The Parable of Unix

Faiz Surani

Two of humanity's most consequential achievements came in July of 1969. The first—you may have heard of it—was the moon landing. The second, in the less exciting venue of a New Jersey office park, was the invention of the Unix operating system at AT&T's Bell Labs. But where the moon landing represented a pinnacle of scientific achievement, Unix was the humble beginning of a computing revolution. Today, Unix derivatives run on the vast majority of computing devices in existence. From smartphones to desktop computers to embedded systems in household appliances, there are tens of billions of Unix systems out in the world.

The story of how Unix came to be is an object lesson in serendipity, but also one of the innovative power of collaboration and openness. Unix dually benefited from and created a culture of "free software," the idea that computer programs should be available to all and that users should be able to study and modify those programs as they see fit.[1] That culture is responsible not just for Unix's world domination, but also the flourishing of the multi-trillion software industry that has reshaped nearly every aspect of the human experience.

Software innovation has thrived for decades in large part because of relatively weak intellectual property enforcement by both companies and courts, enabling competition through interoperability and open standards that drives the whole industry forward.

But while the creation of open-source culture may have been part-accident, preserving it will take careful, deliberate action. When companies have treated software as property from which to leech every last unit of value, as AT&T attempted to do with Unix in its later years and as Oracle presently does, they

[1] Richard Stallman, *The GNU Manifesto*, Free Software Foundation (1985).

have rendered considerable damage to the software community's collaborative nature.

The precarious existence of open-source culture results from a shambolic regulatory regime of software copyrights that was meant for literary works and fails to address the novel challenges presented by work that is both expressive and eminently functional. In short, it is manifestly unfit for the task of regulating the one of the world's most important industries. But it can be fixed, and Congress ought to act to preserve the spirit of collaboration and competition that has built the modern world.

The story of the most important piece of software ever written begins, funnily enough, with a piece of plastic called the "Hush-a-Phone." In 1948, Hush-a-Phone Corporation (HAPC) filed a complaint with the FCC, alleging that AT&T's ban on external equipment—in this case, a plastic cup that attached to telephone microphones—on any part of their phone network was illegal and anticompetitive.[2]

Details of the case eventually made their way to the Truman administration's Justice Department, where Antitrust Division chief Holmes Baldridge was itching for a fight. A year after HAPC's complaint, Baldridge filed suit against AT&T and its manufacturing subsidiary Western Electric, alleging they had "established a monopoly in the manufacture, distribution, and sale of telephone equipment."[3] But the case dragged on for years, and Baldridge's successors under the Eisenhower administration did not share his same enthusiasm for trust busting.

In 1956, the Justice Department entered into a consent decree with AT&T in which it allowed AT&T to retain Western Electric. Given that victory, the company was happy to agree to the conditions attached: that AT&T "was enjoined and restrained from engaging... in any business other than the furnishing of common carrier communication services" and "was required

[2] James Pelkey, Entrepreneurial Capitalism & Innovation: A History of Computer Communications 1968–1988, Chapter B.10 (2007).

[3] *Id.*

to license Bell patents to any applicant that agreed to pay a reasonable royalty."[4]

13 years later, in the summer of 1969, Ken Thompson of Bell Labs was growing bored. Bell Labs had recently pulled out of the Multics operating system project, leaving Thompson without a system on which to run a video game he had written, Space Travel.[5] So, he got together with his colleague Dennis Ritchie and they wrote their own operating system. It took them a little under a month.

Unix continued to evolve and see wider use within AT&T, reaching a staggering 16 (sixteen) installations in February 1973. But everything changed that October, when Thompson and Ritchie presented a paper detailing the Unix system at a symposium of the Association for Computing Machinery.[6]

Unix was a smash hit. Its elegant and modular design was a work of Einsteinian genius.[7] All of a sudden, AT&T was inundated with requests from research labs and academic institutions for copies of the system.[8] There was a problem, though. Under the 1956 consent decree, the company wasn't allowed to be in any business other than "common carrier communication services." Unix was not that, and AT&T's (rightfully) paranoid lawyers were in no mood to draw the ire of the Justice Department over some toy computer software that wouldn't amount to anything.[9]

Luckily, what Bell Labs *could* do (and in fact, were required to do) under the decree was license the system and its source code to anyone who asked

---

[4] *Id.*

[5] Eric S. Raymond, The Art of Unix Programming, Chapter 2 (2003).

