

C H A P T E R

1

The Penguin on Top

February, 2001, New York City

You've got all kinds on the floor of LinuxWorld Expo, the semi-annual convention (one per coast) where the Linux faithful (and not so faithful) meet. Start walking from one end of the exhibit hall to the other, and you'll see the giant penguins hovering over post-adolescents wearing devil horns, corporate types from IBM, teenage boys hunched over laptops, mind-readers, hired models, smirking tech journalists, glad-handing executives, assorted geeks, and the occasional Linux evangelist preaching to his public.

This year's convention is particularly interesting. Beyond the sideshow-like atmosphere of some of the exhibits—not to mention the ill-matched menagerie of suits, nerds, and ad-men—many of us tech journalists sense that we should pursue a final interview with some of the exhibitors, because it may well be the end of the road for the smaller companies. This notion is fueled by rumors of layoffs in some major Linux companies; by an ever-dwindling number of outfits hawking shrink-wrapped distributions on the strength of installation support; and by the

largest booths in the building, which are owned by truly giant companies, such as IBM and Intel—indicating that Linux is a market that’s been entered, for good or ill, by the people who define computing.

“Look around,” one writer says, “half of these people will be gone next year.” An older editor makes a crack about the kids being run out of the sandbox. A look over at the Slashdot booth, replete with beanbag chairs (these are *de rigueur* for Linux gatherings,

implying the carefree spirit and sturdy backs of youth), Nerf guns, and a Playstation, shows a collection of middle-aged men dressed in corporate-weekender outfits, eyeing the whole thing with mixed puzzlement and condescension. Their hands dip into the pockets of their Polar-fleece vests to produce business cards as the

perimeter around the temporary geek-chic habitat holds, itself produced by the corporate largesse of Slashdot’s owners.

I take a moment to sit in a beanbag chair alongside a senior executive from a company new to the Linux world. He’s building software that makes Linux easier to deal with for housewives, secretaries, and “the end user,” a mythical creature widely assumed by true geeks to be the tragic by-product of a lobotomy and an unfortunate youth spent playing in the sun with other children. The executive mentions “the hacker ethic,” referring to the loose set of characteristics so many of Linux’s pioneers seemed to possess: curiosity, constructive anti-authoritarianism, mutualism, independence, and mistrust of the ready solution or typical answer.

“Your company is new to all this,” I say. “And I mean no disrespect, but it’s hardly a collection of hackers. You talk about the hacker ethic, but it seems like your company won’t really be

“Look around,” one writer says, “half of these people will be gone next year.”

a success until hackers are a much smaller proportion of the Linux user base.”

The executive, perched in his beanbag chair, is quiet for a moment, and then picks his words with care. “It’s still early. The hackers are vital to us, creatively vital people...but, well, Linux is bigger than any group.” Five minutes later, he’s bemoaning the woeful state of quality control present in open-source software, and the slovenly adherence to deadlines observed by the average hacker.

He wants the kids out of the sandbox.

The same day, though, I walk over to a company trying to make its mark with a version of Linux that runs on handheld computers. The people at the booth are excited, their body language is almost frantic. The words of one of the developers I speak to come out in a flood. He excitedly demonstrates a handheld computer performing instant chat over a wireless connection to the Internet. He shows how even the most mundane data can be transformed with his software to make sense in a variety of contexts from handheld computer to desktop machine to processes that don’t ever show themselves to users, but communicate amongst each other. There’s a high cool factor to what he’s preaching, but there’s also the point to which he keeps returning: All of it happens with open-source software.

“This is all with open standards,” he exclaims. “With open-source software. And it’s happening with all the tools we’ve been building from free software for years. People have been building the foundations to make information take on its own life for years without knowing it, and now we’re bringing it together.”

Part of it is, to anyone who’s been around computing since the first days of the Internet creeping into the awareness of self-annointed Information Age prophets, old hat. Fads come and go. At each conference, we’re promised a golden age of intelligent software, or “smarter information,” or some other buzzword.




Some maintain that open-source software is itself a fad preached by businesses looking to sound hip even as they figure out ways to share nothing while harnessing the creativity of youth eager to sign up with a revolution or join a movement. But the fact of the matter is, this guy is right. For years and years, people have been building an infrastructure that was surely meant for something besides pushing banner ads, pornography, and corporate brochure sites. Tools have been crafted, sometimes because they had to be to progress any further, sometimes because it was just fun.

Standards have been defended against attempts to close them because a community of Internet users needed those standards kept open to keep their tools working, and the lines of communication open.

The result? Small companies are out to redefine everything with big ideas and the enthusiasm of revolutionaries. Linux was never the meaning of the exercise to them, but Linux is free and open—the perfect base from which they can explore their own ideas without having to reinvent or build from scratch.

It all adds up to paradox after a few hours: Internet visionaries who see Linux as just another tool; industry veterans who predict the demise of more than a handful of Linux companies as the stock market euphoria fades and doomed revenue models they defended prove unviable; graybeards who have come forth from the most hidebound backgrounds to embrace an operating system that was built by hackers but is now in the hands of everybody regardless of cultural allegiance. All the horses seem to be pulling in a direction that indicates the end of Linux as a phenomenon. And yet, it's precisely because of these things that, well, the penguin is on top.



Small companies are out to redefine everything with big ideas and the enthusiasm of revolutionaries.

TALKIN' 'BOUT A REVOLUTION

Linux poked its head into the world barely ten years ago, not much more than an interesting diversion for a hobbyist, and promising little more than the novelty of booting something very UNIX-like on PC hardware. During the past ten years, however, Linux has gone from toy to contender, putting the fear into more than a few companies as it has found its place as a “glue” operating system. Maybe it didn't perform some tasks that well, but it excelled when it came to doing the sort of day-to-day grunt work network and system administrators rely on to keep things moving.

