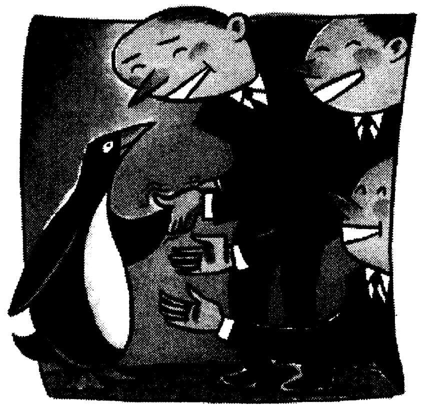
But Finland in the 21st century is far from luckless. In today's world, you've got to find your niche, your one thing that you do better than anyone else. Finland's niche turns out to be the network. Not a bad gig, if you can get it.

And not a bad way to explain the power of Linux, either. When Linus Torvalds stands up in front of tens of thousands of people at a major computer industry convention, he projects an aura of untouchable self-confidence and yet at the same time an eminently approachable openness. And why shouldn't he? He hails from a nation of cooperators who revere the power of information -- of lore -- in their myths and legends, who seem to be born knowing how to take advantage of the unique potential of the network. He comes from a land where open-source attitudes are as natural as the frozen lakes and endless Arctic nights.

The people who do best in a networked world have a great deal in common with the people who devote themselves to open-source software: they distrust rigid hierarchies, they thrive on shared information and they are eager to try new things -- new methodologies, new software, new gadgets, new ways of doing business. It turns out to be no mystery, after all, that something like Linux and someone like Linus Torvalds have emerged from the "sad country" of the North. It was an inevitability.

**salon.com** | April 20, 2000

**To print this page, select "Print" from the File menu of your browser**



Chapter 7, part 1

# How Big Blue fell for Linux

When open-source developers and IBM took gambles on each other, free software showed it can flourish in the heartland of corporate computing.

**By Andrew Leonard**

Sept. 12, 2000 | The kitchen is the receptionist at Collab.net, a software start-up in the South of Market neighborhood of San Francisco. No one is present to greet an inquisitive visitor walking through the open door on the fourth floor of a nondescript building -- just stacked cases of Snapple fruit juice and giant bags of pretzels; a refrigerator and a sink; a coffee machine and a water dispenser.

The ambience screams of youthful coder necessities. On top of the refrigerator, huge boxes of Trix and Cap'n Crunch line up like crates of ammunition. Next to the sink sits a large jar of Twizzlers. From the large, open room stretching beyond the kitchen, a seductive slither of spooky trance music pulses -- inviting, and yet at the same time a little intimidating. People who work in this kind of environment are almost *too* cool.

A few concessions are made to the sensitivities of the less hip -- the potential investors or clients from the old world of computing who might drop by, looking to dump a million bucks here or there. Along one wall stands a free-standing rack packed with hundreds of issues of business/high tech magazines -- The Industry Standard, Business 2.0, Red Herring, Fortune.

The magazines may not make good marketing material right now. Collab.net, the brainchild of open-source star Brian Behlendorf,* aims to make a business out of, he says, "distilling the principles of open source." But at least half of the covers of these new-economy bibles are screaming dire, boldface warnings about the current dot-com meltdown, including Wall Street's sharp turn away from Linux-related stocks in the spring and summer.

It's a good thing the office tunes are soothing, because jangled nerves are suddenly everywhere in that strange land where free software and dot-com start-ups mix. In the summer of 1999, Red Hat's IPO, occurring right in the middle of a packed LinuxWorld convention, sent attendees into a dither of delight. But in mid-August, no less an authority than the New York Times takes advantage of another LinuxWorld convention to declaim about how Wall Street is souring on Linux.

The Gray Lady is a bit late to the story -- the trade press has been hooting about declining valuations since early in the spring, and competitors have long become adept at using the stock price declines of companies like Red Hat and VA Linux as evidence that the open-source upstarts don't pose a threat to established, proprietary software enterprises. Critics of free software are also muttering about continuing delays pushing back the release of the next version of Linux, and the failure of Netscape's Mozilla project to release a usable browser.

But the Times may also be a bit overeager: Barely one week after August's LinuxWorld, Linux companies like Caldera and VA Linux handily beat analyst estimates and watch their stock prices surge. Linux investors suddenly rejoice.