[6] Warren Toomey, *The Strange Birth and Long Life of Unix*, IEEE Spectrum (2011).

[7] One academic later recounted his reaction upon reading the journal article detailing Unix: "It was sort of like being hit in the head with a rock. . . I got up and went out of my office, around the corner to [my colleague] . . ., threw the issue down on his desk and said: 'How could this many people have been so wrong for so long?... We've been crazy. This is what we want.'" Peter H. Salus, The Daemon, the Gnu, and the Penguin, Chapter 3 (2008).

[8] Raymond, *supra* note 5, at Chapter 2.

[9] Salus, *supra* note 7, at Chapter 2.

for a nominal fee. And that, they did in spades.[10] Soon, Unix was in use at universities and labs all around the world.[11]

Unix was a singularly odd offering. Bell Labs didn't provide any support for the system[12]— they couldn't, legally—so users were left with the source code and well wishes for happy hacking.

So they got to work. The University of Toronto created typesetting software for Unix. Graduate students are Purdue made massive performance optimizations to the Unix core. With the help of an on-sabbatical Ken Thompson, Berkeley graduate student Bill Joy and his team produced set of utilities and improvements to Unix so comprehensive, it would eventually evolve into its own operating system, BSD.[13]

But what was truly revolutionary about this moment was not this newfound model of extensibility and user freedom—though those were extraordinary too—but instead that the community shared this all with each other. Patches flew between universities on magnetic tapes and even to Bell Labs itself [14] Unix user groups formed, and entire curriculums sprung up around understanding and building on the Unix base. For the first time, software was being built not by a person or even a company, but by a community. It was the birth of open source.

Not all of this was strictly legal. Much of the distribution of Unix code

[10] Toomey, *supra* note 6.

[11] Raymond, *supra* note 5, at Chapter 2.

[12] "[F]or many years Bell Labs researchers proudly displayed their Unix policy at conferences with a slide that read, "No advertising, no support, no bug fixes, payment in advance." Toomey, *supra* note 6.

[13] Raymond, *supra* note 5, at Chapter 2.

[14] Bell Labs was not allowed to distribute patches or bug fixes of any kind as part of its legal obligation to not support the software. That is not to say that it stopped them. "Lou Katz, the founding president of [Unix user group] Usenix, received a phone call one day telling him that if he went down to a certain spot on Mountain Avenue (where Bell Labs was located) at 2 p.m., he would find something of interest. Sure enough, Katz found a magnetic tape with the bug fixes, which were rapidly in the hands of countless users." Dennis Ritchie later suggested the tape had perhaps "fallen from a truck." Toomey, *supra* note 6; Salus, *supra* note 7, at Chapter 3.

and documentation ran afoul of some AT&T license restriction or other. But AT&T was prohibited from making money off of the system, and so was not particularly focused on those transgressions. In any case, the rule-breaking was part of the fun.

When University of New South Wales computer scientist John Lions produced the authoritative work on Unix (and the only comprehensive documentation available outside AT&T), complete with an annotated copy of the source code, AT&T blocked its use in classrooms and distribution to unlicensed users.[15] Samizdat photocopies subsequently spread far and wide among computer students. Universities were forced to drop Unix material from their curriculums, but students with illicit copies of the Lions book, in the style of Dead Poets Society, continued to study Unix and create patches for it.[16]

The Unix movement, and by extension the open-source movement, cannot be understood merely through the lens of software. In truth, it was as much a countercultural movement as it was an operating system. It was Woodstock for geeks. Copying Lions' book was not simply the sharing of knowledge, but a revolutionary act, a Galilean quest for truth, a way for scrappy kernel hackers to stick it to the powers that be. It was a communal spirit that transcended borders and outlasted the social currents that birthed it.

And then AT&T almost killed it.

In 1984, Ma Bell's luck finally ran out. After decades of antitrust disputes, AT&T was finally forced to agree to divestiture. AT&T's various telephone companies were splintered into regional holding companies and spun off. But as part of the settlement, AT&T retained Bell Labs, and crucially, was freed from the terms of the 1956 consent decree that banned it from selling anything other than carrier communication services.[17]

[15] *Id.* Chapter 4.
[16] *Id.*
[17] Raymond, *supra* note 5, at Chapter 2.