Alternately lauded and reviled, sometimes by the same people, Linux's progress in the server room and the public imagination has been undeniable. The phenomenon crescendoed in the summer and fall of 1999, when several Linux companies held initial public offerings that set new records as their share prices rose to twenty and thirty times the initial offering price. That explosion died, though, and the very same companies struggle now, still moving toward profitability, but buoyed by buzz no longer.

How is this indicative of a revolution won? If the corporate guys are talking about nudging out the hackers who made Linux happen in the first place even as the public euphoria over the phenomenon fades, where's the victory?

The answer lies in the ubiquity of Linux in 2001.

When Linus Torvalds gave the gift of Linux to the world, he did so (after a false start) under terms that left Linux open to all takers, for good or ill. An implicit understanding of the movement from which Linux derives its license, the document which

sets the terms and conditions under which Linux may be distributed, is that software must remain open to modification. That way, if a programmer misses the mark where another's needs are concerned, the next person down the line may do what's necessary to build on the existing work and make it better.

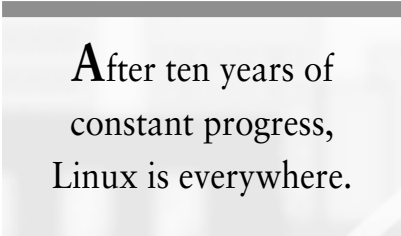
And Linux has missed the mark plenty of times over the years, at first not even able to be booted on its own, then lacking so much as a collection

of reliable networking tools. With time, however, each of these deficits has been corrected—the result rolled into the ever-growing, ever-diversifying collection of software and tools that is

Alternately lauded and reviled, sometimes by the same people, Linux's progress in the server room and the public imagination has been undeniable.

Linux, slowly making its way into server rooms and back offices, Web servers and desktops, until we arrive at today, with an operating system that's taken seriously by the likes of IBM and Microsoft (one hoping to profit, the other hoping it will go away before more damage is done).

After ten years of constant progress, Linux is everywhere. It's everywhere in so many forms and because of the efforts of so many people with so many different agendas that it can't go away, because it's not a single thing by a long shot.



After ten years of
constant progress,
Linux is everywhere.

As I stand in that convention hall, surrounded by the booths of companies soon to disappear, enclosed by executives who are grateful for all the code they can work with but less so for the hackers who wrote that code, overshadowed by IBM's gigantic booth, I realize something important. Even if the revolution is over, it has left its mark on the computing landscape that won't be removed no matter what sort of marketing hype is brought to bear on it, and no matter how dismally the companies who sought to sell it perform.

The sound and fury of the past few years on the part of Linux evangelists mighty and small came together to buy Linux a place at the table. Whether the hackers carry it forward, or some computing giant ends up driving it, Linux is a fact.

IS THERE A GENERAL PURPOSE, TIME-SHARING SYSTEM IN THE HOUSE?: THE REALLY QUICK AND DIRTY STORY OF UNIX

It's impossible to understand Linux at all without knowing where it came from, and that's a story that goes back over 30 years to Bell Laboratories, where UNIX was born.

Two computer scientists named Ken Thompson and Dennis Ritchie were working on an operating system called Multics (*Multiplexed Information and Computing Service*); an attempt to create an operating system for large computers with the means to provide access to many users at the same time. Multics, thanks to its resource hungriness, had earned the disapproval of management, which promptly, in the manner of management everywhere, pulled the plug on the project, leaving the gentlemen without a sanctioned operating system project.

Thompson was eventually granted the use of a DEC PDP-7, and in best computer-nerd fashion, set about to make it a better game machine. That in turn led to the first UNIX kernel. For his part, Ritchie had developed a computer language, which he called “C,” for Thompson’s new operating system. C is important for the simple reason that it was one of the first proofs of the notion that a programming language can be “cross platform.” That is, if you use C to come up with a really good version of “Space Wars” on one machine, you’ll be able to run it on a different machine with a different operating system with little hassle.



This is called *portability*, and computer historians agree that there wasn't much of it leading into the 1970s.

So with C in hand, Thompson and Ritchie were tasked with producing an office-automation system for Bell, and they were given better hardware with which to do it—a DEC PDP-11 to be exact. Thanks to C and the wonder of portability, the two rewrote much of the embryonic UNIX and ported it to the new hardware.

Portability allowed UNIX to easily move from its earliest host to its new home, and it also eased the spread of UNIX among computer enthusiasts (who weren't, at the time, quite the same people as they are today, owing to the scarcity of computers outside research institutions and universities). Bell Labs, thanks to the antitrust woes of AT&T, was fairly friendly about the distribution of the UNIX source code for several years, releasing at least one version to universities free of charge, and another for around \$100. Businesses and the government had to pay tens of thousands of dollars. For a while, the source code for UNIX flourished until AT&T made moves to halt it, citing its ownership of the proprietary source code.

UNIX, however, had caught on.

Books have been written about what made UNIX so popular, and we could spend the rest of this volume mucking about with the gory details. If we did, however, this book would stop being a “connoisseur's guide to open source,” and start being an anatomical text. We don't want to go there. Some key items should be listed, though, because they exist in one form or another in Linux as we know it today. For every baggy-pants-wearing kid

This is called
portability, and
computer historians
agree that there wasn't
much of it leading into
the 1970s.

on a skateboard who runs Linux because it's "kewl," there are plenty of people who first picked it up because it gave them a UNIX they could play with in the privacy of their own home, and that was something they'd wanted for a very long time.

So...the reasons for all that enthusiasm:

- **Portability.** Thanks to C, you could move UNIX from one machine to another without having to rewrite all your favorite programs from scratch.
- **Modularity.** It provided an abundance of very functional, single-purpose tools that were easy on memory and resources, but could be combined to provide big results in a number of ways.
- **Flexibility.** It was flexible without requiring a ton of in-depth knowledge. You could leverage your command of the individual components to produce bigger results through scripting and piping, without needing to come up with a whole new program each time your needs changed.

