Coding Freedom

THE ETHICS AND AESTHETICS OF HACKING

E. GABRIELLA COLEMAN

PRINCETON UNIVERSITY PRESS
PRINCETON AND OXFORD

Published by Princeton University Press, 41 William Street, Princeton,
New Jersey 08540

In the United Kingdom: Princeton University Press, 6 Oxford Street,
Woodstock, Oxfordshire OX20 1TW

press.princeton.edu

At the time of writing of this book, the references to Internet Web sites (URLs) were accurate. Neither the author nor Princeton University Press is responsible for URLs that may have expired or changed since the manuscript was prepared.

Library of Congress Cataloging-in-Publication Data

Coleman, E. Gabriella, 1973–

Coding freedom : the ethics and aesthetics of hacking / E. Gabriella Coleman.

p. cm.

Includes bibliographical references and index.

ISBN 978-0-691-14460-3 (hbk. : alk. paper)—ISBN 978-0-691-14461-0 (pbk. : alk. paper) 1. Computer hackers. 2. Computer programmers. 3. Computer programming—Moral and ethical aspects. 4. Computer programming—Social aspects. 5. Intellectual freedom. I. Title.

HD8039.D37C65 2012

174'.90051--dc23 2012031422

British Library Cataloging-in-Publication Data is available

This book has been composed in Sabon

Printed on acid-free paper. ∞

Printed in the United States of America

1 3 5 7 9 10 8 6 4 2

INTRODUCTION

A Tale of Two Worlds

Free and open-source software (F/OSS) refers to nonproprietary but licensed software, much of which is produced by technologists located around the globe who coordinate development through Internet-based projects. The developers, hackers, and system administrators who make free software routinely include the following artifact in the software they write:

> This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

While seemingly insignificant, this warning is quite meaningful for it reveals something important about the nature of free software and my subsequent representation of it. This legal notice is no doubt serious, but it also contains a subtle irony available to those who know about free software. For even if developers cannot legally guarantee the so-called FITNESS of software, they know that in many instances free software is often as useful as or in some cases superior to proprietary software. This fact brings hackers the same sort of pleasure, satisfaction, and pride that they derive when, and if, they are given free reign to hack. Further, even though hackers distribute their free software WITHOUT ANY WARRANTY, the law nevertheless enables them to create the software that many deem superior to proprietary software—software that they all "hope [. . .] will be useful." The freedom to labor within a framework of their own making is enabled by licenses that cleverly reformat copyright law to prioritize access, distribution, and circulation. Thus, hackers short-circuit the traditional uses of copyright: the right to exclude and control.

This artifact points to the GNU General Public License (GPL), an agreement that many hackers know well, for many use it (or other similar licenses) to transform their source code—the underlying directions of all software—into "free software." A quick gloss of the license, especially its preamble, reveals a more passionate language about freedom and rights:

> When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.[1]

This type of language spills far beyond licensing agreements. It is routinely voiced in public discourse and everyday conversation. Such commitments to freedom, access, and transparency are formalized in a Linux distribution known as Debian, one of the most famous free software projects. These values are reflected in a pair of charters—the Debian Constitution and the Debian Social Contract—that articulate an organizational vision and formulate a set of promises to the wider free software community. These charters' names alone unmistakably betray their liberal roots, even if they were not explicitly created with the goal of "advancing" liberal ideals.

By liberalism, I do not mean what may first come to mind: a political party in Europe usually associated with politicians who champion free market solutions, or in the United States, a near synonym for the Democratic Party. Nor is it just an identity that follows from being a proud, card-carrying member of the American Civil Liberties Union or Electronic Frontier Foundation, although these certainly can be markers.

Here I take liberalism to embrace historical as well as present-day moral and political commitments and sensibilities that should be familiar to most readers: protecting property and civil liberties, promoting individual autonomy and tolerance, securing a free press, ruling through limited government and universal law, and preserving a commitment to equal opportunity and meritocracy. These principles, which vary over time and place, are realized institutionally and culturally in various locations at different times. Perhaps the most famous of these are the institutions of higher education, market policies set by transnational institutions, and the press, but they are also at play on the Internet and with computer hackers, such as those who develop free software.[2]

The small statement that prefaces the GNU GPL thus hints at two elements of this community: one is esoteric, and grounded in technology and its material practices; and the other concerns a broader, culturally familiar vision of freedom, free speech rights, and liberalism that harks back to constitutional ideals. We should not take either for granted but instead open them up to critical reflection, and one route to do so is by bringing them together. This ethnography takes seriously free software's visions of liberty and freedom as well as the mundane artifacts that hackers take pleasure and joy in creating. In considering them together, important lessons are revealed about the incomplete, sometimes fraught, but nonetheless noticeable

relationship between hacking and liberalism, and the transformations and tensions evident within the liberal tradition and computer hacking.

A Liberal Critique within Liberalism

The terms free and open as applied to software are distinct yet often come paired. This is in part because they designate the same alternative licenses and collaborative methodologies, but they differ in their moral orientation: the term free software foremost emphasizes the right to learn and access knowledge, while open source tends to flag practical benefits.[3] Many participants, whether they are volunteers or corporate employees paid to work on free software, refer to themselves with pride as hackers—computer aficionados driven by an inquisitive passion for tinkering and learning technical systems, and frequently committed to an ethical version of information freedom.

Although hackers hold multiple motivations for producing their software, collectively they are committed to *productive freedom*. This term designates the institutions, legal devices, and moral codes that hackers have built in order to autonomously improve on their peers' work, refine their technical skills, and extend craftlike engineering traditions. This ethnography is centrally concerned with how hackers have built a dense ethical and technical practice that sustains their productive freedom, and in so doing, how they extend as well as reformulate key liberal ideals such as access, free speech, transparency, equal opportunity, publicity, and meritocracy.

I argue that F/OSS draws from and also rearticulates elements of the liberal tradition. Rather than designating only a set of explicitly held political, economic, or legal views, I treat liberalism in its cultural registers.[4] Free software hackers culturally concretize a number of liberal themes and sensibilities—for example, through their competitive mutual aid, avid free speech principles, and implementation of meritocracy along with their frequent challenge to intellectual property provisions. Indeed, the ethical philosophy of F/OSS focuses on the importance of knowledge, self-cultivation, and self-expression as the vital locus of freedom. Hackers bring these values into being through an astounding range of social and technical practices, covered in detail throughout this book.

Because hackers challenge one strain of liberal jurisprudence, intellectual property, by drawing on and reformulating ideals from another one, free speech, the arena of F/OSS makes palpable the tensions between two of the most cherished liberal precepts—both of which have undergone a significant deepening and widening in recent decades. Thus, in its political dimension, and even if this point is left unstated by most developers and advocates, F/OSS represents a liberal critique from within liberalism. Hackers sit simultaneously at the center and margins of the liberal tradition.

The expansion of intellectual property law, as noted by some authors, is part and parcel of a broader neoliberal trend to privatize what was once public or under the state's aegis, such as health provision, water delivery, and military services. "Neoliberalism is in the first instance," writes David Harvey (2005, 2), "a theory of political economic practices that proposes human well-being can be best advanced by liberating entrepreneurial freedoms and skills within an institutional framework characterized by strong property rights, free markets, and free trade." As such, free software hackers not only reveal a long-standing tension within liberal legal rights but also offer a targeted critique of the neoliberal drive to make property out of almost anything, including software.

While most of this ethnography illustrates how free software hacking critiques neoliberal trends and reinvents liberal ideals by asserting a strong conception of productive freedom in the face of intellectual property restrictions, it also addresses the material, affective, and aesthetic dimensions of hacking. In pushing their personal capacities and technologies to new horizons (and encountering many frustrations along the way), hackers experience the joy that follows from the self-directed realization of skills, goals, and talents. At times, hacking provides experiences so completely overpowering, they hold the capacity to shred self-awareness, thus cutting into a particular conception of the liberal self—autonomous, authentic, and rational—that these hackers otherwise routinely advance. Thus, at least part of the reason that hacker ethics takes its liberal form is connected to the aesthetic experiences of hacking, which are informed by (but not reducible to) liberal idioms and grammars. Hacking, even if tethered to liberal ideologies, spills beyond and exceeds liberal tenets or liberal notions of personhood, most often melding with a more romantic sensibility concerned with a heightened form of individual expression, or in the words of political theorist Nancy Rosenblum (1987, 41), a "perfect freedom."

Fieldwork among Hackers

For most of its history, anthropology stuck close to the study of non-Western and small-scale societies. This started to shift following a wave of internal and external critiques that first appeared in the 1960s, expanded in the 1970s, and peaked in the 1980s. Now referred to as "the critical turn in anthropology," the bulk of the critique was leveled against the discipline's signature concept: culture. Critics claimed that the notion of culture—as historically and commonly deployed—worked to portray groups as far more bounded, coherent, and timeless than they actually are, and worse, this impoverished rendition led to the omission of topics concerning power, class, colonialism, and capitalism (Abu-Lughod 1991; Asad 1973; Clifford 1988; Clifford and Marcus 1986; Dirks 1992; Said 1978). Among other

effects, the critique cracked open new theoretical and topical vistas for anthropological inquiry. An anthropologist like myself, for example, could legitimately enter nontraditional "field sites" and address a new set of issues, which included those of technoscientific practice, information technologies, and other far-flung global processes stretching from labor migration to transnational intellectual property regulations.

Partly due to these disciplinary changes, in winter 2000, I left a snowy Chicago and arrived in a foggy San Francisco to commence what cultural anthropologists regard as our foundational methodological enterprise: fieldwork. Based on the imperative of total immersion, its driving logic is that we can gain analytic insight by inserting ourselves in the social milieu of those we seek to understand. Fieldwork mandates long-term research, usually a year or more, and includes a host of activities such as participating, watching, listening, recording, data collecting, interviewing, learning different languages, and asking many questions.

When I told peers of my plan to conduct fieldwork among hackers, many people, anthropologists and others, questioned it. How does one conduct fieldwork among hackers, given that they just hang out by themselves or on the Internet? Or among those who do not understand the name, given that they are all "outlaws"? Often playfully mocking me, many of my peers not only questioned how I would gather data but also routinely suggested that my fieldwork would be "so easy" (or "much easier than theirs") because I was studying hackers in San Francisco and on the Internet.

The subtext of this light taunting was easy enough to decipher: despite the transformations in anthropology that partially sanctioned my research as legitimate, my object of study nonetheless still struck them as patently atypical. My classmates made use of a socially acceptable medium—joking—to raise what could not be otherwise discussed openly: that my subjects of study, primarily North American and European (and some Latin American) hackers, were perhaps too close to my own cultural world for critical analysis, or perhaps that the very activity of computing (usually seen as an instrumental and solitary activity of pure rationality) could be subject only to thin, anemic cultural meanings.[5]

By the turn of the twenty-first century, although anthropology had certainly "reinvented" itself as a field of study—so that it is not only acceptable but one is in fact, at some level, also actively encouraged to study the West using new categories of analysis—Michel-Rolph Trouillot (2003, 13) has proposed that "anthropologists reenter the West cautiously, through the back door, after paying their dues elsewhere." As a young, aspiring anthropologist who was simply too keen on studying free software during graduate school and thus shirked her traditional dues, I knew that for myself as well as my peers, my project served as an object lesson in what constitutes an appropriate anthropological "location" (Gupta and Ferguson 1997) for study—in particular for graduate students and young scholars.

I myself wondered how I would ever recognize, much less analyze, forms of cultural value among a group of mostly men of relatively diverse class and national backgrounds who voluntarily band together online in order to create software. Would I have to stretch my ethnographic imagination too far? Or rely on a purely formal and semiotic analysis of texts and objects—a methodology I wanted for various reasons to avoid? Amid these fears, I took some comfort in the idea that, as my peers had indicated, my initial fieldwork would be free of much of the awkwardness that follows from thrusting oneself into the everyday lives of those who you seek to study, typically in an unfamiliar context. At the very least, I could communicate to hackers in English, live in a familiar and cosmopolitan urban setting, and at the end of the day, return to the privacy and comfort of my own apartment.

As it turned out, my early ethnographic experiences proved a challenge in many unexpected ways. The first point of contact, or put more poetically by Clifford Geertz (1977, 413), "the gust of wind stage" of research, was harder than I had imagined. Although not always discussed in such frank terms among anthropologists, showing up at a public gathering, sometimes unannounced, and declaring your intent to stay for months, or possibly years, is an extraordinarily difficult introduction to pull off to a group of people you seek to formally study. More difficult is describing to these strangers, whose typical understanding of anthropology stems from popular media representations like the Indiana Jones trilogy, our methodology of participant observation, which is undertheorized even among anthropologists.[6] Along with the awkwardness I experienced during the first few weeks of fieldwork, I was usually one of the only females present during hacker gatherings, and as a result felt even more out of place. And while I may have recognized individual words when hackers talked shop with each other—which accounted for a large percentage of their time—they might as well have been speaking another language.