And yet, those who take heart in a one-day surge are just as guilty of overeagerness. Both cynics and Pollyannas are like marks suckered into a New York huckster's game of three-card monte. While they busily stare, striving to follow the movements of the dealer's hand, they never notice that Times Square around them is meanwhile being transformed from pimp heaven into Disneyland. Sure, companies in the business of *selling* Linux may have questionable prospects -- but the open-source revolution is still in full effect, rebuilding the software industry from top to bottom, forcing everyone to adapt.

Corporations involved in the software industry are exploring open-source software, some with the enthusiasm of bodysurfers losing themselves in the roaring surf, others with the timidity of diffident waders in a lagoon full of sharks. They are by no means unified in their approach as an industry sector, or even internally within a single company. But there are executives and engineers at all of these companies who believe that an extraordinarily clear business case can be made for open-source software: Figure out how to make it your friend, before it starts dancing on your grave.

To see this process in action, you don't need to look further than the computer industry's venerable giant, IBM -- which has become perhaps the best corporate friend open-source software has ever had.

The morning of Dec. 16, 1999, started out as it usually did for Linas Vepstas. Warming himself against the Austin, Texas, winter cold seeping through his drafty, unheated house, he settled down to read his e-mail and drink his coffee. Sometimes the ritual was a relaxing way to ease into the day; other times, the caffeine and the messages would combine to get him bouncing off the walls. Like most hackers, Vepstas lived his life via e-mail -- his main hobby, at the moment, involved coordinating a major Linux project via online communiques with an international band of similarly dedicated coders.

But his e-mail on this winter morning was neither soothing nor invigorating. It was paralyzing. In just a few lines, all the work he had done for the last year and a half evaporated.

The message came from Alan Cox,* a man widely considered to be the second most influential hacker in the Linux community, after Linus Torvalds. From his home in Swansea, Wales, Cox -- his independent contractor's salary paid by Red Hat -- fulfilled an extraordinarily important role as maintainer of the Linux kernel.* While Torvalds was off working on the next version, Cox spent much of his time consolidating bug fixes and patches to older versions -- keeping the kernel up to date and secure, extending its ability to interface with new kinds of hardware.

The message read as follows:

> They finally delivered code. A decent-looking SMP kernel, console and some networking stuff. Glibc, gcc, binutils, gdb patches.
>
> The kernel stuff is in 2.2.14pre14. I'll forward you the other patches if you want.

Alan

"They" meant IBM. And the "code" was a package of extensions and patches to the Linux kernel and other associated free programs, created by a team of IBM programmers in Böblingen, Germany, near Stuttgart. The additions made it possible to run Linux-based operating systems on IBM's top-of-the-line mainframe computer, the System 390.

Vepstas stared at the message from Cox in shock.

I tried to read a few more e-mails," he says, "but found I couldn't concentrate. I bit my lip, I bit my tongue. I'd long ago learned the lesson of regretting one's words, and wasn't about to regress. A measured response would come later."

Vepstas was irritated for several reasons. He had heard rumors about the Böblingen "skunk works,"* but nothing definitive, nothing as impossible to ignore as the actual delivery of code. He felt he should have been better informed. At the very least, he thought he deserved first notice of IBM's official efforts.

For the better part of two years, Vepstas and a small cadre of programmers had been writing their own version of Linux capable of running on the 390. The project was called Bigfoot, and it had attracted a fair amount of admiring -- even if somewhat perplexed -- attention from the Linux community. Mainframes were "big iron," the biggest, most powerful and expensive computers available this side of a supercomputer -- the kind of computers that a bank or an airline would use to run its operations.

Once upon a time, mainframes had ruled the computer roost. But toward the close of the 20th century, mainframes had lost some of their grand allure. During the '90s, the network came of age, and scampering, decentralized agglomerations of PCs made lumbering mainframes seem like evolutionary losers. Heck, all you needed was a cheap PC, a Linux-based operating system and the Apache Web server, and you could host your own Web site, right?

Right, and wrong. By the end of the '90s, mainframes, much to the surprise of some observers, were back in favor. Only now they were being called "servers." The reasons for their comeback? Running Web sites had become big business for many companies, including a significant portion of IBM's traditional Fortune 500 customers. The pure processing power and ironclad reliability of the monster mainframe was once again beginning to look attractive, as the Web increasingly became part of an infrastructure channeling massive quantities of mission-critical data in torrents that would drown even the mightiest of PCs.