Portability

UNIX places an emphasis on values that seemingly represent diametric opposites of what you see in other operating systems. For

UNIX places an emphasis on values that seemingly represent diametric opposites of what you see in other operating systems.

instance, portability is a key value that UNIX enthusiasts share. It's because of this desire to be able to move a common set of tools from one computer to the next that Linux (and its cousins) now run on a staggering array of computing hardware: everything from garden-variety PCs to iMacs to handheld computers to massive mainframe computers.

Modularity

Think about a word-processing program and everything it does. It allows text to be formatted, printed, previewed, spell-checked, and generally mangled—and it does it all in a single program that gets bigger and slower as time goes by and more features get chunked in. Word processors are not modular. Now consider a collection of small programs that, on their own, are the functional equivalent of bees in a hive, with a single purpose apiece.

You might, for instance, have a single program for doing nothing more than entering text. It won't check your spelling, or offer a way to make a pretty printout of what you're working on. Once you're done with that program, however, you can then run your file through a spell-checking program that, although it may not do much else, really knows how to spell check. Done with the spell checker? Move the file on to a formatting program that reads your formatting codes and converts the text file into something a printer can understand. From there, you hand things off to a program that handles printing quietly and efficiently with minimal overhead.

To the “modern” way of thinking, this is clearly insane. It implies complexity, because each of these programs must be run on their own—and we all know how bad complex things are. Why bother with a bunch of small, efficient programs when there's a one-stop solution?

We're not going to be OS fascists and suggest that this is a bad question to ask. In some cases, especially considering how

Now consider a collection of small programs that, on their own, are the functional equivalent of bees in a hive, with a single purpose apiece.

inexpensive good computer hardware is these days, it's entirely appropriate to just stick with that big, memory-hungry word processor with more toolbars than there are lost socks in the world and plow ahead.

On the other hand, when you consider that there are hundreds more programs just like the ones I've described, which can do all sorts of other things really, really well in all sorts of combinations, it makes you think. If you're in an environment where processing power is shared, those small programs doing their thing and then quietly exiting end up saving a lot more processing time and memory than a single, huge program.

Flexibility

"Flexibility" in the UNIX world is a simple concept once we take all those single-purpose tools and marry them together. Much the same way a necklace made of many tiny links will appear more supple than one made of big, iron rings, UNIX's "small tools" approach means that there are more small, meaningful relationships a user or programmer can establish between all those tools, and there are plenty of ways to do the same thing, each as "correct" as the next, even if some are more appropriate in certain contexts than others.

The beauty of the UNIX way of doing things is that your knowledge of a fairly small set of all the available tools can be leveraged by compounding their effects and making more tools in all sorts of combinations. And because these compound tools are made of tiny, replacable parts, there's a seemingly infinite array of possibilities when it comes to accomplishing a task. A few features of the UNIX approach confer this flexibility nicely:

Piping

A practical corollary to the modularity of single-task programs within the UNIX way is that they should also act as filters through

which you can “pipe” information. Consider, for instance, one of the more tedious tasks of daily corporate life: taking credit for someone else’s work.

In this scenario, you, a simple corporate drone with an overpriced apartment and an expensive monthly car payment, have been toiling in the shadow of your neighbor, the ever-industrious John J. Spurworthy, who has spent the last year working on a project you secretly covet. You’ve got a chance to pass all of his work off as your own...but wait! The rotter has made sure his name is in every file related to that project.

With that *other* word processor (we’re not naming names), you could open each file and hunt down his name. Not too bad if he just put it at the top of each file, a trivial task for a moderately experienced Word-pro warrior if it’s liberally sprinkled throughout, but a colossal pain if it’s not only liberally sprinkled through one file, but all 183 he was working on.

Thanks to your ability to pipe commands with UNIX, help is on the way. You just visit the directory where the mightily gifted and devoted servant to your ungrateful corporate overlords keeps his stuff and type:

```
cat * | sed s/John\ Spurworthy/Dick\ Phillips/g | lpr
```

And out comes his work, having been opened by the cat command (which does one thing very well: shows the contents of files), filtered by the sed command (which does one thing very well: slices and dices text files), and then handed off to the lpr command (which sends files off to the printer), pristine in all its creative glory. Except, of course for the niggling detail that his

Consider, for instance,
one of the more
tedious tasks of daily
corporate life: taking
credit for someone
else’s work.

name is no longer anywhere to be found in the printout, which means you are, um, one gravy-sucking corporate stud-muffin.

Scripting

So you've got small, single-purpose programs that do their one thing well. Added to that, they can talk to each other, making your computer more of a hive of determined and capable bees than a collection of really, really expensive pink elephants with wet-bars tied to their backs. Add the element of easy scripting to the mix, and you've really got something.

Scripting is an easy concept to grasp: Rather than typing a bunch of commands in over and over again, you can write a very simple program (a *script*) that does the typing for you. This capability may remind DOS fans of batch files, but because UNIX was so oriented to the power users of computing in its early days, it developed much more powerful scripting capabilities and, of course, always had many more of those powerful, single-purpose programs to use when it came time to write the scripts they used.

For many tasks, all of those little programs—combined with their capability to filter information before sending it to the next

Scripting can combine all the programs you'd use one at a time into a single script that you can use and reuse.

program in a pipe, and further combined with their capability to string together even longer sequences of commands and pipes into a script—mean that there's less need to do what we think of as “programming” for common tasks. You don't need to know C, for instance, to come up with a nifty way to put all of Mr. Spurworthy's work in a secret direc-

tory, introduce a few embarrassing typos into a few of his files, and mail your boss mentioning that Spurworthy seems to be

having a hard time lately, what with all his bad spelling. Scripting can combine all the programs you'd use one at a time into a single script that you can use and reuse. Better yet, your scripts can pipe data amongst each other if they're properly written, which means you can come up with chunks of commands that do much, much more than a single program, in a dizzying number of combinations. This scriptability and combinability gives life to another UNIX value: code reusability.