At the start of my research period, then, I rarely wanted to leave my apartment to attend F/OSS hacker social events, user group meetings, or conferences, or participate on email lists or Internet relay chat channels—all of which were important sites for my research. But within a few months, my timidity and ambivalence started to melt away. The reason for this dramatic change of heart was a surprise to me: it was the abundance of humor and laughter among hackers. As I learned more about their technical world and was able to glean their esoteric jokes, I quickly found myself enjoying the endless stream of jokes they made in all sorts of contexts. During a dinner in San Francisco's Mission district, at the office while interning at the Electronic Frontier Foundation, or at the monthly gatherings of the Bay Area Linux User Group held in a large Chinatown restaurant, humor was a constant bedfellow.

Given the deep, bodily pleasures of laughter, the jovial atmosphere overcame most social barriers and sources of social discomfort, and allowed me

to feel welcome among the hackers. It soon became clear to me, however, that this was not done for my benefit; humor saturates the social world of hacking. Hackers, I noticed, had an exhaustive ability to "misuse" most anything and turn it into grist for the humor mill. Once I began to master the esoteric and technical language of pointers, compilers, RFCs, i386, X86, AMD64, core dumps, shells, bash, man pages, PGP, GPG, gnupg, OpenPGP, pipes, world writeable, PCMCIA, chmod, syntactically significant white space, and so on (and really on and on), a rich terrain of jokes became sensible to me.

My enjoyment of hacker humor thus provided a recursive sense of comfort to a novice ethnographer. Along with personally enjoying their joshing around, my comprehension of their jokes indicated a change in my outsider status, which also meant I was learning how to read joking in terms of pleasure, creativity, and modes of being. Humor is not only the most crystalline expression of the pleasures of hacking (as I will explore later). It is also a crucial vehicle for expressing hackers' peculiar definitions of creativity and individuality, rendering partially visible the technocultural mode of life that is computer hacking. As with clever technical code, to joke in public allows hackers to conjure their most creative selves—a performative act that receives public (and indisputable) affirmation in the moment of laughter. This expression of wit solidifies the meaning of archetypal hacker selves: self-determined and rational individuals who use their well-developed faculties of discrimination and perception to understand the "formal" world—technical or not—around them with such perspicuity that they can intervene virtuously within this logical system either for the sake of play, pedagogy, or technological innovation. In short, they have playfully defiant attitudes, which they apply to almost any system in order to repurpose it.

A few months into my research, I believed that the primary anthropological contribution of this project would reside in discussing the cultural mores of computer hacking, such as humor, conjoined with a methodological analysis of conducting research in the virtual space of bits and bytes. Later in my fieldwork, I came to see the significance of another issue: the close relationship between the ethics of free software and the normative, much broader regime of liberalism. Before expanding on this connection, I will first take a short ethnographic detour to specify *when* it became unmistakably apparent that this technical domain was a site where liberal ideals, notably free speech, were not only endowed with concrete meaning but also made the fault lines and cracks within liberalism palpably visible.

It was August 29, 2001, and a typical San Francisco day. The abundant morning sun and deep blue skies deceptively concealed the reality of much cooler temperatures. I was attending a protest along with a group of

about fifty programmers, system administrators, and free software enthusiasts who were demanding the release of a Russian programmer, Dmitry Sklyarov, arrested weeks earlier in Las Vegas by the Federal Bureau of Investigation (FBI) as he left Defcon, the largest hacker conference in the world. Arrested at the behest of the Silicon Valley software giant Adobe, Sklyarov was charged with violating the recently ratified and controversial Digital Millennium Copyright Act (DMCA). He had written a piece of software, the Advanced eBook Processor software, for his Russian employer. The application transforms the Adobe eBook format into the Portable Document Format (PDF). In order for the software to perform this conversion, it breaks and therefore circumvents the eBook's copy control measures. As such, the software violated the DMCA's anticircumvention clause, which states that "no person shall circumvent a technological protection measure that effectively controls access to a work protected under this measure."[7]

We had marched from the annual LinuxWorld conference being held in San Francisco's premier conference center, the Moscone Center, to the federal prosecutor's office. Along the way, a few homeless men offered solidarity by raising their fists. Two of them asked if we were marching to "Free Mumia"—an assumption probably influenced by the recent string of protests held in Mumia Abu-Jamal's honor. Indeed, as I learned soon after my first arrival in San Francisco, the city is one of the most active training grounds in the United States for radical activists. This particular spring and summer was especially abuzz with activity, given the prominence of counterglobalization mobilizations. But this small and intimate demonstration was not typical among the blizzard of typically left-of-center protests, for none of the participants had a way of conveying quickly nor coherently the nature of the arrest, given how it was swimming in an alphabet soup of acronyms, such as DRM, DMCA, and PDF, as opposed to more familiar ideas like justice and racism. A few members of our entourage nonetheless heartily thanked our unlikely though clearly sympathetic supporters, and assured them that while not as grave as Mumia's case, Dmitry's situation still represented an unfair targeting by a corrupt criminal justice system, especially since he was facing up to twenty-five years in jail "simply for writing software."

Once at the Hall of Justice, an impassioned crew of programmers huddled together and held up signs, such as "Do the Right Thing," "Coding Is Not a Crime," and "Code Is Speech."

There must have been something about directly witnessing such fiery outpourings among people who tend to shy away from overt forms of political action that led me to evaluate anew the deceptively simple claim: code is speech. It dawned on me that day that while I had certainly heard this assertion before (and in fact, I was only hearing it increasingly over time), it was more significant than I had earlier figured. And after some research,

FIGURE INTRO.1. Protesting the DMCA, San Francisco
Photo: Ed Hintz.

it was clear that while the link between free speech and source code was fast becoming entrenched as the new technical common sense among many hackers, its history was remarkably recent. Virtually nonexistent in published discourse before the early 1990s, this depiction now circulates widely and is routinely used to make claims against the indiscriminate application of intellectual property law to software production.

Early in my research, I was well aware that the production of free software was slowly but consistently dismantling the ideological scaffolding supporting the expansion of copyright and patent law into new realms of production, especially in the US and transnational context. Once I considered how hackers question one central pillar of liberal jurisprudence, intellectual property, by reformulating ideals from another one, free speech, it was evident that hackers also unmistakably revealed the fault line between two cherished sets of liberal principles.

While the two-hundred-year history of intellectual property has long been freighted with controversies over the scope, time limits, and purpose of various of its instruments (Hesse 2002; Johns 2006, 2010; McGill 2002), legal scholars have only recently given serious attention to the uneasy coexistence between free speech and intellectual property principles (McLeod 2007; Netanel 2008; Nimmer 1970; Tushnet 2004). Copyright law, in granting creators significant control over the reproduction and circulation of their work, limits the deployment of copyrighted material in other expressive activity, and consequently censors the public use of certain forms of expressive content. Legal scholar Ray Patterson (1968, 224) states this dynamic eloquently in terms of a clash over the fundamental values of a democratic society: "A society which has freedom of expression as a basic principle of liberty restricts that freedom to the extent that it vests ideas with legally protected property interests."

Because a commitment to free speech and intellectual property is housed under the same roof—the US Constitution—the potential for conflict has long existed. For most of their legal existence, however, conflict was

noticeably absent, largely because the scope of both free speech and intellectual property law were more contained than they are today. It was only during the course of the twentieth-century that the First Amendment and intellectual property took on the unprecedented symbolic and legal meanings they now command in the United States as well as many other nations. (Although the United States has the broadest free speech protections in the world, many other Western nations, even if they limit the scope of speech, have also expanded free speech and intellectual property protections in the last fifty years.)

For example, copyright, which grants authors significant control over their expression of ideas, was initially limited to fourteen years with one opportunity for renewal. Today, the copyright term in the United States has ballooned to the length of the author's life plus seventy years, while works for hire get ninety-five years, regardless of the life of the author. The original registration requirement has also been eliminated. Most any expression—a scribble on a piece of paper, a blog post, or a song—automatically qualifies for protection, so long as it represents the author's creation.

Free speech jurisprudence follows a similar trajectory. Even though the Constitution famously states that "Congress shall make no law [. . .] abridging the freedom of speech, or of the press," during the first half of the twentieth century the US Supreme Court curtailed many forms of speech, such as political pamphleteering, that are now taken to represent the heart and soul of the democratic process. It is thus easy to forget that the current shape of free speech protections is a fairly recent social development, largely contained within the last fifty years (Bollinger and Stone 2002).

Due to the growing friction between free speech and intellectual property, US courts in the last twenty-five years have openly broached the issue by asserting that any negative consequences of censoring speech are far outweighed by the public benefit of copyright law. In other words, as a matter of public policy, copyright law represents an acceptable restriction on speech because it is the basis for what is designated as "the marketplace of ideas."[8] The theory animating the marketplace of ideas is that if and when ideas are allowed to publicly compete with each other, the truth—or in its less positivist form, the best policy—will become evident.

Given this historical trajectory, the use of F/OSS licenses challenges the current, intellectual property regime, growing ever more restrictive, and thus dubbed ominously by one legal scholar as the contemporary motor for "the second enclosure movement" (Boyle 2003). Many free software developers do not consider intellectual property instruments as the pivotal stimulus for a marketplace of ideas and knowledge. Instead, they see them as a form of restriction so fundamental (or poorly executed) that they need to be counteracted through alternative legal agreements that treat knowledge, inventions, and other creative expressions not as property but rather as speech to be freely shared, circulated, and modified.

The Aesthetics of Hacking

If free software hackers render the tensions between two liberal principles visible, and offer a targeted, if not wholesale, critique of neoliberalism in challenging intellectual property law (but rarely using the language of neoliberalism), their commitment to free speech also puts forth a version of the liberal person who strays from the dominant ideas of liberal personhood: a self-interested consumer and rational economic seeker. Among academics, this has often been placed under the rubric of "possessive individualism," defined as "those deeply internalized habits of thinking and feeling [. . .] viewing everything around them primarily as actual or potential commercial property" (Graeber 2007, 3; see also Macpherson 1962). Among hackers, selfhood has a distinct register: an autonomous being guided by and committed to rational thought, critical reflection, skills, and capacity—a set of commitments presupposed in the free speech doctrine (Peters 2005).[9]

However important these expressive and rational impulses are among programmers, they don't fully capture the affective stances of hackers, most notably their deep engagement, sometimes born of frustration, and at other times born of pleasure, and sometimes, these two converge. Soon after commencing fieldwork, what I quickly learned is that hacking is characterized by a confluence of constant occupational disappointments *and* personal/collective joys. As many writers have noted, and as I routinely observed, hacking, whether in the form of programming, debugging (squashing errors), or running and maintaining systems (such as servers), is consistently frustrating (Rosenberg 2007; Ullman 2003). Computers/software are *constantly* malfunctioning, interoperability is frequently a nightmare to realize, users are often "clueless" about the systems they use (and therefore break them or require constant help), the rate and pace of technological change is relentless, and meeting customer expectations is nearly impossible to pull off predictably. The frustration that generally accompanies the realities of even mundane technical work is depicted as swimming with sharks in *xkcd*, one of the most beloved geeks' comic strips (figure Intro.2).

What this comic strip captures is how hackers, as they work, sometimes swim in seas of frustration. To tinker, solve problems, and produce software, especially over one's lifetime, will invariably be marked by varying degrees of difficulties and missteps—a state of laboring that one theorist of craftspersonship describes as material "resistance" (Sennett 2008). In encountering obstacles, adept craftspeople, such as hackers, must also build an abundant "tolerance for frustration" (ibid., 226), a mode of coping that at various points will break down, leading, at best, to feelings of frustration, and at worst, to anguish and even despair and burnout.

Despite these frustrations and perhaps because of them, the craft of hacking demands a deep engagement from hackers, or a state of being most commonly referred to in the literature as "flow" (Csikszentmihalyi 1990).