By this time, the Unix community was thriving. Version 7 Unix was widely popular and community patches had largely resolved its performance issues.[18] Free of the 1956 decree's restraints, AT&T did the only thing it knew how to do: monopolize and squeeze customers for money. As open-source historian Eric S. Raymond writes:

> Knowing no other model than secrecy for collecting profits from software and no other model than centralized control for developing a commercial product, AT&T clamped down hard on source-code distribution. Bootleg Unix tapes became far less interesting in the knowledge that the threat of lawsuit might come with them. Contributions from universities began to dry up.[19]

Things got grimmer from there. As Unix system vendors struggled to compete with AT&T and each other, they began to seek ways to differentiate their systems from others', or even moved away from Unix altogether in favor of proprietary kernels. The end result was the severe fragmentation of the Unix community and bitter internecine warfare between competing Unix standards. In one instance, AT&T sued Berkeley, alleging its BSD operating system infringed on its copyrights despite the BSD team having previously removed all AT&T code from its Unix-derivative system.[20]

All was not lost, thankfully. Like any revolutionary movement worth its salt, open source had a policy arm. In the mid-1980's, frustrated with his MIT AI Lab colleagues' move to proprietary software, Richard Stallman launched the Free Software Foundation and the GNU Project.[21]

Stallman was a bona fide ideologue who believed in free software, and only free software. Free software, as Stallman defined it, was "free as in free

[18] Salus, *supra* note 7, at Chapter 7.
[19] Raymond, *supra* note 5, at Chapter 2.
[20] *Id.*
[21] Salus, *supra* note 7, at Chapter 9.

speech, not free beer."[22] Indeed, Stallman had no problem per se with paid software, on the condition that a user must always have access to the source code of any program they use so that they could modify and redistribute it as they saw fit.[23]

To that end, Stallman sought to stave off the drift towards proprietary, closed-source software (which he termed "software hoarding") that he had observed among his MIT colleagues. In 1984, he released the first version of the GNU General Public License (GPL).[24] GPL was the first widely-used "copyleft" license, which allowed users freedom to modify, distribute, and even sell derivatives of GPL-licensed software on the condition that the derivative also be licensed under GPL and have its source code freely available.[25]

Then in the mid-1990's, the project that is today almost synonymous with open source appeared. At the time, the BSD team was mired in copyright disputes with AT&T, crippling its ability to produce a free Unix-like operating system kernel for all to use. So in 1991 Finnish student Linus Torvalds began the Linux project as an open-source effort to create a Unix-like kernel.[26] Torvalds was less of an ideologue than Stallman, but he was adamantly committed to the idea that nobody should have to pay for Linux. That commitment to open access combined with BSD's contemporaneous legal woes proved a potent combination for recruiting outside contributors, and by 1994, the Linux kernel exited beta.[27] From there, Linux was off to the races and would eventually become the world's dominant free kernel, used in everything from embedded systems to supercomputers.

Unix's long, bizarre journey is worth studying in its own right for how it has shaped modern computing. But it also offers valuable historical lessons

[22] Lawrence Lessig, *Free, as in Beer*, Wired Magazine (2006).
[23] Stallman, *supra* note 1.
[24] Li-Cheng Tai, *The History of the GPL*, Free Software (2001).
[25] *Id.*
[26] Christopher Tozzi, *Linux at 25: Why It Flourished While Others Fizzled*, IEEE Spectrum (2016).
[27] *Id.*

about the value of collaboration and interoperability. Collectively, the Unix community created a system magnitudes better than anything the Bell Labs team could have done alone, to say nothing of the thousands of extensions and utilities written for the platform and shared free of charge.

Open source was born by an accident of history, but its continued existence has been anything but. When the tradition of free software came under threat from AT&T's commercialization of Unix and fragmentation of the community, pioneers like Richard Stallman and Linus Torvalds stepped in to foster open-source culture anew.

Today, the open-source tradition is stronger than ever in many ways. Nearly every large technology company—from Google to Facebook to even the historically stingy Microsoft—has invested in contributing to existing projects and maintaining many more of their own. The bulk of Google's Chrome browser is open source, the React library powering Facebook's website too, and on and on. These projects reflect the shared understanding that open collaboration is better not just for the community, but for the company too. As an old adage in the software industry goes, "given enough eyeballs, all bugs are shallow," meaning easy to identify and correct.[28]

But we also stand at a moment of danger of the kind Stallman and Torvalds reacted to. As then, we must act to preserve open source. This time, though, the threat does not come from the broader industry but instead the legal system that underlies it.