Using Text-Based Configuration Files

Another element of your typical UNIX is a reliance on text-based configuration files. People roll their eyes and think of the earlier days of Windows when everything was configured in multitudes of INI files. Unroll them, and think about the hell you go through with a single, gigantic registry file.

Thanks to your tiny filtering, inter-operating, scriptable tools, configuring UNIX and all its applications becomes a pretty simple proposition. You can pipe your configuration files through a series of commands that make the changes you need. It may seem trivial for changing only one setting, but if you're a network administrator who suddenly finds that a computer further up the line has changed its name or address and that many of your programs count on knowing where that computer is, the beauty of all those text files and all those little programs is evident.

IS THAT A COMPLETELY FREE RE-IMPLEMENTATION OF UNIX IN YOUR POCKET?

So UNIX was everywhere and people loved it. It was the de facto computing standard for many universities, and many a nerd felt the earth move courtesy of pipes, filters, and scripts. In fact, one man's love for UNIX, combined with his hatred of being kept

from modifying computer software when it didn't work as he required, led to a rare thing indeed: a social movement driven by "computer people" that spawned a new way of thinking about licensing software. This new way of thinking seemingly subverted peoples' conceptions of intellectual property as a vehicle for restriction. Along the way, he also built a foundation for Linux to which it may owe everything.

There's certainly no arguing that without his work, Linux would have had a different complexion.

Richard M. Stallman, more commonly referred to as "RMS", was a member of MIT's Artificial Intelligence Lab. Among hackers, the lab is legendary for the role it played in the formative years of computing, and RMS is part of that lab's history.

THE REST OF THE STORY...

To painfully understate the next 20 years of computing history, UNIX went on to become the workhorse of the Internet and a mainstay in corporations and universities. UNIX was the glue that held the Internet together in many ways. Businesses relied on UNIX; it was the operating system of choice for many.

Unfortunately, because of its popularity and the number of ways it was re-created by many different companies, UNIX became a victim of its own success. It's a story for another day, really, but UNIX underwent a period of fragmentation. Because of conflicting attempts to establish standards, leveraging one of its traditional benefits, portability, became so difficult that the UNIX market fell into chaos. In many ways, this gave Microsoft an opening into the server market to which it continues to cling to this day.

Now is as good a time as any to delve into the whole “hacker” issue, briefly, since so many Linux enthusiasts describe themselves as hackers and so many people without cathode ray tans know (as all good law abiding citizens do) that a hacker is nothing more than a computer vandal, which makes the “real hackers” angry.

The classical definition of “hacker” varies depending on the source. The safest bet, though, seems to read something like this:

Hackers are people possessed of a love for things that aren’t known. They’re curious, and interested in the way things work, and often disinterested in formalism when it comes to figuring things out. Hackers are not tied to computers. Hackers aren’t universally disinterested in rules and order. On the other hand, their curiosity is a driving value, and it takes high priority.

There are, of course, a lot of self-styled hackers wandering around these days, thanks in part to the popularity of Linux and its roots among software hackers. We once had the distinct pleasure of reading a message by somebody who had claimed to “hack” a word processor to display page previews correctly by setting the “zoom” level to 10%. What it lacked in ingenuity, one could argue it made up for in elegance...two mouse clicks et voilà! In fact, it’s fashionable to be a hacker and there’s woefully little in the way of peer review or certification to keep people from claiming the title. But the main thing to keep in mind is that “hacker” doesn’t necessarily mean “vandal,” and people claiming the title are frequently law abiding enough folk, and certainly not out to get your credit cards.

Among hackers, the lab is legendary for the role it played in the formative years of computing, and RMS is part of that lab’s history.

MIT's AI Lab was a gathering place for computer hackers of the benign sort and RMS was one of them. The culture of the Lab, by all popular accounts, was one of extreme openness for many years. Sharing one's software was the norm among the members of the Lab, as they helped each other solve problems or simply make their computers do, well, cool stuff. RMS relates his own sense of that community in an essay entitled "The GNU Project":

Hackers are people possessed of a love for things that aren't known.

"We did not call our software 'free software,' because that term did not yet exist; but that is what it was. Whenever people from another university or a company wanted to port and use a program, we gladly let them. If you saw someone using an unfamiliar and interesting program, you could always ask to see the source code, so that you could read it, change it, or cannibalize parts of it to make a new program."

Over the years, though, the AI Lab's open spirit began to decline, RMS himself attributing this to a number of things, including the departure of many of its members to private interests. In addition, the ever-growing computer industry was seeing to it that software stayed a proprietary, closed body of work. To a hacker, curious about the workings of things, this is a burden. To someone who believed that there was a moral imperative to share information for mutual betterment as RMS did (and does), it was intolerable.

So RMS found himself without the community of hackers he'd thrived in, and faced with a larger industry that had identified proprietary secrets as a key ingredient to ongoing growth. It wasn't the world he wanted, and he realized that:

“So I looked for a way that a programmer could do something for the good. I asked myself, was there a program or programs that I could write, so as to make a community possible once again? The answer was clear: what was needed first was an operating system. That is the crucial software for starting to use a computer. With an operating system, you can do many things; without one, you cannot run the computer at all. With a free operating system, we could again have a community of cooperating hackers—and invite anyone to join. And anyone would be able to use a computer without starting out by conspiring to deprive his or her friends.

As an operating system developer, I had the right skills for this job. So even though I could not take success for granted, I realized that I was elected to do the job. I chose to make the system compatible with UNIX so that it would be portable, and so that UNIX users could easily switch to it. The name GNU was chosen following a hacker tradition, as a recursive acronym for ‘GNU’s Not Unix.’

An operating system does not mean just a kernel, barely enough to run other programs. In the 1970s, every operating system worthy of the name included command processors, assemblers, compilers, interpreters, debuggers, text editors, mailers, and much more. ITS had them, Multics had them, VMS had them, and UNIX had them. The GNU operating system would include them too.”