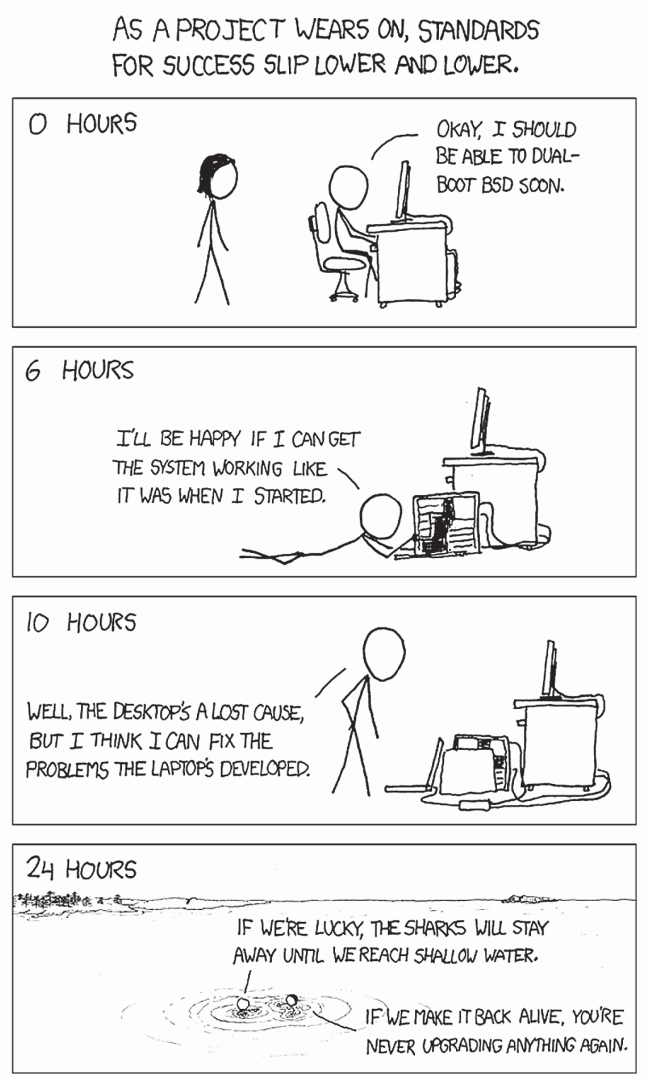

FIGURE INTRO.2. "Success," *xkcd*
Credit: Randall Munroe.

In its more mild and commonplace form, hacker pleasure could be said to approximate the Aristotelian theory of eudaemonia, defined succinctly by philosopher Martha Nussbaum (2004, 61) as "the unimpeded performance of the activities that constitute happiness." In pushing their personal capacities and skills though playing around with and making technologies, hackers experience the joy that follows from the self-directed realization of skills, goals, and talents. Indeed, overcoming resistance and solving problems, some

of them quite baffling, is central to the sense of accomplishment and pride that hackers routinely experience.

Hacker pleasure, however, is not always so staid and controlled; it far exceeds the pride of eudaemonia. Less frequently, but still occurring often, hackers experience a more obsessive and blissful state. Hacker descriptions of immersing themselves in technology remind me of Rainer Maria Rilke's terse and beautiful depiction of the passion that drives his intellectual pursuits: "All the soarings of my mind begin in my blood." This form of pleasure approximates what Roland Barthes (1975) has portrayed as bliss or jouissance—a pleasure so complete, engrossing, and enveloping that it has the capacity to obliterate every last shred of self-awareness. In native hack jargon, the state of bliss is the "Deep Hack Mode." Matt Welsh, a well-known hacker and computer scientist, humorously describes the utter magnetism of this mode, "very few phenomena can pull someone out of Deep Hack Mode, with two noted exceptions: being struck by lightning, or worse, your *computer* being struck by lightning."[10]

Because hackers often submit their will and being to technology—and are famous for denying their bodies sleep, at least for short periods—the joy that hackers derive from attending to and carefully sculpting technologies are at times experienced as transcendent bliss. In these moments, utility is exceeded. The self can at once express its most inner being and collapse within the objects of its creation. In the aftermath of a particularly pleasurable moment of hacking, there is no autonomous liberal self to be found.

To be sure, these forms of pleasure and engagement were impossible for me, the ethnographer, to touch and feel. But I routinely witnessed the social markers of the joy of hacking, as hackers talked shop with each other, as they joked about technical minutiae, and especially during their festive hacker celebrations. The key point is that the multifaceted pleasures of hacking signal that utility is not the only driving force in hackers' creative acts. Although hackers are fiercely pragmatic and utilitarian—technology after all must work, and work exceptionally well—they are also fiercely poetic and repeatedly affirm the artistic elements of their work. One of the clearest expressions of technology/software as art is when source code is written as poetry, or alternatively when poetry is written in source code (Black 2002). For many free software hackers, the act of writing software and learning from others far exceeds the simple enactment of an engineering ethic, or a technocratic calculus for the sake of becoming a more proficient as well as efficient programmer or system administrator.

This is hacking in its more romantic incarnation—a set of characterizations and impulses that hold an affinity with liberalism, and yet also stray into different, largely aesthetic and emotional territory. Liberalism, as a body of thought, certainly allows for pleasure, but for the most part does not theorize the subjective and aesthetic states of pleasure, which the Romantic tradition has centralized and made its own. Romanticism, explains

Rosenblum (1987, 10), is a "lavish departure from sober individualism," but also "amounts to an exploitation of liberal ideals." Although it is important to differentiate liberal from romantic sensibilities, they nonetheless can coexist without much friction, as Rosenblum contends in her account on Romanticism. She draws on various prominent historical figures, such as John Stuart Mill and Henry David Thoreau, to examine the compatibilities and symbiosis between liberalism and Romanticism. Hackers, borrowing from free speech commitments *and* also committed to aesthetic experiences, are a social group whose sensibilities lie at the interface between a more rational liberal calculus and a more aesthetic, inward-looking one.

Hackers are not alone in embracing this aesthetic, expressive sensibility, which philosopher Charles Taylor (1992) argues persuasively is a fundamental part of our contemporary imaginary, or what he calls the "expressive self." First visibly emerging in the eighteenth century, this sentiment formed the basis for "a new fuller individualism," and places tremendous weight on originality, sentiments, creativity, and at times, even disengagement. What must be noted is that expressive individualism and the moral commitments it most closely entails—self-fulfillment, self-discovery, and self-improvement—can be secured, as many critics have shown, through consumption, self-help, human enhancement technologies, and body modification (Bellah et al. 1985; Elliott 2003; Hogle 2005), and thus can converge seamlessly with elements of possessive individualism. Today to liberate and express the "authentic," "expressive" self is usually synonymous with a lifelong engagement with consumption, fine tuned by a vast advertising apparatus that helps sustain the desire for a seemingly limitless number of consumer goods and, increasingly, human enhancement technologies such as plastic surgery.

The example set by free software (and a host of similar craftlike practices), however, should make us at least skeptical of the extent to which an ethic of consumption has colonized expressive individualism. Free software hackers undoubtedly affirm an expressive self rooted not in consumption but rather in production in a double sense: they produce software, and through this technical production, they also sustain informal social relations and even have built institutions. Given the different ethical implications entailed in these visions of fulfillment, expression, and self-development (consumerist versus productive self), it behooves us to analytically pry them apart.

While the liberal articulations made by free software hackers, notably those of free speech, carry a familiar political imprint, their material experiences, the frustrations and pleasures of hacking, (including the particularities of making, breaking, and improving software) might seem politically irrelevant. Yet the passionate commitment to hacking and especially the ethics of access enshrined in free software licensing, express as well as celebrate unalienated, autonomous labor, which also broadcasts a powerful political

message. A number of theorists (Galloway 2004; Söderberg 2007; Wark 2004) have previously highlighted this phenomenon. Hackers insistence on never losing access to the fruits of their labor—and indeed actively seeking to share these fruits with others—calls into being Karl Marx's famous critique of estranged labor: "The external character of labour for the worker appears in the fact that it is not his own, but someone else's, that it does not belong to him, that in it he belongs, not to himself, but to another" (Marx and Engels 1978, 74). It evokes Marx's vision precisely because free software developers seek to avoid the forms of estrangement that have long been nearly synonymous with capitalist production. Freedom is thus not only based on the right to speak free of barriers but also conceived as (although primarily through practice) "the utopian promise of unalienated labor, of human flourishing through creative and self-actualizing production," as Barton Beebe (2010, 885) aptly describes it.

F/OSS hacker morality is therefore syncretic—a quality that is also patently evident in its politics. It enunciates a liberal politics of free speech and liberty that speaks to an audience beyond hackers as well as a nonliberal politics of cultural pleasure and political detachment, which is internally and intensely focused on the practice of hacking only and entirely for its own sake, although certainly inspiring others to follow in their footsteps. When assessing the liberal ethics and affective pleasure of hacking, we should not treat pleasure as the authentic face of hacking, and the other (liberalism) as an ideological veneer simply in need of debunking (or in need of celebrating). From an ethnographic vantage point, it is important to recognize many hackers are citizens of liberal democracies, and have drawn on the types of accessible liberal tropes—notably free speech—as a means to conceptualize their technical practice and secure novel political claims. And in the process, they have built institutions and sustain norms through which they internalize these liberal ideals as meaningful, all the while clearly upholding a marked commitment to unalienated labor.

On Representing Hacker Ethics

If I was comforted by the fact that hacking could be analyzed in light of cultural issues like humor, liberalism, and pleasure, and that I had some methodological tools at my disposal to do so, as I learned more about hacking, my ease vanished as I confronted a new set of concerns. I increasingly grew wary of how I would convey to others the dynamic vitality and diversity that marks hackers and hacking, but also the points of contention among them. To further illustrate this point, allow me to share a brief story.

Soon after ending my official fieldwork, I was having dinner in Chicago with three local free software developers. One of them asked me about some of my memorable fieldwork experiences. There were many stories I could

have chosen, but I started to tell the story of a speech by Kevin Mitnick—a more transgressive hacker (for he had engaged in illegal behavior) than most free software developers and one of the most infamous of all time—that I heard during summer 2004 at Hackers on Planet Earth (HOPE)—a conference founded in 1994 to publicize his legal ordeals. Mitnick is known to have once been a master "social engineer," or one who distills the aesthetics of illicit acts into the human art of the short cons. Instead of piercing through a technological barricade, social engineers target humans, duping them in their insatiable search for secret information. Because of various legendary (and at times, illegal) computer break-ins, often facilitated by his social engineering skills, Mitnick spent a good number of his adult years either running from the law or behind bars, although he never profited from his hacks, nor destroyed any property (Coleman and Golub 2008; Mitnick 2011; Thomas 2003).

In July 2004, free at last and allowed to use computers again, Mitnick attended HOPE in New York City for the first time. He delivered his humorous and enticing keynote address to an overflowing crowd of hackers, who listened, enraptured, to the man who had commanded their political attention for over a decade as part of a "Free Kevin Campaign." He offered tale after tale about his clever pranks of hacking from childhood on: "I think I was born as a hacker because at ten I was fascinated with magic," he explained. "I wanted a bite of the forbidden fruit." Even as a kid, his victims were a diverse lot: his homeroom teacher, the phone company, and even the Los Angeles Rapid Transit District. After he bought the same device used by bus drivers for punching transfers, he adopted the persona of Robin Hood, spending hours riding the entire bus network, punching his own pirated transfers to give to customers. He found transfer stubs while dumpster diving, another time-honored hacker practice for finding information that was especially popular before the advent of paper shredding. Despite the way that lawyers and journalists had used Mitnick's case to give hackers a bad name, Mitnick clearly still used the term with pride.

When I finished my story describing what I personally thought was a pretty engrossing speech, one hacker, who obviously disapproved of my reference to Mitnick as a "hacker," replied, "Kevin is *not* a hacker. He is a cracker." In the mid-1980s, some hackers created the term cracker to deflect the negative images of them that began appearing in the media at that time. According to *The Hacker Jargon File*, crackers are those who hack for devious, malicious, or illegal ends, while hackers are simply technology enthusiasts. Although some hackers make the distinction between crackers and hackers, others also question the division. To take one example, during an interview, one free software hacker described this labeling as "a whitewashing of what kind of people are involved in hacking. [. . .] Very often the same techniques that are used in hacking 2 [the more illegal kind] are an important part of hacking 1."

To be sure, hackers can be grasped by their similarities. They tend to value a set of liberal principles: freedom, privacy, and access. Hackers also tend to adore computers—the glue that binds them together—and are trained in specialized and esoteric technical arts, primarily programming, system, or Net administration, security research, and hardware hacking. Some gain unauthorized access to technologies, though the degree of illegality varies greatly (and much of hacking is legal). Foremost, hacking, in its different forms and dimensions, embodies an aesthetic where craft and craftiness tightly converge. Hackers thus tend to value playfulness, pranking, and cleverness, and will frequently perform their wit through source code, humor, or both: humorous code.

Hackers, however, evince considerable diversity and are notoriously sectarian, constantly debating the meaning of the words hack, hacker, and hacking. Yet almost *all* academic and journalistic work on hackers commonly whitewashes these differences, and defines all hackers as sharing a singular "hacker ethic." Offering the first definition in *Hackers: Heroes of the Computer Revolution*, journalist Steven Levy (1984, 39) discovered among a couple of generations of MIT hackers a unique as well as "daring symbiosis between man and machine," where hackers placed the desire to tinker, learn, and create technical beauty above all other goals. The hacker ethic is shorthand for a list of tenets, and it includes a mix of aesthetic and pragmatic imperatives: a commitment to information freedom, a mistrust of authority, a heightened dedication to meritocracy, and the firm belief that computers can be the basis for beauty and a better world (ibid., 39–46).