To understand this threat, we have to take a brief look at the legal status quo under which open source developed, principally the copyrightability of interfaces. In American copyright law, there is a principle known as the idea-expression dichotomy. The concept here is that ideas are not themselves copyrightable, "protection is given only to the expression of the idea—not the idea itself."[29] That theory combined with early court rulings in software

[28] Scott Merrill, *With Many Eyeballs, All Bugs Are Shallow*, TechCrunch (2012).
[29] *Mazer v. Stein*, 347 U.S. 201, 217 (1954).

copyright cases established the conventional wisdom that software interfaces, like programming languages or file formats, could not be copyrighted, or at the very least would fall under fair use if copied.[30]. That meant anyone was free to create clones of popular programs and the interfaces they offered, like spreadsheet software or operating systems, so long as they wrote the actual functionality themselves.

Even when the AT&T's Unix went proprietary, the understanding that the interfaces it offered to developers could not be owned by anyone was what allowed projects like BSD and Linux to reimplement those interfaces and gain traction in the first place. Without that understanding, interoperability as the entire software industry knows it cannot exist.

Without that understanding, we risk entering what legal scholar Clark D. Asay called a "copyright anticommons," in which different rightsholders refuse to cooperate with each other and prevent use of central technologies, paralyzing an industry.[31] And while that is a depressing potential reality to the rest of us, it makes Larry Ellison feel warm and fuzzy on the inside.[32] Enter *Google v. Oracle*.

Distinguished open-source leader and one-time Oracle engineer Bryan Cantrill famously summarized Oracle's ethos as "Ship mediocrity, inflict misery, lie our asses off, screw our customers, and make a whole shitload of money."[33] It should be self-evident why that theory of business is incompatible with principles of collaboration and fair competition.

---

[30] *See, e.g.*, *Sony Comput. Entm't, Inc. v. Connectix Corp.*, 203 F.3d 596 (9th Cir. 2000) (holding that verbatim copying of an operating system's code for development of an emulator constituted fair use); *Sega Enters. Ltd. v. Accolade, Inc.*, 977 F.2d 1510 (9th Cir. 1992) (holding the copying of Sega's game code to reverse engineer console's safeguards against unlicensed developers constituted fair use).

[31] Clark D. Asay, *Software's Copyright Anticommons*, 66 Emory Law Journal 265, 272 (2016) ("[T]he robustness of software innovation, and information technology innovation more generally, is likely to decrease as parties seek to navigate copyright "thickets" in pursuing collaborative software innovation in an interconnected world.").

[32] *See generally* Mike Wilson, The Difference Between God and Larry Ellison: God Doesn't Think He's Larry Ellison (2003).

[33] Bryan M. Cantrill, *Fork Yeah: The Rise and Development of illumos*, USENIX (2011).

The case was set in motion when, in April 2009, Oracle bought Sun Microsystems for $7.4 billion, and with it, the rights to the Java programming language that Google had previously adopted for use in its Android mobile operating system.[34] Sun had developed and maintained the Java project as a largely open-source endeavor. When Google announced its use of Java as part of the Android project in 2007, Sun offered its "heartfelt congratulations."[35]

But almost immediately after Oracle's purchase of Sun completed, Oracle filed suit against Google, alleging that Google's reuse of Sun's Java interfaces (Google had reimplemented all of the actual functionality itself) violated its copyright.[36]

The case has had a long and tangled procedural history, but the current iteration before the Supreme Court concerns two questions: (1) whether the Java interfaces Google copied were copyrightable in the first place and (2) if so, whether Google's use of them was "fair use" as defined by the Copyright Act. Oracle's theory of the case is fairly straightforward: Google "copied 11,500 lines of respondent's copyrighted code. In doing so, [it] also copied the complex architecture of the 37 packages at issue, including the names and specifications of the thousands of methods and classes in those packages and their hierarchical and interdependent relationships to each other."[37] In Oracle's view, it is a clear-cut copyright infringement case where the copying is far too extensive to be considered minimal or fair use.