So RMS had his mission: the creation of a free UNIX-like operating system with which he could rally the previously unnamed

“free software” community once again, and restore the openness he’d enjoyed in his time at the AI Lab. He named his project “GNU,” a recursive acronym that stands for ‘GNU’s Not Unix,’” which is not only in keeping with the fondness many hackers have for things like recursive acronyms, but the litigious nature of the computing industry of the time, which would have compelled AT&T to land on the nascent project like a ton of bricks.

IS THAT A RADICAL INVERSION OF OUR UNDERSTANDING OF COPYRIGHT LAW IN YOUR POCKET?

RMS’ formula for “free software” is easy enough to follow:

- You have the freedom to run the program, for any purpose.
- You have the freedom to modify the program to suit your needs. (To make this freedom effective in practice, you must have access to the source code, since making changes in a program without having the source code is exceedingly difficult.)
- You have the freedom to redistribute copies, either gratis or for a fee.
- You have the freedom to distribute modified versions of the program, so that the community can benefit from your improvements.

Don’t ever try to keep other people from getting at your improvements to the source code.

These principles are elaborated on in the GNU General Public License (known widely as “the GPL”). The GPL is designed to guarantee that once software is made “free” under the above definition, it stays that way. There are lengthy and bloody brawls over licensing esoterica in the computing community, but the gist of the GPL is simple enough to express here, until you can get to the Appendix:

Here's the software. Here's the source code. Do what you want with it. If you improve it, make sure you include all the source code to your improvements and pass it along. Don't ever try to keep other people from getting at your improvements to the source code. We can tell you to do this, because we wrote this software and these are the terms under which we're willing to let you have it.

That's a longish way of getting around to the intent of the whole exercise, which might read, in a less litigious society, more like: "Be excellent to each other."

RUNNING WITH THE DEVIL: A BRIEF DETOUR TO THE OTHER FREE RE-IMPLEMENTATION OF UNIX

RMS wasn't the only person with a thing for UNIX.

During the '70s, the University of California at Berkeley was developing their own variant, based on source code licensed from AT&T known as "BSD." Their own version was very popular...to the point that lawsuits ensued and they embarked on creating a version of UNIX that was "unencumbered" by AT&T's source code.

As one might imagine, the process of stripping all the proprietary source out of an entire operating system is a difficult undertaking, but by the time the legal dust settled and the job was done, Berkeley had given 4.4BSD-Lite to the world. From that code we have a collection of free Unixes in common use today: FreeBSD, OpenBSD, and NetBSD.

When Linux and BSD fans come around each other, there are several key differences that pop to the top once they decide to quit mincing words and pull out the brass knuckles.

Where GNU, the eventual underpinning of Linux as we'll see shortly, was built around the premise of freeing software and restoring the hacker culture to its former heights of openness, BSD was built on the notion of building a better UNIX than UNIX.

Where GNU software requires sharing of source code and redistribution of improvements, the BSD license simply says “take this and do what you like with it, just make sure we’re given prominent credit. No need to give back the source.”

A lot of snarling goes on over these distinctions, not to mention the occasional flare-up when someone releases a new set of test results showing that some version of BSD is much faster than Linux. The UNIX “purists” in the BSD camp also like to point out that their variant is descended from the mother source itself and that Linux is a mere “imitation.”

These are fine points to make, and true for what they’re worth. The vitriol you detect between the camps sometimes comes from the fact that despite their “purity,” occasional technical superiority, and easier-going licenses, the BSD’s haven’t caught on with the same ferocity and hype that Linux has. BSD machines are out on the Internet doing the good work, but they aren’t as well known. Some people attribute that to the legal issues that kept BSD-Lite from being released earlier, giving Linux a crucial lead. Others say that the nature of the communities surrounding each is radically different, with BSD’s being more closed to newbies and outsiders.

When Linux and BSD fans come around each other, there are several key differences that pop to the top once they decide to quit mincing words and pull out the brass knuckles.

In the end, it’s irrelevant. For people who love the essentials of UNIX, both provide a good option at no cost. The differences that make choosing one or the other are hard to pin down, since both the BSDs and Linux are close to ubiquitous across hardware platforms and have different strengths depending on the needs of the user.

If there’s a lesson to be taken from this and earlier parts of our narrative,

it's simply that UNIX, for a variety of reasons, has enjoyed an unprecedented following. Even when operating systems were at their most closed and inaccessible, people have been working on ways to preserve key pieces of the UNIX experience. It says quite a bit about how important the UNIX legacy is to quite a few people that we felt a little guilty only covering four UNIX variants, knowing full well that there are many, many more.

BSD machines are out on the Internet doing the good work, but they aren't as well known.

THE BIRTH OF LINUX

So in 1991, whether anyone realized it or not, the computing world was primed for an operating system that would bring the power and flexibility of UNIX to the desktop PC's that were becoming more and more common and inexpensive. Then Linus Torvalds posted a message...

From: torvalds@klaava.Helsinki.FI (Linus Benedict Torvalds)

Newsgroups: comp.os.minix

Subject: What would you like to see most in minix?

Summary: small poll for my new operating system

Message-ID: <1991Aug25.205708.9541@klaava.Helsinki.FI>

Date: 25 Aug 91 20:57:08 GMT

Organization: University of Helsinki

Hello everybody out there using minix -

I'm doing a (free) operating system (just a hobby, won't be big and professional like gnu) for 386(486) AT clones. This has been brewing; since april, and is starting to get ready. I'd like any feedback on things people like/dislike

in minix, as my OS resembles it somewhat (same physical layout of the file-system (due to practical reasons) among other things). I've currently ported bash(1.08) and gcc(1.40), and things seem to work. This implies that I'll get something practical within a few months, and I'd like to know what features most people would want. Any suggestions are welcome, but I won't promise I'll implement them :-)

Linus (torvalds@kruuna.helsinki.fi)

PS. Yes - it's free of any minix code, and it has a multi-threaded fs. It is NOT portable (uses 386 task switching etc), and it probably never will support anything other than AT-harddisks, as that's all I have :-).