In many respects, the fact that academics, journalists, and hackers alike refer to the existence of this ethic is a testament not only to the superb account that Levy offers—it is still one of the finest works on hacking—but also to the fact that the hacker ethic in the most general sense is an apt way to describe some contemporary ethics and aesthetics of hacking. For example, many of the principles motivating free software philosophy reinstantiate, refine, extend, and clarify many of those original precepts. Further, and rarely acknowledged, Levy's account helped set into motion a heightened form of reflexivity among hackers. Many hackers refer to their culture and ethics. It is an instance of what Marshall Sahlins (2000, 197; see also Carneiro da Cunha 2009) describes as "contemporary culturalism"—a form of "cultural self-awareness" that renders culture into an "objectified value." This political dynamic of self-directed cultural representation is suggested in the following quote by Seth Schoen, an avid free software advocate and staff technologist at the Electronic Frontier Foundation. In the first line of text that appears on his Web page, Schoen announces, with pride: "I read [Levy's *Hackers*] as a teenager. [. . .] I was like, 'God damn it, I should be here!' Then, about ten years later, I thought back about it: 'You know, if there was a fourth section in that book, maybe I would be in there!' That's a nice thought."[11]

As I delved deeper into the cultural politics of hacking, though, I began to see serious limitations in making any straightforward connections between the hacker ethic of the past and the free software of the present (much less other hacker practices). Most obviously, to do so is to overlook how ethical precepts take actual form and, more crucially, how they transform over time. For example, in the early 1980s, "the precepts of this revolutionary Hacker Ethic," Levy (1984, 39; emphasis added) observes, "were not so much debated and discussed as silently agreed upon. *No Manifestos were issued*." Yet (and somewhat ironically) a mere year after the publication of his book, MIT programmer Richard Stallman charted the Free Software Foundation (FSF) ([1996] 2010) and issued "The GNU Manifesto," insisting "that the golden rule requires that if I like a program I must share it with other people who like it."[12] Today, hacker manifestos are commonplace. If hackers did not discuss the intricacies of ethical questions when Levy first studied them, over the span of two decades they would come to argue about ethics, and sometimes as heatedly as they argue over technology. And now many hackers recognize ethical precepts as one important engine driving their productive practices—a central theme to be explored in this book.

Additionally, and as the Mitnick example provided above illustrates so well, the story of the hacker ethic works to elide the tensions that exist among hackers as well as the different genealogies of hacking. Although hacker ethical principles may have a common core—one might even say a general ethos—ethnographic inquiry soon demonstrates that similar to any cultural sphere, we can easily identify great variance, ambiguity, and even serious points of contention.

Therefore, once we confront hacking in anthropological and historical terms, some similarities melt into a sea of differences. Some of these distinctions are subtle, while others are profound enough to warrant what I, along with Alex Golub, have elsewhere called genres of hacking (Coleman and Golub 2008). F/OSS hackers, say, tend to uphold political structures of transparency when collaborating. In contrast, the hacker underground, a more subversive variant of hacking, is more opaque in its modes of social organization (Thomas 2003). Indeed, these hackers have made secrecy and spectacle into something of a high art form (Coleman 2012b). Some hackers run vibrant technological collectives whose names—Riseup and Mayfirst—unabashedly broadcast that their technical crusade is to make this world a better one (Milberry 2009). Other hackers—for example, many "infosec" (information security) hackers—are first and foremost committed to security, and tend to steer clear of defining their actions in such overtly political terms—even if hacking usually tends to creep into political territory. Among those in the infosec community there are differences of opinion as to whether one should release a security vulnerability (often called full disclosure) or just announce its existence without revealing details (referred to as antidisclosure). A smaller, more extreme movement that goes by the name

of antisec is vehemently against any disclosure, claiming, for instance, in one manifesto that it is their "goal that, through mayhem and the destruction of all exploitive and detrimental communities, companies, and individuals, full-disclosure will be abandoned and the security industry will be forced to reform."[13] There is also an important, though currently untold, story about gaming and hacking, not only because hackers created some of the first computer games, notably Space Wars, written in 1962, but because of the formal similarities between gaming and hacking as well (Dibbell 2006).

National and regional differences make their mark as well. For instance, southern European hackers have followed a more leftist, anarchist tradition than their northern European counterparts. Chinese hackers are quite nationalistic in their aims and aspirations (Henderson 2007), in contrast to those in North America, Latin America, and Europe, whose antiauthoritarian stance makes many—though certainly not all—wary of joining government endeavors.

Finally, while the brilliance of Levy's account lies in his ability to demonstrate how ethical precepts fundamentally inhere in hacker technical practice, it is important to recognize that hacker ethics, past and present, are not entirely of their own making. Just a quick gloss of the language many hackers frequently invoke to describe themselves or formulate ethical claims—freedom, free speech, privacy, the individual, and meritocracy—reveals that many of them unmistakably express liberal visions and romantic sensibilities: "We believe in freedom of speech, the right to explore and learn by doing," explains one hacker editorial, "and the tremendous power of the individual."[14] Once we recognize the intimate connection between hacker ethics and liberal commitments *and* the diversity of ethical positions, it is clear that hackers provide less of a unitary and distinguishable ethical position, and more of a mosaic of interconnected, but at times divergent, ethical principles.

Given this diversity, to which I can only briefly allude here, the hacker ethic should not be treated as a singular code formulated by some homogeneous group called hackers but instead as a composite of distinct yet connected moral genres. Along with a common set of moral referents, what hacker genres undoubtedly share is a certain relation to legality. Hacker actions or their artifacts are usually either in legally dubious waters or at the cusp of new legal meaning. Hence, they make *visible* emerging or contentious dilemmas.

Although hackers certainly share a set of technical and ethical commitments, and are in fact tied together by virtue of their heated debates over their differences, given the existence of the diversity just noted, my claims and arguments should not be taken as representative of all hacking, even though for the sake of simplicity (and stylistic purposes), in the chapters that follow I will often just refer to hackers and hacking. My discussion is more modest and narrow for it will stick primarily to the example of

free software.[15] My preference for announcing the "self-conscious, serious partiality" (Clifford 1986, 7) of this account comes from witnessing motivations, ethical perceptions, desires, and practices far more plastic, flexible, sublime, contradictory, and especially fiery and feverish than usually accounted for in academic theories. The world of hacking, as is the case with many cultural worlds, is one of reckless blossoming, or in the words of Rilke: "Everything is blooming most recklessly; if it were voices instead of colors, there would be an unbelievable shrieking into the heart of the night."

Omissions and Chapter Overview

Some readers may be asking why I have not addressed Silicon Valley entrepreneurship and Web 2.0, both of which might further illuminate the ethics and politics of F/OSS.[16] For those interested in Web 2.0—a term that is bandied around to refer to nearly all contemporary digital tools and the social practices that cluster around these technologies—you might want to jump to the short epilogue, where I critique this term. It is a moniker that obscures far more than it reveals, for it includes such a wide range of disparate phenomena, from corporate platforms like Flickr, to free software projects, to dozens of other digital phenomena. In fact, by exploring in detail free software's sociocultural dynamics, I hope this book will make it more difficult to group free software in with other digital formations such as YouTube, as the media, pundits, and some academics regularly do under the banner of Web 2.0.

The relationship between Silicon Valley and open source is substantial as well as complicated. Without a doubt, when it comes to computers, hackers, and F/OSS, this high-tech region matters, as I quickly came to learn within weeks of my arrival there. For the last thirty years, hackers have flocked to the Bay Area from around the world to make it one of their most cherished homelands, although it certainly is not the only region where hackers have settled and set deep roots. At the turn of this century, open source also became the object of Silicon Valley entrepreneurial energy, funding, and hype, even though today the fever for open source has diminished significantly, redirected toward other social media platforms.

The book is thus not primarily about free software in Silicon Valley. In many respects my material tilts toward the North American and European region but, nevertheless, I have chosen to treat free software in more general than regional registers as well, so as to capture the reality of the legal transnational processes under investigation along with the experience of the thousands and thousands of developers across the world. Debian, for example, has developers from Japan, Australia, Canada, New Zealand, all over western and eastern Europe, Brazil, Venezuela, Argentina, and Mexico.[17] I decided on this approach as it is important to demonstrate different values

and dynamics at play than those found in Silicon Valley, which are too often mistaken to represent *the* commitments of all engineers, computer scientists, and hackers.[18]

Coding Freedom is composed of six chapters, divided conceptually into pairs of two. The first two chapters are historically informed, providing the reader with a more general view of free software. Chapter 1 ("The Life of a Free Software Hacker") provides what is a fairly typical life history of a F/OSS hacker from early childhood to the moment of discovering the "gems" of free software: source code. Compiled from over seventy life histories, I demonstrate how hackers interact and collaborate through virtual technologies, how they formulate liberal discourses through virtual interactions, how they came to learn about free software, and how they individually and collectively experience the pleasures of hacking. I also offer an extended discussion of the hacker conference, which I argue is the ritual (and pleasurable) underside of discursive publics. Chapter 2 ("A Tale of Two Legal Regimes") presents what were initially two semi-independent legal regimes that over the last decade have become intertwined. The first story pertains to free software's maturity into a global movement, and the second turns to the globalization and so-called harmonization of intellectual property provisions administered through global institutional bodies like the World Trade Organization. By showing how these trajectories interwove, I emphasize various unexpected and ironic outcomes as I start to elaborate a single development that will continue to receive considerable treatment later in the book: the cultivation, among hackers, of a well-developed legal consciousness.

The next two chapters provide a close ethnographic analysis of free software production. Chapter 3 ("The Craft and Craftiness of Hacking") presents the central motif of value held by hackers by examining the practices of programming, joking, and norms of socialization through which they produce software and their hacker selves. Partly by way of humor, I tackle a series of social tensions that mark hacker interactions: individualism and collectivism, populism and elitism, hierarchy and equality as well as artistry and utility. These tensions are reflected but also partially attenuated through the expression of wit, especially jokes, and even funny code, whereby jokes ("easter eggs") are included in source code. Chapter 4 ("Two Ethical Moments in Debian") addresses ethical cultivation as it unfolds in the largest free software project in the world—Debian. This project is composed of over one thousand developers who produce a distribution of the Linux operating system (OS). I present and theorize on the tensions between Debian's governance, which blends democratic majoritarian rule, a guildlike meritocracy, and ad hoc deliberations. In comparing these three modes of governance, I unearth various ethical processes—informal, formal, pedagogical, and dramatic—by which Debian developers inhabit a liberally based philosophy of free software, and use it as an opportunity to revisit the tension between liberal individualism and corporate sociality explored earlier.

The final two chapters engage with more overtly political questions, examining two different and contrasting political elements of free software. Chapter 5 ("Code Is Speech") addresses two different types of legal pedagogy common among free software developers. First, in the context of Debian, I look at everyday legal learning, where debating and learning about the law is an integral part of project life. I then compare this with a series of dramatic arrests, lawsuits, and political protests that unfolded between 1999–2004 in the United States, Europe, and Russia, and on the Internet, and that allowed for a more explicit set of connections to be drawn between code and speech. These demonstrations were launched against what was, at the time, a relatively new copyright statute, the DMCA, and the arrest of two programmers. These multiyear protests worked, I argue, to stabilize a relatively nascent cultural claim—nearly nonexistent before the early 1990s—that source code should be protected speech under the First Amendment (or among non-American developers, protected under free speech laws). In contrast to the political avowal of the DMCA protests, my conclusion ("The Politics of Disavowal and the Cultural Critique of Intellectual Property Law") discusses how and why hackers disavow engagement in broad-based politics, and instead formulate a narrow politics of software freedom. Because a commitment to the F/OSS principles is what primarily binds hackers together, and because many developers so actively disavow political associations that go beyond software freedom, I contend that the technoscientific project of F/OSS has been able to escape the various ideological polarizations (such as liberal versus conservative) so common in our current political climate. F/OSS has thus been taken up by a wide array of differently positioned actors and been placed in a position of significant social legibility whereby it can publicly perform its critique of intellectual property law.

Finally, to end this introduction, it is worth noting that this book is not only an ethnography but also already an archive of sorts. All cultural formations and ethical commitments are, of course, in motion, undergoing transformation, and yet many technological worlds, such as free software, undergo relentless change. What is written in the forthcoming pages will provide a discrete snapshot of F/OSS largely between 1998 and 2005. Much of this book will still ring true at the time of its publication, while other elements have come and gone, surely to have left a trace or set of influences, but no longer in full force. And despite my inability to provide a warranty for this archival ethnography, I hope such an account will be useful in some way.