Vepstas wasn't particularly interested in the resurgence of the mainframe market; he wanted to hone his technical chops. To get Linux to run on a killer machine like the 390 would be a nice hack indeed -- he would have to write his own compiler and assembler and master the tricky job of porting an entire kernel to a new hardware architecture. As Vepstas notes, the 390 had "a fabled, legendary status as a computer design, and I figured it was damned high time I learned it."

But Linux had originally been designed to run on cheap Intel PC hardware. And the 390 already had two different proprietary-to-IBM operating systems designed just for it, considered by many IBM engineers to be the culmination of decades of the best work of IBM research and development talent. Getting Linux to run on an IBM mainframe was not only technically challenging, but also seemingly pointless -- like using a cheap, tinny transistor radio as the sound system in your brand new BMW.

As it turned out, IBM had very good reasons for wanting Linux running smoothly on the 390 -- as well as for keeping the project quiet while it was still incomplete. But on the morning of Dec. 16, nothing could have prevented Vepstas' shock from quickly turning to anger. As he wrote to the Linux/390 mailing list on Dec. 18, after IBM announced to the world what it had demonstrated to Cox two days earlier:

"I personally have spent many evenings and weekends working on this project, without pay, for just the glory of it," wrote Vepstas. "Although I cannot speak for others, others have also invested their time. I am not happy; I take IBM's actions to be a personal affront."

Eight months later, Vepstas has let his grudges subside -- he's immersed in a new project, GNUCash, a free personal-finance management program. He's moved on. But at the time, Vepstas could be excused for feeling slighted. One of the motivating forces fueling free software hackers is the reputation game -- the better the hack, the more cred you get in your community. But by obliterating his project, IBM had eviscerated his chance for such cred. His own background as a programmer who had worked for IBM for 10 years made the blow hit especially hard.

Hurt feelings were only one part of Vepstas' discontent. Of larger concern was the fundamental contradiction between a "skunk works" project -- carried out in secrecy, not only from the rest of the world, but also from the rest of IBM -- and the basic philosophy of open-source software. Ideally, open-source software involves the coordination of large numbers of programmers, thus reducing unnecessary duplication of work and improving the chances for peer review. Even more fundamentally, open source is open: Everyone gets to look at the code. But IBM's programmers had done their work in private. Was the company attempting to gain the advantages of Linux without allowing the collective participation essential to a smoothly functioning (and ideologically correct) open-source effort?

"Without conversations and communications, development cannot be coordinated," Vepstas declared to the list. "We could have gotten more done, been further along. Due to bad management decisions within IBM, time was wasted and money was wasted. I believe that these bad decisions were made because the managers do not understand the open-source development process. This is why I write this screed."

"Lack of transparency and secretive development leads to other problems besides just wasted and duplicated effort," continued Vepstas. "It directly harms the open-source community, and directly harms the corporate image and credibility of IBM. I have a four-year-old son who has recently learned the phrase 'trust me.' He says 'trust me,' and then, minutes later, is doing something bad again. We are trying to reason with him: 'You know that is a bad thing to do. Why did you do it? Next time, think before you act. Do the right thing. If you always do the right thing, then we can trust you.' (Unfortunately, the only effect this has had is that he's stopped saying 'trust me.')"

Perhaps the most incongruous aspect of Vepstas' unfortunate experience with IBM was its context. IBM boasts a reputation for playing by open-source rules that surpasses that of any other major computing corporation. Even Richard Stallman, a man utterly unafraid of castigating the high and mighty, has little negative to say about Big Blue. Indeed, the eyebrow-raising announcement, in the summer of 1998, that IBM was basing its WebSphere family of e-business products on the Apache Web server program did more to create a relationship between the world of open source and the established corporate software industry than any other single act.

Today, IBM executives like to portray the Linux-for- the-390 effort as part of a coherent strategy aimed at coming to grips with vast changes overtaking the software landscape, changes it saw coming way back in 1998. As Bill Zeitler, the general manager of the Enterprise Servers division, declaims -- "It is in IBM's strategic interest to work as closely with the open source community as we can... This is not a fad -- this is a profound disruptive change in the way that software will be developed and deployed." So Linux for the 390 is not only the crown jewel in a current initiative to support Linux on every level of IBM hardware, from Thinkpads to mainframes, but is also the logical conclusion to a three-year journey of rapprochement with the world of free software.