The “Minix” he was referring to is a variant on UNIX (still in use today), written by Andy Tannenbaum, a professor of computer science and author of some well-regarded books on the subject. Minix provided something that acted a lot like UNIX on the more and more popular Intel-based computers, but it had limitations.

Linus, reacting to those limitations, took a small bit of working code he had and used Minix as a guideline and supporting infrastructure in the earliest stages of his project. As his mail mentioned, he'd already begun the work of porting some of the tools Richard Stallman's GNU project had provided

(Bash is the shell in most common use on Linux systems and gcc is a program for compiling programs from source code.) For the most part, though, what he had wasn't so much an operating system as it was a simple kernel: the part of an operating system

By virtue of licensing and some of the earliest tools ported to the fledgling operating system, Linux and GNU were closely tied.

that controls the most basic functions of a computer and provides a way for programs to interact with the hardware (in the form of input, output, networking, or other functions) and each other.

Initially, Linus had also intended to release his kernel under a non-commercial license, allowing any and all use by all takers except for businesses who'd use it for profit. He changed his mind eventually, and released it under the GPL originated by RMS and the Free Software Foundation. By virtue of licensing and some of the earliest tools ported to the fledgling operating system, Linux and GNU were closely tied.

With a second message, Linus announced that he was releasing Linux to the world:

From: torvalds@klaava.Helsinki.FI (Linus Benedict Torvalds)

Newsgroups: comp.os.minix

Subject: Free minix-like kernel sources for 386-AT

Message-ID: <19910ct5.054106.4647@klaava.Helsinki.FI>

Date: 5 Oct 91 05:41:06 GMT

Organization: University of Helsinki

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 :-)

As I mentioned a month(?) ago, I'm working on a free version of a minix-lookalike for AT-386 computers. It has finally reached the stage where it's even usable (though may not be depending on what you want), and I am willing to

put out the sources for wider distribution. It is just version 0.02 (+1 (very small) patch already), but I've successfully run `bash/gcc/gnu-make/gnu-sed/compress` etc under it.

Sources for this pet project of mine can be found at `nic.funet.fi` (128.214.6.100) in the directory `/pub/OS/Linux`. The directory also contains some README-file and a couple of binaries to work under linux (`bash`, `update` and `gcc`, what more can you ask for :-). Full kernel source is provided, as no minix code has been used. Library sources are only partially free, so that cannot be distributed currently. The system is able to compile "as-is" and has been known to work. Heh. Sources to the binaries (`bash` and `gcc`) can be found at the same place in `/pub/gnu`.

So, with a very basic operating system built from Linus' kernel and a handful of GNU tools, Linux was born. It would be appropriately dramatic and narration-minded to leave it at that, but it's important to note one element of the union of Linux (the

kernel) with GNU (the attempt to build a Free Software implementation of UNIX) that continues to stir up some conflict from time to time, depending on whether it's a slow news day or not.

It had always been Richard Stallman's intent to build a complete operating system, which involves not only software tools, but a kernel. The

GNU project's attempts to build that kernel were moving fairly slowly (they continue to this day with the ambitious "HURD" project), and with the arrival of Linus' kernel there was no longer

So, with a very basic operating system built from Linus' kernel and a handful of GNU tools, Linux was born.

a need to wait: GNU tools could be mated to the Linux kernel, and an operating system was ready to go.

RMS and many others have long argued that since the operating system most people simply call Linux was largely dependent on GNU tools (and since it continues to depend on GNU tools for much of its basic functionality), it's most appropriate to call it "GNU/Linux." RMS says it's giving credit where it's due, others say it's trying to take too much credit when there are plenty of other elements that make up Linux (the operating system) that have nothing to do with GNU at all.

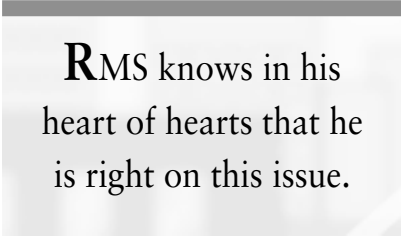
There are a couple of elements at work that have given this debate more life than you might think it would enjoy otherwise:

For one, RMS knows in his heart of hearts that he is right on this issue. People who set out to rewrite an entire operating system because it's the moral thing to do aren't given to backing down on any point, no matter how trivial a point of nomenclature it is to a less involved populace. He's widely respected (as he should be) and a large number of people think his argument resonates.

For two, an equal number of people think he's wrong on this issue, and argue that even if GNU tools are important, they aren't indispensable.

For three, an even larger number than the other two groups combined have no opinion and really don't want to use the phrase "GNU/Linux" where "Linux" will suffice to convey what you're talking about when it comes to "that operating system that Linus Torvalds is associated with."

We've copped out to common usage in this book, but both authors are certain Linux needed GNU tools in its early days to



RMS knows in his heart of hearts that he is right on this issue.

get where it is today. There are some people who have other issues with Richard Stallman and the Free Software movement in general who will disagree: We think they're wrong and believe it serves little to downplay the importance of GNU to the Linux operating system because of issues outside simple history.

LINUX ON THE RISE

There have been a lot of attempts to explain why Linux grew in popularity as rapidly as it did. Only ten years after those first messages announcing it, plenty of people have tried to sum the phenomenon up while it's still underway, and there's no doubt that in twenty years people who write about operating systems will see the whole thing differently. But it's pretty clear that there are a few things that contributed.

First, Linux provided exactly what its creator wanted: a UNIX-like operating system that ran on the Intel-based PC hardware that was growing in popularity during the early '90s. UNIX was very popular at universities, and the alternative on the most common PC's of the time was Microsoft's MS-DOS, which, though largely derivative of "serious" operating systems in some of its elements, was never built with hackers in mind. Students and computer scientists were picking up PC's for their homes more and more during this period as it became a less than \$1000 proposition to own fairly powerful hardware, and they wanted a bit of UNIX on those machines: not something designed for consumers with all the inherent design compromises.