CHAPTER 3

The Craft and Craftiness of Hacking

I have nothing to declare but my genius.
—Oscar Wilde

I, for the first time, gave its proper place among the prime necessities of human well-being, to the internal culture of the individual.
—John Stuart Mill, *Autobiography*

Hackers value cleverness, ingenuity, and wit. These attributes arise not only when joking among friends or when hackers give talks but also during the process of making technology and writing smart pieces of code. Take, for example, this short snippet of what many hackers would consider exceptionally clever code written in the computer language Perl:

```
#count the number of stars in the sky
    $cnt = $sky =~ tr/*/*/;
```

This line of Perl is a hacker homage to cleverness; it is a double entendre of semantic ingenuity and technical wittiness. To fully appreciate the semantic playfulness presented here, we must look at the finer points of a particular set of the developer population, the Perl hacker. Perl is a computer language in which terse but technically powerful expressions can be formed (in comparison to other programming languages). Many Perl coders take pride in condensing long segments of code into short and sometimes intentionally confusing (what coders often call "obfuscated") one-liners (Monfort 2008). If this above line of code were to be "expanded" into something more traditional and accessible to Perl novices, it might read something like:

```
$cnt = 0;
    $i = 0;
    $skylen = length($sky)
    while ($i < $skylen) {
```

```
    $sky = substr($sky,0, $i) . '*' . substr($sky, $i+1,
    length($skylen));
    $i++;
}
$cnt = length($sky);
```

We see that the Perl programmer has taken six lines of code and reduced them to a single line by taking advantage of certain side effects found in the constructs of the Perl language, and the very act of exploiting these side effects is a great example of a hack. With this transformation of "prose" into terse "poetry," the developer displays a mastery of the technical aspect of the language. This mastery is topped on the semantic level by a quip. The programmer has named the variable $sky, and the star is the asterisk (*) character.[1] The counting function in this program counts any appearance of the asterisk symbol—hence, "counting the number of stars in the sky." This code has a technical function, but within a community of peers, its performance is also a declaration and demonstration of the author's savvy.

Hackers will publicly acknowledge such acts of "genius" and are thus fiercely meritocratic—in ideology and practice. Yet given that so much of hacker production is collective, a fact increasingly acknowledged and even celebrated in the ethical philosophy of F/OSS, a commitment to individuality, meritocracy, and independence is potentially subverted by the reality of as well as desire to recognize their fundamental interdependence. The belief in the value of individuality coupled with the constant need for the help of other hackers points to a subtle paradox that textures their social world. The tension between individualism and collectivism, in particular, is negotiated through the extremely well-developed and common penchant that hackers have for performing cleverness, whether through technological production or humor. Hackers do not treat all forms of expression, technology, and production as original and worthy expressions of selfhood. Instead, one must constantly manifest, in the face of one's peers, a discriminating and inventive mind by performing its existence through exceptionally ingenious and clever acts. By contributing a shining, awe-inspiring sliver of their creative self in a domain otherwise characterized by a common stock of knowledge and techniques, hacker utilize humor or clever code to perform their craftiness, and thus momentarily differentiate themselves from the greater collective of hackers.

While this chapter describes the ethnographic expression of humor and cleverness among hackers (which might be valuable and interesting in its own right), it does so at the service of other, analytic goals. Examining humor and cleverness will allow me to more richly demonstrate how tensions (say, between individualism and collectivism) arise through the course of technological practice, and how hackers partially resolve them. Taking a close look at these frictions takes us a long way toward understanding

the social context under which these hackers labor and why free speech ideals—in contrast to those of intellectual property instruments—resonate with their experiences. The friction between individualism and collectivism (and its articulation in meritocratic discussions) helps, for one, underwrite a dynamic social environment in which hackers labor. Second, this tension speaks directly to issues of authorship, selfhood, creativity, and intellectual property in a way that extends, contrasts, and critiques the dominant intellectual property regime.

The analysis opens by examining the pragmatics and aesthetics of hacking, by which I mean the constraints and properties of their technological activities, and contrasting the writings of two hackers, Espe and Da Mystik Homeboy (DMH). Understanding the pragmatics of hacking is necessary to grasp the contradictions/tensions that mark hacking along with what I call the poetics of hacking: the extreme value hackers place on ingenuity, craftiness, and cleverness. I will explore these largely through the angle of humor. The final section revisits the tension between individualism and collectivism. Hackers assert a form of individualism that valorizes self-expression and development among peers engaged in similar acts of technological production, while tightly entangled with each other through constant collaboration.

Hacker Pragmatics

Python: Reaching a Transcendental Space

> I remember when I found python, back in the 1.52 days [1.52 refers to a version number].[2] I was an unemployed slacker living in a student co-op. I'd sit in a (since disappeared) cafe in Berkeley and write reams of more or less useless code, simply for the joy of it. I'd reach some sort of transcendental state fueled by relevant whitespace, clear syntax, and pints of awfully strong, black coffee. In those days I first felt the pure abstract joy of programming in a powerful way—the ability to conjure these giant structures, manipulate them at will, have them contain and be contained by one another. I think I learned more in those couple of months, thanks to Google and a free ricochet connection, than in my previous years in CS [computer science].
>
> Eventually, however, it became clear I had to get a real job. Flaky freelance contracts which never paid sucked so hard. So, I hemmed and hawed and was conflicted and finally got a job, and it involved perl. It was, perhaps, a worst-case perl scenario. A very rapidly growing website, a few developers with vastly different styles, a lack of real communication, and a pronounced lack of appreciation for namespaces. From my high tower of control and purity, I'd been thrown into a bubbling pool of vaguery and confusion. Cryptic variables would pop out of

> the aether, make an appearance in a 2000 line CGI [Common Gateway Interface], and never be heard from again. Combating naming schemes would meet where different spheres of developer influence overlapped—$postingTitle and $PostingTitle doing battle in the same subroutine. Scripts almost—but not quite—deprecated. The situation is quite a bit more under control now, 3 years later.
>
> *— Espe*

PERL: HACKING IN THE BIG BALL OF MUD

> Perl has been derided by many people as an ugly, difficult to learn language that enforces bad habits. I generally do not advocate perl to people who are attempting to learn programming, or even mention it's existence. However, perl, for better or worse, is a culmination of decades of culture. Perl is a Unix Gematria—an arcane relation of symbols evolved in a manner similar to Jewish Qabbalistic numerology. Many other languages, such as python or Java, attempt to enforce a strict framework and rule set of contracts, interfaces, strong typing, and private methods to delineate functionality. While much of this stems from noble traditions of SmallTalk and ML [they are computer languages], much of it also fails to realize the point of these ancestral languages: categorization (such as through strict typing and object models) is itself a form of computation. When this fact is not respected, you wind up with a bastardized language that is [. . .] Anal.
>
> Perl was designed by a linguist, and realizes that people have different things to say in different contexts, and your language is defined by the environment and not vice versa. As Paul Graham said, both the world and programming is a "Big Ball of Mud," which perl has evolved around. The implicit variables, the open object model, the terse expressions all contribute to hacking on the Big Ball of Mud.
>
> Finally, there is a very pragmatic reason to like perl: It will save your ass. Those who are fluent enough in the culture to realize that "this problem has been solved before," will be able to invoke forces through perl. Again, similar to the numerologists, with a few arcane symbols that are undecipherable to the outside world, great acts of magik can be accomplished.
>
> *— Da Mystik Homeboy*

Espe is a San Francisco hacker who is clearly fond of Python, an open-source computer language. Originally created by a Dutch programmer as a teaching language, Python is now a thriving open-source project. The language's distinguishing feature (both aesthetic and technical) is its strict technical parameters that require bold syntactic clarity. For example,

Python is unusual among programming languages in that the amount of space used to indent a line of code actually affects the code's meaning. On his blog (excerpted above), Espe explains how he was able to hack to his heart's delight for no other reason than to experience "the joy of programming." His stance toward Python is reverent, rooted in deep pleasure. He obviously adores both the formal structure—Python—and the substance—coffee—that have enabled him to hack for his own enjoyment and self-development. In this instance, Espe constructs programming as a pleasing, unencumbered exercise of ample creativity. He seeks in hacking to reach the elusive quality of perfection.

By the next paragraph, however, his register shifts to one of dismayed irreverence toward another programming language, Perl, considered by many to be the antithesis of Python, and therefore a source of antipathy for many Python fanatics. Eventually forced to hack for money (a problem itself for this programmer), he was handed "a worst-case scenario." Poorly coded Perl transformed programming from an activity of boundless satisfaction into a nightmarish ordeal. Espe describes this unfavorable turn of events as being plucked from his "high tower of control and purity," only to be "thrown into a bubbling pool of vaguery and confusion." In having to read and parse other people's codes, programmers routinely encounter what has been depicted aptly as a "twisting maze of corridors, a bottomless pit" (Ullman 2003, 262).

In the second extract, we have DMH, also a San Francisco hacker, but unlike Espe, a self-styled Perl alchemist. Perl's creator, a linguist and programmer named Larry Wall, intended the code to embody the flexible and often-irrational properties of a natural language. As noted by DMH, Perl's aesthetic and technical features are opaqueness, complexity, and flexibility. Also run as an open-source project, Perl is incorporated into the identity of many of its supporters, who call themselves Perl Monks, underscoring the single-minded dedication they have for what is considered a language that can produce poetic (or highly unreadable code) that is creatively displayed during obfuscated code contests, which are usually held for Perl, C, and C++.[3]

While DMH respects Perl for what it is most famous for—its cryptic nature and poetic elegance—he is drawn to Perl for pragmatic reasons. Its "implicit variables, the open object model, the terse expressions," DMH says, allow him to hack on the "Big Ball of Mud"—that is, the world of thick, unmanageable problems and constraints. For DMH, Perl's appeal lies in its extensive common stock of shared solutions and architectural flexibility, which he contrasts to Python, a language so "anal" it is unable to accomplish "great acts of magik." By this he means what is known among Perl geeks as the Perl's motto: "TIMTOWTDI" (There's more than one way to do it).

Digital computers allow for the creation and use of *mini-machines* (aka software) written by programmers using any number of computer languages.

Instead of having to build a piece of hardware for every type of desired function (like a calculator, music recorder, or word processor), the computer is a general-purpose machine that once animated by software programs, can potentially behave as all those functional objects. Espe captures the expansive technical capability of software when he defines coding as "the ability to conjure these giant structures, manipulate them at will, have them contain and be contained by one another." This is computing in its dimension of unfettered freedom.

If at one level hackers adroitly exploit the expansive technical capabilities of the computer, they are also significantly limited by a powerful force field of constraint—the Big Ball of Mud that DMH refers to in his tract on Perl. Constraints are constant and of a nearly infinite variety, such as hardware specifications and failures, computer language syntax, "clueless" managers, inherited "crufty" or vague code, spam, incompatible file formats, "dumb" patent laws, misguided customers, technical specifications, and manager-dictated deadlines. Problems are so central to software that some have even portrayed "glitches" as the "manifestation of genuine software aesthetic" (Goriunova and Shulgin 2008, 111).

Programming thus entails an expansive form of exploration and production that unfolds into a labyrinthine landscape of intricate barriers and problems. Julian Dibbell (2006, 104; see also Ensmenger 2010, 3) depicts the nature of computing, quite poetically, as an "endlessly repeatable collusion of freedom and determinism—the warp and woof of fixed rules and free play, of running code and variable input." Because of constraints and the complexity of coding, to hack up solutions effectively, as Michael Fischer (1999, 261) notes, requires "a constant need for translation, interfacing, sharing, and updating."

As part of this practical capacity, the very nature of hacking—turning a system against itself—is the process of using existing code, comments, and technology for more than what their original authors intended. This is the paradox of constraint. Since many technical objects are simultaneously bound by certain limits yet exhibit potential excesses (Star and Griesemer 1998), during the course of their existence, they can be exploited and redirected toward new paths of functionality by acts of hacking. Hackers are thus attuned not simply to the workings of technology but also seek such an intimate understanding of technology's capabilities and constraints that they are positioned to redirect it to some new, largely unforeseen plane. They collectively and individually derive pleasure in outwitting constraint. In essence, while hacking follows a craftlike practice, it is predicated on a stance of craftiness to move the craft forward. Hacking is where craft and craftiness converge.

Programming and similar technical activities require extremely rigorous logical skills, an unwavering sensitivity to detail (a single wrong character can render a program useless), and such an intimate command of a system that one can, if need be, exceed the conventional or intended constraints of

the system. It requires, in the words of programmer Ellen Ullman (2003, 177), a "relentless formalism." Given the accelerated pace of technological change, hackers also have to perpetually learn new technologies as old ones are phased out due to obsolescence, in order to remain competitive in a marketplace.