The truth is a little more complicated.

The story of how IBM made friends with free software hackers, from the early days when it dipped its toes into the Apache Project to its current headfirst plunge into Linux, is not the story of a carefully executed strategy. It is instead a tale of contingency, luck, a few committed engineers and a few canny executives. Its twists and turns hinge on the results of combating agendas, political maneuvering and software ambition. At its most mundane, it is a story that hints at how the battle for dominance over new software markets will be waged over the next few years. At its most metaphysical, it is a story that illuminates the contradictions inherent in the very concept of a "corporation."

It's all too easy to see a company like IBM, or Sun, or even Microsoft, in the terms of the legal fiction that is represented by the word "corporation," to anthropomorphize it as a "body" and give it attributes -- evil, good, brilliant, stupid, spunky, lumbering. But the modern corporation is far too fragmented and balkanized to personify in such simple, unitary terms.

The 390 Project provides a perfect example. The engineers responsible, a group of young Germans, wanted to, in the words of team leader Boas Betzler, "do something totally strange. We were just a group of techies that wanted to find out how smart we were."

"In the beginning, we really did not think about how big an impact we could make," says Betzler. "We always wanted to demonstrate the power and capability of the mainframe and then give it to someone who would know Linux and see the machine and use it and say 'Wow, that's a really big Linux.'"

A higher tier of engineers, those who defended the project in turf wars within IBM (or hid knowledge of it from competing factions), saw a chance to make a strategic move that would help boost 390 sales -- by ensuring that the 390 would be a platform comfortable with the vast array of Unix/Linux applications available. Even further up, executives jockeying their way up the corporate ladder placed bets on Linux as a means of gaining advantage in the never-ending political warfare that exists in any large company. And at the top, even CEO Lou Gerstner played a role, determined that if IBM was going to support open source, it wouldn't do so in a halfhearted manner.

Even Linas Vepstas, after his initial rage had subsided, acknowledges that IBM's internal politics made it impossible to allow the Burblingen team to interact with the wider open-source software community.

"I think many people don't realize how much the social dynamics inside of large companies [such as IBM] resemble that of the open-source community," says Vepstas. "It's just that within large corporations the cooperation and the bickering are hidden from public view. The Linux/390 guys within IBM were stepping on all sorts of land mines internally."

The huge importance of the 390 mainframe within IBM -- both symbolically and strategically -- ensured that the executives with the most knowledge about Betzler's activities kept them quiet. But at just about the same time Betzler got started -- the spring of 1998 -- other groups at IBM were reaching out to the open-source world with open arms.

On the corner of Third Street and Bryant, in the South of Market neighborhood of San Francisco, there is a restaurant known as the Big Tomato. Its real name is Vine e Cucina, but the local clientele are too busy programming or otherwise online-obsessed to be bothered with its actual name, and just refer to it by the unavoidable sign out front.

In 1998, the Big Tomato enjoyed a fortuitous propinquity to one of the world's most thriving physical nodes of Internet culture and business. Countless Web-related start-ups clustered in the buildings nearby. Organic Online, the high-end Web production studio that employed open-source star Brian Behlendorf as its chief technical officer, was just a few feet away. So were the offices of Wired magazine, for which Behlendorf, at the tender age of 19, had brought Wired's online adjunct, HotWired, onto the Net. Behlendorf still hasn't strayed far -- Collab.net, his current startup, is just another long block away.

So when people came to visit Behlendorf in his own neighborhood, there was a good chance that the Big Tomato was where they would end up -- which explains why one spring evening in 1998, he had dinner there with two representatives of IBM, James Barry and Yen-ping Shan.

In retrospect, the meeting was a dramatic turning point, the moment when the old world and new world of computing met to shake hands. At the time, though, unless you were a very close follower of the nascent open-source scene, you might have been excused for wondering what reason Big Blue could have for setting up a powwow with a ponytailed 24-year-old who split his time between Organic Online and rave DJing.

By that summer, Linux-based operating systems had already attracted a huge following, and earlier that spring Netscape had made the dramatic announcement that it would be releasing the source code to its Navigator Web browser. But the traditional corporate world, at least from a managerial standpoint, still didn't seem to know what to make of this hacker frenzy. Software engineers everywhere were already gung-ho, but the suits were a step or two behind.