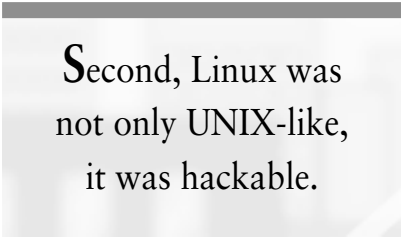
Second, Linux was not only UNIX-like, it was hackable. People could get into the workings of the operating system and make it better if they so chose, or at least have access to the people writing the software to ask for improvements. Looking back over the storied history of UNIX (20 years old at this point), loss of the UNIX source code to litigation and corporate

imperatives had been a real blow to many, who had cut their teeth on poring over the very blueprints of an operating system. Linux, though crude, restored some of what they'd lost, and it held the promise of at least providing something they could mold into something bigger and better. In the ensuing years, another related element of Linux's popularity was derived from its ability to run on ubiquitous hardware, which is that it runs very well on hardware others might be ready to discard. Old 486's make great Linux servers, handling mail and file sharing with aplomb while many other commercial products have long since stopped trying to work on all but fairly new equipment.

Third, the arrival of Linux was closely matched to the spread of the Internet. Though around for as long as UNIX, the Net was becoming a daily reality for more and more people. It became easier and easier to download software, communicate, and collaborate over the Internet with each passing day. By the early '90s, Net connected terminals in student dorm rooms were far from uncommon, and even a university employee could easily connect via a modem to their employer's computers on campus. The development community that sprang up around Linux was global, and the Net held it together.

After the initial flurry of interest in Linux, another interesting facet of Linux culture developed: the *distribution*.

One of the true pains of getting a working Linux machine going was downloading all the needed bits and compiling them, or even just getting them onto a machine. People answered this problem with distributions, which were simply all the pieces of a Linux-based operating system copied onto floppies or CD-ROM's and, well, distributed. Because Linux and the bulk of the software



Second, Linux was not only UNIX-like, it was hackable.

was available under the FSF's GPL (which provided for free and open redistribution of software provided its source code was made available), anybody could download it, set it up in such a manner that it worked out-of-the-box for most people, and redistribute it as a "distribution."

Initially, some of these were crude lash-ups, designed merely to get Linux onto a machine. Others, though, became more and more polished as Linux evangelists began to realize that the easier it was for a second, less expert tier of computer enthusiasts to get Linux up and running, the further it would spread.

Some companies formed around the business of selling Linux distributions, offering incentives to pay for something that could be had for free in the form of convenience and even support via phone or e-mail if something went wrong. Of the current dis-

tributions, Slackware is perhaps the one left today with the longest lineage back to the early days of Linux, but in terms of Linux's time on the Earth, Red Hat (the first widely-successful commercial distribution) and Debian (a distribution run entirely by volunteers) are both looking "long in the tooth."

An early notion of some commercial Linux distributors was that their profit would come from support for installation and administration of Linux computers.

An early notion of some commercial Linux distributors was that their profit would come from support for installation and administration of Linux computers. In shorthand,

they'd make money selling shrink-wrapped, pre-packaged products that the general public would feel more secure with than a simple download and no phone support. This notion survived for quite a few years, relatively speaking. We'll see shortly that it probably won't survive the end of 2001.

The Open-Source Explosion

Linux was gaining mindshare at an incredible rate, thanks to distributions, be they companies or not, it was becoming more and more accessible. Their efforts certainly put Linux in easy reach of any moderately motivated hobbyist who was willing to read all the documentation and figure out how to boot his computer from a floppy disk or CD-ROM.

At the same time, though, some in the Linux community were beginning to identify what they considered a problem: The label that RMS had applied to his attempt to reengineer the way we all thought about software, “free software,” with all the ambiguity inherent in the English word “free.” Stallman never meant that software was to “cost nothing,” but rather had coined the phrase to represent the freedom of the source code itself to be copied and redistributed. Among many computer enthusiasts, though, “free software” often carried the connotation of being something a developer wouldn’t be able to sell if she wanted to. “Freeware” was typically viewed as low quality or incomplete, something companies gave away to prove to you that you needed to buy something better.

Further, despite the fact many had benefited from Stallman’s GNU project, there was a growing sense that the implicit politics of the Free Software Foundation would scare off corporate adoption of Linux, thanks to its language, which less charitable people would characterize as “communistic” in the very least charitable sense of the word.

When you pause to consider how much UNIX hackers loath rebuilding any wheel, what seemed like a constant reiteration of the FSF definition of “free software” was becoming truly irritating to many Linux advocates, who believed Linux had a place as a “serious” operating system if only it could shake off the notion of the less informed that it was a cheap freebie of disposable value.



One person key to the eventual movement to market this notion out of existence was Eric S. Raymond, who, like RMS, goes by his initials. ESR was known for several projects he maintained, including a piece of software for downloading mail (called “fetchmail”), his dictionary of hackish language (called “The New Hacker’s Dictionary” in its print form and “The Jargon File” in its online version), and his contributions to a file that helps UNIX machines understand how to talk to a number of computer terminals from the days when “dumb terminals” were the primary way to communicate with large, multi-user computers. ESR is also a self-styled anthropologist of the hacker community, and a “tribal historian” for the same.

According to Raymond, what was required was not a continual effort to rehabilitate the negative connotations of “free software,” but to invent a new marketing approach. Further, his

own politics were such that the language found throughout the Free Software Foundation's licensing and manifestos was disagreeable; he sought to depoliticize free software.

Along with another prominent figure in the early Linux community, Bruce Perens (former project leader of the Debian distribution), and a collection of others, ESR led a push to market the name *open-source software* as a new way of thinking about the sort of code sharing in which free-software advocates had been engaging in up to that point. They presented the "Open-Source Definition," which created a set of criteria by which software could be determined to be open-source, and monitored the many licenses under which software could be released to determine their compatibility with the Open-Source Definition.