Out of this routine form of technical activity hackers have constituted an expansive pragmatic practice of instrumental yet playful experimentation and production. In these activities the lines between play, exploration, pedagogy, and work are rarely rigidly drawn. Sometimes hackers will be motivated by a work-oriented goal, as is/was the case with DMH. At other times, they are motivated to hack for the sheer pleasure of doing so, as Espe emphasized. In either case, frustration *and* pleasure are fundamental to hacking.

A lifetime of creative and pleasurable technical production that often depends on computers also blurs the line between selves and objects. As famously phrased by Sherry Turkle (1984), computers are a hacker's "second self." The hacker relationship to computers and software, though, rarely exists in a steady state in which the self unproblematically melds with this object to catapult hackers into a posthuman, postmodern state of being. The hacker relationship with the computer is a far more finicky, prickly, and interesting affair in which computers themselves constantly misbehave and break down (as do the hackers, at times, when they burn out from such an intense and demanding craft). Hackers sometimes confront their computers as an unproblematic and beloved "object," and at other times view them as an independent and recalcitrant "thing"—a differentiation posed by Heidegger ([1927] 2008) in his famous exploration of things and objects.

In Heidegger's cartography, an object strikes its users as familiar and beyond the scope of critical awareness. Its social meaning is held in place through regular patterns of use and circulation. But when we misuse an object (a spoon used as a knife or a can opener utilized as a hammer) or when an object malfunctions, its thingness is laid bare in the sense that its material characteristic becomes evident. As noted by scholar of things and stuff Bill Brown (2001, 4), "the story of objects asserting themselves as things is the story of how the thing really names less an object than a particular subject-object relation."

In order to appreciate the hacker relationship to computers, this subtle differentiation between an object and a thing is crucial. Hacker technical practices never enact a singular subject-object relation, but instead one that shifts depending on the context and activity. There are times when hackers *work with* computers, and in other cases they *work on* them. Much of hacker technical practice can be described as an attempt to contain the thingness of computers that arises through constant problems and constraints by transforming it back into a pacified, peaceful object that then becomes an ideal vehicle for technical production as well as creative expression. At times, their labor is characterized by grinding effort, and in other

instances, it involves far more pleasurable streams of seemingly friction-free work. The "Python versus Perl Wars" above articulates the metapragmatic understandings of hacker labor that makes it possible to enter into this relational oscillation in the first place.

Hacker Cleverness

> Humor can be dissected, as a frog can, but the thing dies in the process and the innards are discouraging to any but the pure scientific mind.
> —E. B. White, *A Subtreasury of American Humor*

As the examples provided by Espe and DMH display, hacker technical practice is rooted in a playful, analytic, and especially reflective stance toward form that switches between reverence and irreverence depending on individual preferences as well as the context of activity. Hackers routinely engage in a lively oscillation of respect and disrespect for form, often expressed in arguments over the technical idiosyncrasies, strengths, and weaknesses of a programming language, OS, or text editor. These disagreements are the subject of a range of humorously formulated "holy wars," such as Perl versus Python (which we just got a glimpse of), vi versus Emacs (text editors), and Berkeley Software Distribution versus Linux (different Unix-based OS). Despite this, hackers otherwise share an ideal about how labor and production should proceed: with remarkable craftiness and wit.

One important vehicle for expressing wit is humor. As Mary Douglas (1975, 96) famously theorized, joking brings together "disparate elements in such a way that one accepted pattern is challenged by the appearance of another," and can be generally defined as "play upon form." Before expanding on the role of humor among hackers, it is key to highlight that hackers are able to joke with such facility because of the habituated dispositions (Bourdieu 1977) of thought along with tacit knowledge (Polanyi 1966) acquired through a lifelong and routine practice of logic-oriented problem solving. Hackers liberally enjoy hacking almost anything, and because their cultivated technical practice requires an awareness and rearrangement of form, they are able to easily transfer embodied mental dispositions into other arenas. To put it bluntly, because hackers have spent years, possibly decades, working to outsmart various technical constraints, they are also good at joking. Humor requires a similarly irreverent, frequently ironic stance toward language, social conventions, and stereotypes (Douglas 1975).

The mastery and craft of hacking, however, do not fully account for the *craftiness of hackers*.[4] Many of the engineering arts and sciences are guided by similar aesthetic-solving sensibilities, mandates, and preoccupations (Galison 1997; see also Jones and Galison 1998). Engineers and other

craftspeople, such as repairpersons, also deploy similar problem-solving skills rooted in tinkering: they must engage with the limits, possibilities, and constraints of various material objects, and fiddle around to find a nonobvious solution (Orr 1996; Sennett 2008).

Hacker aesthetics share these above-mentioned dispositions, but differ in that hackers see ingenuity and cleverness, often expressed though humor, as far more than a means to regiment and guide technological innovation.[5] Among hackers, humor has a substantial life of its own. Hackers value craftiness and cleverness for their own sake. Whereas academic scientists tend to value referential cleverness as it concerns their work, hackers value cleverness as self-productive, and thus make it appropriate to nearly any context (mathematicians, though, are well known for their prolific humor that exceeds their discipline). Hackers idealize cleverness as a characteristic par excellence that transforms what they spend all of their time doing—creating technology and fixing problems in a great maelstrom of complexity and confusion—into an activity of shared and especially sensual pleasure.

Before extending my theoretical discussion on cleverness and humor, permit me to provide a few examples that are embedded in technical artifacts and one that arose during social interaction. Since much of hacker wit is so technically coded, it is difficult to translate it in any meaningful manner to a lay audience, and I am afraid it might not strike nongeek readers as all that humorous. Analyzing humor, after the fact, is also nearly never humorous, but hopefully it can still be analytically illuminating. I have chosen four examples that are more accessible to a nontechnical audience and supply at least a taste of the types of jokes common among hackers.

Peppering technical artifacts with clever quips occurs quite commonly in hacker technical naming conventions or documentation. For instance, most software applications also come with some sort of description of their purpose and functionality. Jaime Zawinski, the author of a software application called BBDB, portrays his creation via a smattering of jokes (most software applications include a description of their functionality):

> **BBDB** is a rolodex-like database program for GNU Emacs. **BBDB** stands for **Insidious Big Brother Database**, and is not, repeat, ***not*** an obscure reference to the Buck Rogers TV series.
> It provides the following features:
> Integration with mail and news readers, with little or no interaction by the user:
> easy (or automatic) display of the record corresponding to the sender of the current message; automatic creation of records based on the contents of the current message; [. . .]

While the "Insidious Big Brother Database" is an obvious and playful recognition of the common hacker mistrust of governmental authority, the Roger's reference is more esoteric and thus only a small fraction of hackers will

be able to decipher it: those hackers who have watched the television series. With the cue offered in the documentation, those hackers will immediately catch the author's irony (that *this is* a reference to the show) and recognize that BBDB refers to the series' pint-size robot Twiki, whose preferred mode of communicating is a noise that sounds remarkably like "B-D-BBBB-D."

I am particularly fond of the next example contained in the manual (usually shortened to "man page") for Mutt, a popular email client among geeks. Man pages provide documentation and are included with almost all Unix systems. They typically follow a strict standard for conveying information about the program by designating a set of common categories under which programmers provide detailed information about the software, such as the name, synopsis, description, options, files, examples, and authors. One important category is bugs, where authors list the problems and glitches with the software. (Software can have a number of bugs and glitches yet still work. The bug category gives you a sense of what these glitches are and when they will emerge.) The Mutt man page exploits the fact that the word mutt can mean a mongrel dog. Notice the category of bugs:

NAME
mutt—The Mutt Mail User Agent

SYNOPSIS
mutt [-nRyzZ] [-e cmd] [-F file] [-m type] [-f file] [. . .]

DESCRIPTION
Mutt is a small but very powerful text based program for reading electronic mail under Unix operating systems, including support color terminals, MIME, and a threaded sorting mode.

OPTIONS
—A alias
An expanded version of the given alias is passed to stdout.
—a file
Attach a file to your message using MIME. [. . .]

BUGS
None. Mutts have fleas, not bugs.

FLEAS
Suspend/resume while editing a file with an external editor does not work under SunOS 4.x if you use the curses lib in /usr/5lib. It does work with the S-Lang library, however. Resizing the screen while using an external pager causes Mutt to go haywire on some systems. [. . .]

My last example of this subtle integration of wit in a technological artifact comes in the form of a warning message. Many software programs and related artifacts are accompanied by dramatic warnings that appear during configuration. These are intended to alert the user that its integration into some software systems may produce unanticipated, drastic, and completely undesirable results (like breaking multiple parts of your software system that took five weeks to get "just right"). Often this happens because a piece of software is still experimental and riddled with bugs. The following help message is available in the 2.6 branch of Linux kernel configuration and refers to the RAID-6 device driver, which at the time was still under development and hence buggy:

> WARNING: RAID-6 is currently highly experimental. If you use it, there is no guarantee whatsoever that it won't destroy your data, eat your disk drives, insult your mother, or re-appoint George W. Bush

These three examples demonstrate that hackers value subtlety and irony of presentation. Hackers discretely embed nuanced, clever and frequently nonfunctional jokes within what are otherwise completely rational, conventional statements of function. Yet hackers never use jokes to undermine the functionality or trustworthiness of the code or documentation. These technical artifacts are judged seriously by geeks. The presence of wit only works to add to the value of the rational content by reminding the user that behind these highly systematized genres, there is a discriminating and creative individual.

Other instances of hacker wit occur in person and are less subtle. For example, at a security conference in 2001, Peiter Zatko, aka "Mudge," a computer security researcher, professional, and hacker (once part of the famous hacker association L0pht Heavy Industries), arrived in a terrycloth bathrobe to present on a panel on PDAs. This bold sartorial statement distinguished him from his nonhacker colleagues, also security researchers, but scientists. It prioritized hacker over scientific identity. Mudge's attire, however, performed a problematic public-private breach in the context of his talk, which focused on the changing use patterns of PDAs. "PDAs were designed for personal use, but are now being used more for business," Zatko said. "There's a security boundary that's being crossed."[6] Zatko's robe embodied his argument that the shift amounted to a breached security boundary: PDAs should not be used for sensitive, private data.

Though humor is found worldwide, instances like the ones just described are fruitful to the anthropologist because of their cultural particularity. As this playful practice usually induces laughter—a state of bodily affect that enraptures an audience—humor can potentially produce forms of collective awareness and shared sociality. Given these two properties, we can define humor, in the most general terms, as a play with form whose social force lies

in its ability to accentuate the performer, and which at times can work to delineate in-group membership.

Apart from this, the meaning of humor is otherwise quite culturally specific. The power to enrapture and entangle people can lead to entirely contrary social effects. In certain cases and types of groups, joking can establish and maintain hierarchies as well as social boundaries by, say, delineating social roles (Gusterson 1998; Mulkay 1988; Radcliffe-Brown 1952). In other cultural and historical contexts, humor pushes the envelope of conceptual boundaries in ways that may be fleeting and frivolous (Douglas 1975), or politically subversive (Bakhtin 1984; Critchley 2002). In other words, because the effect, purpose, and even form of humor are deeply context dependent, culturally inflected, and historically moored, it is a useful tool for analyzing broader forms of cultural meaning.

Among hackers, humor is a distilled and parsimonious instantiation of the adoration of cleverness. It is an especially effective way of enacting hackers' commitment to wittiness precisely because, unlike the objects of hacker technical production, joking has no strict functional utility, and speaks to the inherent appeal of creativity and cleverness for their own sake. Joking is a self-referential exercise that designates the joker as an intelligent person and cleverness as autonomously valuable.

It bears repetition that hackers draw on their pragmatic ability to manipulate form to engage in this type of joking. These two elements—being good at hacking and valuing cleverness for its own sake—exist in a tight and productive symbiosis, a mutually reinforcing relation that produces an abundance of humor among hackers. There is a close kinship between hacking and humor.

Insofar as humor is tethered to the moment of its utterance, it exudes an auric quality of spontaneous originality (Benjamin [1936] 2005), which among hackers authenticates the self as a distinctive and autonomous individual. Humor is one of the starkest expressions of the hacker "ideal self." By telling jokes, hackers externalize what they see as their intelligence and gain recognition from technically talented peers.