James Barry and Yen-ping Shan weren't your ordinary IBM suits, however. Barry, the product manager for WebSphere, a set of closely related e-business programs, was a jeans-in-the-office kind of guy, and had been employed by IBM for little more than a year. Shan, IBM's chief architect for e-business tools, came from an engineering background. The two men were complementary halves to the same coin. Barry was a gregarious and jovial 43-year-old who in 1998 already had years of experience in online affairs, dating back to a bulletin board he had operated in Boulder, Colo., in the early '90s. He recalls, "Shan was the technical guy who knew a lot about marketing, while I was the marketing guy who knew a lot about the technology."

Both men were certain of one thing: It was in IBM's interest to support the Apache Web server, a program developed by a loose group of volunteer programmers led by -- or, more accurately, coordinated by -- Brian Behlendorf. But just getting as far as this meeting had required mastering an internecine political process at IBM that defied ordinary mortal comprehension. Engineers at IBM had been fans of Apache since at least 1996, when it was used as the Web server platform underlying IBM's Web-based front end to the Atlanta Summer Olympic Games. But IBM also owned Lotus software, which had its own Web server program: Domino Go. IBM software executives kept squashing engineering's Apache enthusiasm, tracing their mandate all the way back to the CEO, Lou Gerstner. You've got to eat your own dog food; if IBM had a Web server product, it should be pushing that product and using it for its own servers.

The only problem was, practically no one besides IBM itself was using Domino Go, which made it rather unwise to rely on the program as a first step for penetrating other Internet software markets. For months, Barry and Shan had been working to persuade IBM of Apache's strategic advantage.

First, Apache was what people were using. Shortly after Barry had been hired, initially as a consultant to evaluate IBM's "middleware"* offerings, he had lectured IBM managers on the fact that Apache was the most popular Web server program on the Internet -- and the single most widely used piece of software for the hosting of Web sites. Even in the Fortune 500, IBM's home territory, more companies were running their Web sites on Apache than on Domino Go. (Though, to be fair, some of those high-profile corporate sites, such as those belonging to Nike and Levi's, were actually being hosted by Organic Online.)

Second, although Apache dominated the statistics for publicly accessible Web servers, owning more than 50 percent of a hotly contested market, Microsoft's share was also growing steadily. And again, that growth was occurring in the well-heeled market sector that IBM most lusted after. Apache owned the low end of the market, but Microsoft was gunning for where the money was. If IBM wanted to prevent Microsoft from claiming yet another software market, it needed to join forces with Apache.

Third, since so many sites were using Apache, a vast amount of software tools had been created that would work with Apache. And since Apache was both open-source and conformed as closely as possible to all public Internet standards, it was easy to adapt those tools to different software platforms. According to Barry, if IBM came up with a set of software services that worked on top of Domino Go, it took a good deal of code rewriting to get that software to work with either Apache or Microsoft's IIS Web server. By making Domino Go the center of IBM's strategy, it was, in effect, handcuffing itself.

For a year and a half -- much of which, say his friends, was spent in the air traveling from IBM office to IBM office -- Barry pushed the open-source strategic imperative to anyone who would listen. If IBM was interested in fending off Microsoft, if it cared at all about creating the widest possible pool of customers for all the fancy e-business services that IBM wanted to offer its customers, then it must get with the real program -- the open-source program, the Apache program.

There was just one niggling problem, even after Barry and Shan finally won over higher levels of IBM management: IBM wanted Apache, but did Apache want IBM?

Certainly, Brian Behlendorf was cautious. He describes his own state of mind at The Big Tomato that night as "guardedly thrilled." Behlendorf is not by nature a suspicious man, but he was wary. He might still have appeared to be a wet-behind-the-ears Internet hacker, but he knew IBM. His parents had actually met each other while they both worked at IBM -- if anyone had grown up steeped in the culture of the computing industry's most dominant enterprise, it was Behlendorf. IBM had a way of swallowing its collaborators, of overwhelming smaller companies with its phalanxes of sales shock troops and mind-numbing invasions of managers. As a representative of not just the Apache Group, but all of emergent Net culture, Behlendorf couldn't help being restrained in the face of outreach from one of the world's biggest corporations.