Although the Open-Source Definition clearly paid due respect to the FSF's GPL, open-source supporters also placed a heavy emphasis on the development methodology elements of the Linux and open-source worlds. They played down the political beliefs of early free-software advocates in favor of expressing the benefits of open source as a set of design paradigms that helped eliminate bugs and provided more secure software through the massive peer review of an extended community of hackers.

The effects of this are still felt today. Whereas many companies would have been revolted at the thought of "giving away their intellectual property," they took to the idea of letting many people work on a project in parallel. That way, the collective creativity of those involved would expose bugs more rapidly and provide solutions to problems more quickly.

ESR is also a self-styled anthropologist of the hacker community, and a "tribal historian" for the same.

Of late, this has led to several takes on open source. These range from “companies giving away all their source code, allowing the developer community to do what it will,” to “gated communities,” wherein companies maintain tight control of their software. These gated communities admit few, if any, outsiders

into their development process, while trying to re-create the organizational model that has served many free/open-source software projects well.

Briefly, software prepared by proprietary interests is typically handled in a “cathedral” fashion, with a single, small team working in isolation to develop the project.

In addition to promoting open-source software, ESR went to work on a series of papers that he hoped would encapsulate the defining characteristics of open-source software development. The most famous was the first, entitled “The Cathedral and the Bazaar,” (often referred to as “CatB”), after what Raymond identified as the primary metaphors dominating software development.

Briefly, software prepared by proprietary interests is typically handled in a “cathedral” fashion, with a single, small team working in isolation to develop the project. The “bazaar” model, on the other hand, involves a large collection of developers working on the parts of the project that most interest them. The bazaar model is characterized by a “take all comers” spirit that pays less attention to the arbitrary definitions of corporate affiliations or workplace product groups and more to the individual merit of the developer approaching a project: If you can contribute meaningfully and gain the respect of those maintaining a project, you can work on it.

CatB was a popular piece of work. It had the benefit of drawing examples from an actual “bazaar style” project (fetchmail),

and it summarized in very short form what all the fuss was about with open source. It had an impact on executives at Netscape that sealed its importance when they announced in 1998 that they were releasing the source code to their browser software to the world. Some pegged Netscape's "gift" as more of an opportunistic attempt to exploit the buzz enjoyed by open-source software, but the impact of the move was felt in terms of enhanced prestige for open-source software. If a corporation as prominent as Netscape was willing to try out this development model, maybe there was something to it.

THE LINUX STOCK ORGY AND *LINUX TODAY* MANIA

In the wake of the notoriety Mozilla lent to open-source software, the media began to pay attention to Linux in a way it hadn't before. People could understand what Netscape was, and some would stick around to listen to the new development model it was using. In turn, they were often directed to Linux as "the most successful adherent to open-source methodologies."

Seemingly overnight, public awareness of the operating system exploded, and a few companies began to position themselves to go public and make some money. Red Hat, the distribution company, and VA

Linux, a hardware manufacturer, both had stellar initial public offerings of stock (IPO's) that turned some of their employees and early investors into overnight paper millionaires. The riotous atmosphere surrounding these two companies, fueled by an ever-booming stock market and increasing buzz about Linux

People could understand what Netscape was, and some would stick around to listen to the new development model it was using.

itself as it became better suited to more and more tasks, made them seem like the surest bets going.

Other companies began to pile on. If they couldn't offer stock, they just took to announcing "Linux support" or "the open sourcing" of key software. For half a year, Linux stocks rose and rose, and then the bottom fell out. Companies eager to break into public trading withdrew IPO's as analysts began to ask hard

If they couldn't offer stock, they just took to announcing "Linux support" or "the open sourcing" of key software.

questions about how Linux companies would make money, and companies already on the market suffered calamitous drops, going from highs of over \$200 a share to lows hovering around \$5 a share. A few smaller companies nearly went under, and in the case of Corel, a company that embraced Linux and created its own distribution to complement its WordPerfect product, sold off big parts of their Linux operations.

The hardest question people were asking was simply "If this thing is free, and everybody's learning how to use it, and it's getting easier and easier to install (one version allowed you to play Tetris while waiting for Linux to finish installation) where's the revenue for installation support going to come from?" Others wanted to know how long hardware companies specializing in Linux would hold up once giants like Dell and Compaq began to adopt it, as they were at ever-increasing levels.

In many ways, the answer has come from the companies themselves in the form of realignments around new models built to earn money from providing "services" instead of "support." Where companies once proposed to make money with phone support, they now offer remote network management

and security auditing. Where it was once fashionable to think simply packaging a lot of extra software on a CD was enough, companies are beginning to realize the real money lies in contracts with giant corporations adopting Linux for use in their day-to-day operations.

So we're left with a wrap-up of our brief history of Linux, back on the convention floor in New York.

The stock hysteria has died, Linux has gone from being a hobbyist's toy to a serious operating system companies are willing to depend on for critical operations. It's also turned up in tiny, embedded devices that control industrial machines, and in handheld computers. The same year that some Linux companies went from booming darlings of the tech industry to near-bankrupt failures, IBM has announced it wants to invest \$1 billion in Linux development, and other companies are joining in.

In other words: Linux, born outside of commercial interest, may well outlive many of the first generation of companies that sought to make it a commercial phenomenon. It's a mistake to believe that its success is tied into how well it does in the stock market, or even how well companies pushing it as a sure-fire way to make money fare.

As much, though, as Linux is becoming a corporate phenomenon, it's also still a "people" phenomenon. It's great for companies, sure, but it's also wonderful for people. For every company that stands up and proclaims that Linux is useful for something after all, there are hundreds and thousands of people driving its

In other words: Linux, born outside of commercial interest, may well outlive many of the first generation of companies that sought to make it a commercial phenomenon.

use on personal computers around the world. “The Linux community” might have once been a handful of far-flung programmers working toward the goal of having UNIX on their new 386’s. Now it’s a far-flung collection of users and hackers of all sorts who use Linux every day. This community is what makes it a joy.