Like hacker technological production, humor also works to implicitly confirm the relational self who is joined to others by a shared domain of practice, and a common stock of implicit cultural and explicit technical knowledge. Recall that many jokes, such as technical Easter eggs, are received as pleasurable gifts. They not only break the monotony and grind of sitting at the computer, usually for hours a day as one churns out code or resolves problems, but also remind hackers of their shared experiences. "One might say that the simple telling of a joke," writes philosopher Simon Critchley (2002, 18), "recalls us to what is shared in our everyday practices. [. . .] So, humor reveals the depth of what we share." If humor creates fine distinctions, it also levels the ground, because in the very moments of laughter, hackers implicitly recognize and celebrate the shared world of meaning in which they work. After all, like many instances of joking, much of hacker

humor is so culturally coded (which here means technically inflected) that the only people who can routinely receive, and as such appreciate, their wit are other hackers. One must rely on the acknowledgment and judgment of those who can appreciate the performance of wit, because they share at least some of one's implicit values, explicit technical knowledge, and standards of creative evaluation.

To the extent that everyone enjoys laughter, humor functions much as a communal gift—the performance of which beckons others to follow suit. Indeed, once one hacker starts joking, many others will dive in. It also breaks the monotony and eases the strains of hacking, and so can also be seen as a mechanism to preserve hackers' humanity (and sanity) in the face of the merciless rationale of the machine they engage with everyday. When humor is woven into the actual code or technical artifacts animating the machine, it brings otherwise-mechanic language directly and unmistakably into the realm of human communication.[7] Once part of the apparatus of human communication, humor powerfully confirms a shared mode of being in the world; in other words, it affirms a lifeworld. The very expression of humor is seen as proof that despite their physical dispersion and sense of independence, hackers nonetheless cohabit a shared social terrain built around a lifelong intimacy with technology and technical thinking—one they have come to celebrate.

Among hackers, humor functions in multiple capacities and undoubtedly reflects the value they place on productive autonomy as well as the drive to perform cleverness. Much of their humor is ironic—a play with form. Its purpose is to arrive on the scene of the joke (often a technical object) unexpectedly. This is also the ideal nature of a great hack, insofar as it should surprise other hackers into a stance of awe. Humor, as Douglas (1975, 96) reminds us, is "a play upon form that affords an opportunity for realizing that an accepted pattern has no necessity." This definition bears a striking resemblance to the pragmatics of hacking; hackers are constantly playing on form, revealing that there is no single solution to a technical problem. And although hackers claim it is abominable to reinvent the wheel, in practice, they are constantly doing so as they follow their own creative instincts and visions.

In its ability to concurrently accentuate inclusiveness and exclusiveness, and make and level hierarchies, humor shapes conventions of sociality, ideals of creativity, and hackers' attitudes toward one another and outsiders. Now let's take a closer look at the tension between individuality and collectivism to which humor so delectably points us.

Communal Populism and Individual Elitism

If hacker pragmatics oscillate between a respect and disrespect for form, hacker sociality alternates between communal populism and individual elitism. Largely by way of F/OSS philosophy, hackers laud mutual aid and

cooperative reciprocity as vital features of technical collaboration. They spend an inordinate number of hours helping each other. But there is also an elitist stance that places an extremely high premium on self-reliance, individual achievement, and meritocracy.[8] While the populist stance affirms the equal worth of everyone who contributes to an endeavor, the elitist one distributes credit, rewarding on the basis of superior accomplishment, technical prowess, and individual talent—all judged meticulously by other hackers. Hackers will spend hours helping each other, working closely together through some problem. Yet they also engage in agonistic practices of technical jousting and boasting with peers, and in turn, this works to create hierarchies of difference among this fraternal order of "elite wizards." Ullman (1997, 101) condenses this tension into few words: "Humility is as mandatory as arrogance." The line between elitism and populism is not simply an intellectual afterthought posed by me, the anthropologist, but also a living, relevant, affective reality discussed and dissected by hackers.

This duality arises during the course of their work, and is openly discussed in ethical and pragmatic terms. On the one hand, hackers speak of the importance of learning from others and construe knowledge production as a collective enterprise—and this rhetoric is frequently matched in practice by truly generous and copious acts of sharing. In any given minute of the day, I can log into one of the developers' IRC channels, and there will be some developers asking a question, getting an answer, and giving thanks, as this example illustrates:

<zugschlus> does anybody know how to configure sound in KDE4? [KDE is a desktop environment.]
<pusling> Zugschlus: in systemsettings
<zugschlus> pusling: applications => settings?
<kibi> but AFAICT ["as far as I can tell"], what you have in svn helped me build various thingies against libqt4-dev and friends.
<pusling> Zugschlus: computer > s-ystemsettingns
<pusling> KiBi: I think qt4 is now waiting in new.
<zugschlus> pusling: that part only has home, network, root and trash.
<kibi> pusling: oh, ok :(
<pusling> Zugschlus: do you have the package systemsettings installed?
<pusling> KiBi: so if you have special contacts to ftp team, feel free to use them.
<kibi> pusling: yep, seen it.
* kibi can try
<kibi> Ganneff: mhy: ^^^ if you want to help kfreebsd-* folks get more packages built, fast-tracking qt4-x11 would really be great. Thanks for considering. :)
<zugschlus> pusling: no, that was missing. thanks.
<pusling> Zugschlus: you then probably want to make sure you have kde-minimal installed.

Guiding this practice is the idea that the free software project represents an endeavor that far exceeds any single person's efforts, and thus everyone's contribution is highly regarded, whether it involves filing a bug report or offering a significant, large-scale innovation.

On the other hand, hackers often express a commitment to self-reliance, which can be at times displayed in a quite abrasive and elitist tone. The most famous token of this stance is the short quip "Read the Fucking Manual" (RTFM). It is worth noting that accusations or RTFM replies are rarer than instances of copious sharing. Let me provide two examples of RTFM in action. In the first, "Error" drops into a new channel after asking a question in the #perl channel, where he got a prompt RTFM, after which everyone else went back to discussing the band Metallica. In this channel, they did not offer an RTFM but instead suggested going to the #metallica channel, which in this case, is a joke [IRC channels are designated by #name-of-channel].

<813-error> i ask a question in #perl and get RTFM and they go back to talking about metallica [. . .]
<813-error> d Match digit character <that would be numbers right?
* C4 knows nothing of perl
<modem> same here :/
<modem> ask in #metallica

The second example does not contain a joke but rather only a rebuke in the form of RTFM:

<karsten> Ace2016: alsamixer / aumix are interactive ncurses programs
<ace2016> so?
<karsten> Ace2016: You may be able to steer 'em w/ stdin as well.
<ace2016> can't they accept a command like aumix—volume decrease 10% or something like that?
<karsten> Ace2016: RTFM
<karsten> Ace2016: Which is to say, I don't know. Go look yourself.

These two poles of value reflect pervasive features of hacker social and technical production as it unfolds in everyday life. It only takes a few days of following hacker technical discussion to realize that many of their conversations, whether virtual or in person, are astonishingly long question-and-answer sessions. To manage the complexity of the technological landscape, hackers turn to fellow hackers (along with manuals, books, mailing lists, documentation, and search engines) for constant information, guidance, and help. Unlike academics—who at times religiously guard their data or findings until published, or only circulate them among a small group of trusted peers—hackers freely share their findings, insights, and solutions. More than ever, and especially in the context of free software projects, hackers see their productive mutual aid as the underlying living credo driving free software philosophy, and the methodology of collaboration and openness. Hackers

Chapter 3: The Craft and Craftiness of Hacking

1. Here is a little more information about the code. The "tr" in this code is a function that translates all occurrences of the search characters listed, with the corresponding replacement character list. In this case, the slash character delimits the search list, so the list of what to search for is the asterisk character. The replacement list is the second asterisk character, so overall it is replacing the asterisk with an asterisk. The side effect of this code is that the "tr" function returns the number of search and replaces performed, such that by replacing all the asterisks in the variable $sky, with asterisks, the variable $cnt gets assigned the number of search and replaces that happen, resulting in a count of the number of stars in the $sky. What follows after the # symbol is a comment, a nonfunctional operator found in most programs, theoretically supposed to explain what the code does.
2. These were once blog entries and no longer exist. These texts are on file with the author. Python and Perl are computer languages.
3. The entries are judged on aesthetics, output, and incomprehensibility, and are only decipherable by the most accomplished of Perl experts, but can undoubtedly be aesthetically admired by all as a postmodern object of utter incomprehension and amusement. For an insightful discussion of obfuscation in code, see Monfort 2008.
4. In his engrossing ethnography, Graham Jones (2011) covers the way in which cunning, cleverness, and inventiveness are learned, performed, valued, and embodied among the magicians that he worked with in Paris.
5. For a discussion of some of the tensions in the corporate world that arose due to the perception of programmers as clever and idiosyncratic, and an excellent history of programmers, see Ensmenger 2010, especially chapter 3.
6. http://www.ingen.mb.ca/cgi-bin/news.pl?action=600&id=10383 (accessed November 20, 2007).
7. I would like to thank Jonah Bossewitch, who pushed me to think about humor in light of the rationality of the computer more deeply.
8. Some notable examples of populist formulations are *Computer Lib* by Ted Nelson (1974) and Stallman's "GNU Manifesto." For examples of the elitist manifestation, see Levy 1984; Sterling 1992; Borsook 2000.
9. http://osdir.com/ml/linux.debian.devel.mentors/2003-03/msg00272.html (accessed July 5, 2009).
10. http://osdir.com/ml/linux.debian.devel.mentors/2003-03/msg00225.html (accessed July 5, 2009).
11. This is quite similar in logic to liberal notions of states of nature that posit forms of individuality outside social relations. An interesting question to further explore is why this view still holds such appeal even though it is most often only conceptualized in these hypothetical terms.
12. http://osdir.com/ml/linux.debian.devel.mentors/2003-03/msg00225.html (accessed July 23, 2010). During interviews, this idea that programming could span the spectrum from unoriginal functionalism to high art came up again and again. For example, one programmer characterized it in the following way: "I think it can be art, but it is not always. [. . .] If I had to pick a comparison, I would pick carpentry because carpentry always has that range. You can start with just making a bookcase or something utilitarian all the way to creating something like creating a piece of art with wood." Developers explained their craft triangulated between math/science, engineering, and

art. Engineering was usually at the apex, respectively tending toward the side of art or science, depending on the idiosyncrasies and preferences of the programmer along with the nature of the project.

13. For instance, it is routine for project developers to thank users or nonmember developers for their contributions. By way of illustration, on the Subversion project, which develops code-tracking software, out of the approximately eighty-seven full and partial committees, fifty-five were thanked by name in a commit log message (that someone else committed) before they became a committee themselves (as of April 25, 2005).
14. Luser is a common intentional misspelling of loser. "A luser is a painfully annoying, stupid, or irritating computer user. The word luser is often synonymous with lamer. In hackish, the word luser takes on a broader meaning, referring to any normal user (i.e. not a guru), especially one who is also a loser (luser and loser are pronounced the same). Also interpreted as a layman user as opposed to power user or administrator" (http://en.wikipedia.org/wiki/Luser [accessed September 9, 2011]).
15. http://www.thinkgeek.com/tshirts/frustrations/3239/ (accessed March 21, 2006).
16. http://lists.debian.org/debian-vote/2005/03/msg00610.html (accessed July 5, 2009).
17. http://www.mail-archive.com/debian-vote@lists.debian.org/msg08500.html (accessed July 17, 2010).
18. http://svn.red-bean.com/repos/kfogel/trunk/.emacs (accessed July 5, 2009).
19. http://evans-experientialism.freewebspace.com/barthes06.htm (accessed September 17, 2011).
20. I would like to thank Martin Langhoff, who suggested the name palimpsest for the authorial tracking that occurs on these version control systems.
21. Those hackers who use Berkeley Software Distribution licenses place more value on "freedom of choice" than necessarily recursively feeding modified code back into the community of hackers. I would still like to point out, however, that by using a Berkeley Software Distribution license, a hacker has still made a deliberate choice to keep their code open and accessible to others. The difference is that the license does not mandate this choice for others and thus adheres to a more negative/libertarian notion of liberty than that of Mill's.

Chapter 4: Two Ethical Moments in Debian

1. A prolific literature in the sociology and anthropology of science fruitfully dissects how professional identities along with ethical commitments are established during periods of training (Good 1994), vocational practice (Gusterson 1998; Luhrmann 2001; Rabinow 1996), and are sustained by the coded and metaphoric language of professions that work to elide ethical concerns (Cohn 1987). All these works have pushed me to think about how ethical commitments are forged by a range of micropractices, many of them narrative based.
2. For the history and working of consensus among Internet engineers, see Kelty 2008; Gitelman 2006; DeNardis 2009.
3. For an analysis of similar dynamics among programmers, see Helmreich 1998; Levy 2011.
4. The analysis of trust in the context of digital media interactions has so far been sporadic, but it is starting to gain momentum. For an edited collection exclusively dedicated

maintain that this mode of production is responsible for better hackers and better technology.