Behlendorf did not "run" Apache. No one did. Instead, he helped coordinate the efforts of a group of programmers, all of whom for one reason or another needed a good Web server program to help them carry out their day job or hobby, to improve the existing publicly available Web server technology. The original base of code came from the University of Illinois, developed by the same team of programmers that had created Mosaic.

But those programmers had moved on en masse to Netscape, which -- at the time of Apache's emergence in the mid-'90s -- was developing, slowly, its own high-priced, proprietary Web server. Meanwhile, as the Web expanded at phenomenal speed, there was a drastic need for improvements to the existing freely available Web server code. All across the Net, webmasters were hacking their own patches* to the code, quick fixes that would help them respond to their daily needs. Finally, a group of these programmers got together, collated all the patches and created "a patchy server" -- Apache.

Behlendorf's influence came through his calming presence on Apache-related mailing lists, as the systems administrator for the Apache Web site and as the maintainer of the Apache "source tree"* -- the code base for Apache to which the core group of some 20 programmers had access. His interest always was, and still is, to devise technological means of enhancing collaboration. Lacking the ideology of a Stallman, or the programming skills of a Linus Torvalds (he is quick to say of himself, with a self-deprecating smile, that "I am *not* a very good programmer"), his motivation has always been to create things that work, that get the job done.

Apache got the job done. It wasn't necessarily the best Web server, the fastest, the most powerful or the most secure. But it was still the most widely used, in large part because it handled, simply and effectively (and freely), the tasks that most people needed handling.

Of course, IBM had a different set of motivations -- generating revenue being chief among them. So when James Barry told Brian Behlendorf that IBM wanted to use Apache as part of its own family of e-business products, and that it wanted to start contributing to the Apache project, Behlendorf's first reaction, recalls Barry, was defensive. The Apache group did not want a giant corporation to come in suddenly and take over. Yen-ping Shan hastened to sooth him.

"I told him," recalls Shan, "that we are going to play by your rules, because we believe that your structure and practice actually works."

Shan added that IBM's support could only strengthen Apache. "There are multiple ways IBM is going to help," Shan remembers saying, "not just technologically but as an endorsement that will solidify Apache in the IT [information technology] world. IBM will announce enterprise-level support."

Fine. If IBM was going to play by Apache's rules, then that's what it would have to do to win the Apache group's support. To do that would require something a bit more substantive than taking Behlendorf out to dinner.

It would require code.

IBM had to become a contributor. And it would have to prove itself the way any Apache contributor did, by submitting patches that were accepted by the core as valuable improvements to the Apache code base. And it had to do so in a sensitive way. Behlendorf did not want to see hundreds of patches appearing from scores of IBM engineers. He didn't want IBM to suddenly dominate the open discourse of existing Apache programmers. If IBM wanted one of its employees to become a member of the Apache core (which Barry and Shan's boss had set forth as an essential requirement before greenlighting their mission to Apache), then that employee would have to earn his or her way there like anybody else, by merit, through the quality of his or her hacking.

Barry and Shan agreed. It wouldn't be easy. The very concept of an IBM employee contributing code to a project that wasn't owned by IBM raised hackles on legions of IBM lawyers. Traditional software industry policy held that an employer owned everything an employee did, even to the extent of idle thoughts the employee might linger over while showering in the comfort of home. There was also the sticky question of patents -- what if a contribution of code from an IBM engineer included concepts or techniques that had been patented by IBM -- what would happen to those patents if they became part of the public domain? What about liability issues?

Barry recalls with a pained grimace the months of meetings that had to be undergone in order to work out such issues. But, credit must be given to IBM's legal team. The issues were worked out. A single programmer, Bill Stoddard, was given the job of being the connection between IBM and Apache -- if any of IBM's programmers came up with a patch, Stoddard reviewed it first, and then he personally submitted to the Apache group.

And in the Apache group, good code always won the day. Stoddard's contributions were accepted. IBM was accepted. IBM endorsed Apache, and gave open source an entree into the land of the suits. And Apache endorsed IBM, proving that hackers could work with the biz guys. The announcement of the agreement generated some 1,000 media stories -- which, more than any other fact, Barry recalls with a rueful grin, sealed the deal for upper management. That kind of positive press was by definition a successful strategic move.

Today, all three representatives at that meeting have moved on to new jobs. Yen-ping Shan is the chief technical officer at the largest payments-processing firm in the United States. Brian Behlendorf is CTO of Collab.net. And James Barry works a desk about 30 feet from Behlendorf. He is now Collab.net's vice president of strategic development.