Google's contention is a little more involved. Its theory is based on a copyright principle known as "merger doctrine."[38] Merger doctrine dictates that in the specific case of the idea-expression dichotomy where there is only one (or a very limited number) way to express an idea, the expression of that

[34] *Oracle America Inc. v. Google LLC*, 886 F.3d 1179, 1187 (Fed. Cir. 2018).
[35] Dan Farber, *Former Sun CEO says Google's Android didn't need license for Java APIs*, CNET (2012).
[36] *Oracle v. Google*, 886 F.3d 1187.
[37] Brief for Respondent, p. 7, *Google LLC v. Oracle America Inc.*, No. 18-956 (2021).
[38] Brief for Petitioner, *Id.*

idea cannot receive copyright protection. Google contends that, combined with the terse nature of the declarations it reused, it does not infringe on Oracle's copyright by reusing less than 0.1% of the Java implementation.[39]

If the Court rules for Oracle, the effect won't be sudden calamity. But it will undermine the foundations of open-source culture, and could lead to a gradual, yet pronounced decline of the kind experienced by the Unix community after AT&T commercialized the system.

What is exasperating about *Google v. Oracle* is that it is a close case. Google's legal theory, while well-rooted in precedent and history, is a fairly attenuated interpretation of the Copyright Act. But with a legal regime designed for works of literature, it is no surprise that courts have struggled with software's simultaneous functional and expressive roles.

The Supreme Court's job isn't (or at least shouldn't be) to make sound copyright law; it's just supposed to interpret what Congress writes down. Maybe the Court will save Congress from itself like the past several hundred times it has, but to establish the kind of regulatory stability that will allow software collaboration to continue to flourish, it needs to establish a definite statutory framework.

As Microsoft noted in its amicus brief supporting Google, the business model of software has fundamentally changed: "[S]oftware today typically is no longer produced by a single author; instead, more and more software is collaboratively built. In the modern software industry, open collaborative innovation projects" increasingly serve as important sources of innovative products, processes, and services. . . That collaboration. . . has allowed developers to treat pieces of code as building blocks."[40]

The "building block" model is what has enabled the industry to morph and change as quickly as it has. By abstracting away some of the most diffi-

[39] Brief for Petitioner, p. 9, *Id.*

[40] Brief for Microsoft Corporation as Amicus Curiae, p. 7-8, *Id* (internal citations and quotations omitted).

cult challenges of development—the operating system, the server runtime, version control to name just a few—open source projects allow businesses to focus on innovating with their core technology and offerings rather than replicating work already done *ad nauseam*. Open source is good for free software, yes, but it is an unquestionable benefit to even proprietary work.

How do we protect it? Congress can start by making some things clear: interfaces cannot be copyrighted, for one. But as a broader approach, it should also recognize that in software, unlike literature, derivative work has tremendous communal and economic value. Software copyright claims should be restricted to only the most egregious theft in recognition of the unique tradition of sharing and interoperability in software that has made the industry so successful. The ability of companies like AT&T and Oracle to upend successful open-source communities with frivolous lawsuits is simply too high a price to pay for any benefits strong copyrights afford creators. As Sun Microsystem's CEO wrote upon leaving the company in the aftermath of the Oracle acquisition, "for a technology company, going on offense with [software intellectual property] seems like an act of desperation, relying on the courts instead of the marketplace."[41] Today's tech giants are indisputable proof that immense success is not at odds with the free software tradition. The danger to the industry comes from too much copyright enforcement, not too little.

Andy Warhol once observed that what made America great was the fact that all Americans—from the lowliest of wage workers to the wealthiest of tycoons—all consumed essentially the same things. "A Coke is a Coke and no amount of money can get you a better Coke than the one the bum on the corner is drinking. All the Cokes are the same, and all the Cokes are good. Liz Taylor knows it, the President knows it, the bum knows it, and you know it."[42]

So too for open source. The startup culture whereby any geek with a com-

[41] Farber, *supra* note 35.
[42] Andy Warhol, The Philosophy of Andy Warhol, 101 (1975).

puter and a good idea can disrupt eons-old giants of industry is inextricably dependent on the "building blocks" model that open source enables. It's good economics, sure. But it's also a triumph of democratic values worth protecting in its own right.

Industries develop strange, often Kafkaesque norms through the various flukes of history. But once in a while, a truly great notion takes hold. Open source is good for the industry, and good for the world. But it is not impervious. Without a serious regulatory effort to reform the sorry, confused state of software copyright law (let alone patents), the principles of collaboration and interoperability that enabled the computing revolution will remain under dire threat from would-be monopolists like Oracle. It's long past time for Congress to put in place a sane framework that places collaboration and interoperability at its center.

Table of Authorities

Cases Page

Books

Articles Page

Table of Authorities