The summer and fall of 1998 saw one open-source-related announcement after another from Big Blue. IBM joined the Apache Project in June. In mid-July, two researchers named David Shields and Philippe Charles announced the release of a version of their JIKES Java compiler for Linux. By September they had open-sourced JIKES. At the same time, a Toronto researcher named Gary Valentin had completed his own skunk works project, porting IBM's db2 database software to Linux. Meanwhile, that spring, Boas Betzler and his Burblingen cohorts had begun work on their 390 port.

But there was still no companywide strategy. In various corners of the IBM empire, individual researchers like Shields or strategists like Barry (who met and became friends through an open-source mailing list started by Barry for IBM employees) were doing their own thing, but as a company, IBM was hardly united. Shields does recall one key meeting of the IBM Academy of Technology, a grouping of 300 of IBM's most distinguished scientists in October 1998, at which both he and Barry spoke, as crucial. The Academy declared Linux to be an "earthquake" (as it had earlier declared Unix and the Internet) and petitioned Lou Gerstner to review their findings.

But even though Gerstner formed a task force to study Linux, the struggle over policy at lower levels still raged without cohesion. Barry recalls how plans to write a version of WebSphere 3.0 for Linux were spiked by an IBM executive who gave a speech at IBM's research lab in Raleigh, North Carolina, declaring that Linux was "going nowhere." That executive, he notes now, without hiding his satisfaction, is currently in charge of an IBM division devoted to Linux.

The task force recommendations, says Barry, were watered down and "milquetoasted." IBM seemed lost at sea.

Then came the morning of Dec. 14, 1998.

That day, John Markoff, the lead technology reporter for the New York Times, wrote a short piece entitled "Sharing Software, IBM to Release Mail Program Blueprint." In it, Markoff detailed how IBM was planning to release a mail program called Secure Mailer, developed by a programmer named Wietse Venema, as open source. What the article didn't say was that the program had been something that Venema had created before joining IBM, that it had *always* been open source, and that IBM was only now acknowledging that Venema could keep working on it as an open-source project.

But that didn't matter. All that counted was that IBM CEO Lou Gerstner read the article, and, according to legend, immediately became apoplectic. As far as he knew, IBM already had a mail program -- it was part of the Lotus Notes package. And if IBM was endorsing open-source software as a worthwhile strategy, then Gerstner wanted to know about it.

James Barry says that Gerstner didn't care one way or another about open source as a software methodology. Barry says that Gerstner frequently liked to note, "I am not a technologist." What he cared about was strategy. Did IBM have an open-source strategy? And if so, what was it?

Gerstner started making phone calls. First he called his chief of software, who called his subordinate, who in turn called his. The conference call kept expanding, until it made its way down to the research director who managed Venema. By the end of day, Gerstner had his answer. There was no clear strategy. Or at least there hadn't been up to that point.

"There was that one morning in December of 1998, and by that afternoon the open-source strategy had jumped into the runway," says Dan Frye, IBM's program director for open source and Linux. "We talked to everyone in the industry. The answer we came back with was that open source was good for us."

As a result, Linux got the green light. The skunks could come out of the woodwork.

Of course, it still wasn't easy.

"Internally, the battles were amazing, and you could understand why," says Jeff Nick, chief architect for the System 390. "A lot of the [IBM] technical community was very incredulous about this. You grow up in an OS/390 [the operating system designed for the 390] community, and there is great passion and pride in the heritage of OS/390 and the integration of its capabilities into the hardware platform. To suggest that there is a value proposition for running an open-source, not controlled, Unix platform on our hardware, and to propose that Unix applications might be better suited for running on Linux on the 390, than on OS/390. I was almost seen in my own technical community, particularly in the system 390 design council, as antichrist. There were multiple painful meetings with my technical peers across IBM on the OS/390 platform. They were saying, you must be out of your mind, why would you want to do this, we need to protect the OS/390 environment."

"It was a huge risk," says Nick. "And the reason we were ultimately successful was that we could show that by supporting middle-tier Unix applications that are collaborative in a distributed environment with the 390 back end and by running that entire heterogeneous workload on our platform, we would actually in the end be providing a bigger platform for our customers than we would if we forced everything to be on OS/390. There's a huge risk in that statement, but we are banking on the power of open source."

Huh? Just what exactly is Nick trying to communicate here, in that mix of techno-jargon?

In essence, pretty much the same thing that James Barry and Yen-ping Shan were saying when they pushed Apache. There is a whole world of people using Linux-based operating systems and the vast ecology of programs that run on those operating systems right now. An enormous amount of work is being done using a set of tools that have a Unix heritage and are currently at home on Linux-based systems.

The Linux generation is in some ways the heart and soul of the Net, and its numbers, according to Nick, are surging. An entire generation of programmers has adopted Linux. It doesn't matter whether they are doing this because they hate Microsoft, or think Linux-based systems are technically superior, or just like to hack on free software. The fact is, it's happening. Market share for Linux-based operating systems -- and mind share for Linux among developers -- is continuing to rise.

By ensuring that Linux will run on the 390, IBM would ensure that its mainframes would be an attractive environment for all those programmers and system administrators to work in. A bank could use its mainframe to handle its massive data-processing needs, and at the same time allow its Linux-skilled programmers to do whatever they needed to do -- in particular, to make use of all the middle-tier software applications that have been developed to get things done in an open, Internet environment. By expanding the possibilities, IBM would be able to expand its own market penetration.

The break with tradition represented by IBM's decision to open up the mainframe to non-IBM software is hard to overstate. For decades, IBM banked on selling customers the entire "vertical" package. From the hardware to the software to the support, it was all Big Blue, all the way. But now, by acknowledging that it made strategic sense to -- paraphrasing Chairman Mao -- let a hundred software programs bloom on the mainframe, IBM was signaling that it knew it could no longer call the shots in the mainframe marketplace.

Again, it just doesn't matter, from this perspective, whether Linux-based operating systems might actually be technically inferior to their Windows NT or Sun Solaris competitors in certain aspects. I asked Nick why, if it made so much sense to try and take advantage of all those people with Linux skills, wouldn't it also be prudent to attempt the same with NT, and gain access to that huge world as well?

"It would be great if you could do that," says Nick. "But the difference is that because Linux is open source, which allows it to be worked on by a large collaborative set of developers, it has also been built to be platform agnostic. It's got what we call horizontal layering of function, so you can easily port the OS to multiple machine architectures and platforms. This is not true of NT and this is not true of most Unixes -- where each operating system has grown up tied to its machine architecture. "

In other words, Linux, even if it may have started out as a hack to run Unix on a cheap Intel processor, has since evolved into the ultimate protean operating system. Over the years, its functions have been streamlined and compartmentalized to the point where it has become relatively easy to adapt it to different systems.

As such, Linux-based operating systems (as well as BSD operating systems, although they represent a much smaller percentage of the current OS marketplace) are the true heirs of what one programmer once immortalized as the "worse is better" paradigm.

In the 1980s, a programmer named Dick Gabriel wrote a paper about the programming language C++ and the operating system Unix called "Worse is Better." His argument was that simple systems that get most of the job done are better at surviving, over the long run, than complex systems designed to do everything perfectly. Complex systems are hard to adapt to new situations, and can break down easily. Simple systems can be fixed quickly, and mutate even faster.

Today, Gabriel is the main open-source evangelist at Sun Microsystems, and C++ and Unix are the building blocks out of which the Internet has been constructed. Linux is just the newest all-purpose building material.

There's a "virtuous cycle" here that feeds voraciously on itself. As Linux is ported to an ever-increasing number of hardware platforms, an ever-increasing number of programmers gain the opportunity to work on code that benefits everyone. Which in turn makes Linux-based systems even more attractive.

And that's the business case for open source. At first listen, "worse is better" sounds like Orwellian doublespeak, a phrase designed more to confuse than enlighten. But in practice, "worse is better" is an actual evolutionary success strategy -- and nothing exemplifies its principles better than open source.

------------

**Read Chapter One of the Free Software Project**

**Join the discussion on this chapter**

**Next installment:** *The Sun also rises on open-source software -- how Sun Microsystems, with a little help from Collab.net, is learning from its mistakes and joining the world of open source.*

------------

**About the writer**
Andrew Leonard is a senior writer for Salon Technology and author of Salon's Free Software Project, an online book-in-progress exploring the history and culture of the free software movement.

**Sound Off**
Send us a Letter to the Editor

GO TO: Salon.com >> Technology