Alongside technical question-and-answer sessions, developers dissect the ethics of their labor. For example, on a Debian mentors' mailing list discussion, one aspiring hacker asked, "How did you get from the middle ground to guru-dom?? Or is the answer that if I need to ask, I will never be a hacker!!??" A developer known for his humility and prolific contributions to the Debian project offered a lengthy response—a small section of which I quote below. In highlighting the importance of sharing, learning for others, and even coding for others, he affirms a populist stance, commonly expressed by many Debian developers:

> One other inspiration for me has been helping people. Though this has been spottier than I could hope, I do from time to time end up doing some program entirely because I can see other people need it. This tends to broaden experience a lot. Things like writing programs for an unfamiliar platform (microsoft), in a unfamiliar language (spanish), and needing to work closely with the people who would use it, cannot help but change how you look at things. My most valuable experiences in this area have been when I had direct contact with the people who would be using the program, rather than just noticing a hole and deciding I would try to go fill it like you did.[9]

Here he accords weight to pedagogy and collective interdependence in which learning from and even coding for others is a crucial component of technical progress as well as self-development.

During this discussion, though, other developers stressed the importance of independence by urging the questioner to follow his own particular interests necessary to cultivate technical independence. For example, one developer offered the following advice:

> I think you made two mistakes. [. . .] The first is looking to other people for problems to be solved. You'll never find the inspiration in solving problems that don't affect you. Since you don't feel the itch, you don't get much satisfaction from the scratch. Speaking for myself, I picked up a programming manual for my first computer and started reading; well before I was finished, I had two dozen ideas for programs to write. Those programs and their spinoffs kept me busy for a couple of years, and I loved it. Second, when an itch hits you, don't research to see if someone has already solved the problem. Solve it yourself. Mathematical texts aren't filled with answers right beside the problems; they teach you by making you work out the answers yourself.[10]

Simply in marking the question as misguided (because he looks to other people for problems to be solved), this developer asserts the value of self-determination. The original question violated what is the predominant

(though not unquestioned) norm of self-sufficiency among developers—a norm that captures the isolated and individualistic phenomenology of much of their labor, which for many hackers commenced in childhood.

One developer, in answering a question I had about the significance of free software, expressed this stance of technical self-determination and independence in the following terms: "If I am cut off from the world, then in theory then I can maintain my own domain over software. I don't have to depend on anyone else; I can do it all myself. If my computing environment diverges from everyone else's in the world, I can still keep on going." This commitment to a fully autonomous, sovereign self who shuns any obvious signs of dependence on others is a common trait among developers. Given this mode of laboring, it is not surprising that hackers place so much emphasis on autonomy and self-sufficiency—qualities that are congenial to many hackers as they resonate so strongly with the very experience of intense periods of isolated labor.

Yet this statement of independence is based on a hypothetical scenario of being "cut off from the world"—something even this developer qualifies as unlikely.[11] In most practical instances, hackers are constantly plugged in, connected through various technical structures of communication. They work together as well as in complete isolation, for personal and joint public projects. Software theorist Matthew Fuller (2008, 5) describes how the freedom of coding gets subsumed by a host of conditions that always lay outside code proper: "Computation establishes a toy world in conformity with its axioms, but at the same time, when it becomes software, it must, by and large [. . .] come into combination with what lies outside of code."

Generally, the need to both work alone and with others is experienced free of contradiction, because the two needs are complementary and readily recognized as such by most hackers. To take another example from the mailing list discussion on what transforms a mediocre hacker into a great one, a developer captured this duality by describing how hacking tacks between two productive extremes—the collaborative and individual—that are not mutually exclusive:

> Creating a linux distribution is a group activity, but creating art is fundamentally a solitary, private experience. Turn off your internet connection; sit in a dark room, with nothing but the glow of a monitor, the warmth and hum of your computer, and the ideas will flow: Sometimes a trickle, sometimes a torrent.[12]

These two modes can clash, however. This is powerfully signaled through a form of stylized boasting that contrasts one's intelligence with the idiocy of "mere users" of software. While users of free software are often lauded as essential participants in the broader project of technical development because they provide insightful queries and bug reports (and also are seen as possible future hackers), at other times they are deemed second-class

technical citizens.[13] This designation is frequently accomplished through the only way in which socially uncomfortable topics can be routinely discussed: by joking. On developer IRC channels, hackers playfully mock users. By complaining about stupid questions and queries, hackers depict users as less worthy contributors for lack of technical proficiency, or may display their complaints elsewhere, such as including humorous email signatures that taunt the wider universe of (l)users.[14] This condescending attitude is aptly and humorously conveyed in the following quote from a developers' email signature, originally formulated by Richard Cook: "Programming today is a race between software engineers striving to build bigger and better idiot-proof programs, and the Universe trying to produce bigger and better idiots. So far, the Universe is winning."

Users, though, are by no means the only type of persons subject to the humorous or more vitriolic accusation of technical incompetence. If a question is posed in the wrong register, is seen as uninteresting, or the answer can easily be found elsewhere, nearly anyone from a mere user to a "skilled" developer can receive the stylized and semihumorous RTFM rebuff. Stated on a hacker site with vivacious bite:

> [RTFM] is a big chromatic dragon with bloodshot beady eyes and fangs the size of oars. RTFM is me screaming at you as fireballs come out of my mouth to get off your precious no-good tush, march down to the local bookstore or MAN page repository, and get the eff off my back because I'm trying very hard to get some freakin' work done. Jeez.[15]

If you are better informed with the knowledge that there is "NO MANUAL," you can quickly defend your honor (i.e., intelligence) by pointing this out and gain substantial respect if you take it on yourself to write documentation. Otherwise, you will have to swallow the rebuke, google for the information, and hope for a better response next time (or simply find another IRC channel and ask elsewhere).

A complicated set of norms and conventions surround asking for help. They depend on the social context of the query and who is asking the question. For example, once someone has garnered a certain amount of trust and respect, they can usually get away with asking what is seen as a nonchallenging, uninteresting question. Developers who have not yet established trust will frequently get immediate help if the question is seen to be a challenge, but a basic questions will raise immediate eyebrows, especially among strangers or members who are technically unvetted, and therefore must maneuver with more caution and tact.

RTFM is a comedic, though stern, form of social discipline. It pushes other hackers to learn and code for themselves as well as affirms that effort has been put into documentation—an accessible form of information that benefits the group—but in a way that still requires independent learning. Many users and developers complain of the lack of adequate documentation for

free software, faulting the tendency of some developers to exist in technical silos, "selfishly" coding only for themselves, and not attending to the needs of other users and developers by writing technically boring but necessary documentation. Many developers also note how the lack of extensive documentation can hinder collaborative technical work. Thus, when someone asks for information that in fact does exist in documentation, they often receive the RTFM rebuke, whose subtext says, "go learn for yourself, especially since others have already put in the work (i.e., documentation) to make this happen." To give too much aid is to deny the conditions necessary for self-cultivation.

The use of RTFM is disputed as well. During the 2005 Debian project leader election, the issue of documentation erupted during a mailing list debate. The subject of RTFM rebukes was broached directly. One developer argued that RTFM is an inflammatory, unproductive response to newcomers who may find themselves confused and overwhelmed with Debian's technical and procedural complexity. To make new users feel welcome, he believed that developers should refrain from replying with RTFM, and instead focus their efforts on achieving greater transparency and accessibility. While debating a Debian project leader candidate who had been with the project for years, he conveyed this commitment to corporate populism when he stated:

> You know a lot about the project [and its project internals], so it's all obvious to you. There are people among us who have not been part of Debian since 1.1, but who would like to know more about what's happening behind the curtains. However, those people are often told to RTFM or go spend time in the code, or just not taken seriously.[16]

In response, the Debian project leader candidate defended the general use of RTFM, concisely enunciating the value of self-determination:

> When the code is public, rtfm is the proper answer. One might add "document it properly afterwards" as well, though. When the data is available as well, that's best. Some data cannot be made available for legal or other binding obligations (new queue, security archive). If you feel that some bits are missing and need to be documented better, point them out and get them documented better, maybe by doing it on your own. I know a lot about the project because I've been involved in many parts. Other developers are involved in many parts as well. Some other developers mostly whine about not being involved without trying to understand. *sigh*[17]

In other words, if the requested information is public, it is incumbent on the developer to seek it out, and if unsatisfied with the current state of accessibility, then the next logical step is to make it happen—by yourself. If one does, one can display self-determination and self-development, the vehicles by which to gain the respect of accomplished peers on a similarly paved technical path.

If the subject of elitism erupts on mailing list discussions over project organization, a form of stylized boasting, taunting, cajoling, and elitist disdain is also frequently performed through code. Here I provide two examples. And again note how humor is used in both, to some degree working to soften the abrasive tone of these messages.

The first one is written in the style of an "I-can't-believe-how-idiotic-this-problem-I-have-to-solve-is rant" that disparages a bug in the Emacs email reader. Before addressing the significance of his code, permit me defer to the coder, Karl Fogel, to explain the context of the problem and the technical nature of his solution:

> Basically, the mailreader insisted on colorizing my mail composition window, even though I tried every documented method available to ask it not to do that. In desperation, I finally wrote code to go "behind the back" of the mailreader, and fool it into thinking that it had already done the colorization when it actually hadn't.[18]

The comments open with a statement of disbelief; take note of the naming of the variable, which I highlight in bold and italics:

```
;; I cannot believe what I have to do to turn off font locking in mail
   ;; and message buffers. Running "(font-lock-mode -1)" from every
   ;; possibly relevant gnus-*, mail-*, and message-* hook still left
   my
   ;; reply buffers font-locked. Arrrgh.
   ;;
   ;; So the code below fools font-lock-mode into thinking the buffer
   is
   ;; already fontified (so it will do nothing—see
   ;; font-lock.el:font-lock-mode for details), and then makes sure
   that
   ;; the very last thing run when I hit reply to a message is to turn
   ;; off font-lock-mode in that buffer, from post-command-hook.
   Then
   ;; that function removes itself from post-command-hook so it's
   not run
   ;; with every command.
      (defun kf-compensate-for-fucking-unbelievable-
      emacs-lossage ()
      (font-lock-mode -1)
      (remove-hook
      'post-command-hook
      'kf-compensate-for-fucking-unbelievable-emacs-lossage))

      (add-hook 'font-lock-mode-hook 'kf-font-lock-mode-hook)
      (defun kf-font-lock-mode-hook ()
```

```
(if (or (eq major-mode 'message-mode)
    (eq major-mode 'mail-mode))
    (progn
    (make-local-variable 'font-lock-fontified)
    (setq font-lock-fontified t)
    (add-hook 'post-command-hook
    'kf-compensate-for-fucking-unbelievable-emacs-
    lossage)
    )))
```

By opening the comments with "I cannot believe what I have to do" and ending with "Arrrgh," he signals the fact that this sort of trite problem is so idiotically banal, it should have never appeared in the *first place*. Fixing it is a waste of his superior mental resources. Lest there be any ambiguity as to what the author really thought about the code, he continues to drive the point home in his rant by naming the variable with an unmistakably deliberate insult: "compensate-for-fucking-unbelievable-emacs-lossage."

During the course of my early research, I was shocked at the disjoint between the in-person real-world "codes of conduct" and the "codes of software conduct." Nothing about this coder's personality, who I got to know very well over the course of five years, would indicate such haughty declarations. There is no need for such an indication because these enunciations are rarely a matter of innate psychology. Instead, these are conventionalized statements by which hackers declare and demarcate their unique contribution to a collective endeavor. They also represent culturally sanctioned mechanisms for judgment.

Fogel's code is an apt example of "face work" (Goffman 1967, 5)—when a hacker is sanctioned to perform a "line," which is the "pattern of verbal and nonverbal acts by which he expresses his view of the situation and through this his evaluation of the participants, especially himself." Within such a presentation, hackers can declare and demarcate their unique contribution to a piece of software while at the same time proffering technical judgment. One may even say that this taunting is their informal version of the academic peer-review process. In this particular case, Fogel is declaring the code he patched as an utter failure of the imagination.

Because these insults are critical evaluations of work, if hackers dare to make such pronouncements, they also have to make them technically clever enough to be accepted as accurate critiques. After a declaration is made, a hacker should be ready to enter the arena of competitive jousting. If one hacker judges some piece of code, it is almost guaranteed that another hacker may reply with chutzpah of their own, often in humorous guise.

The second example demonstrates this type of competitive play of technical volleyball, a form of "antiphony" of "call and response" common to jazz poetics (Gilroy 1993, 78). While jazz poetics may seem strange to apply

to hacking, I will expand on this connection later when addressing hacker notions of creativity. First, let's take a closer look at this portion of the code that shows the use of boasting to induce a response (I have highlighted the relevant section in italics):

```
/* Prime number generation
   Copyright (C) 1994 Free Software Foundation

This program is free software; you can redistribute it and/or
   modify it under the terms of the GNU General Public License as
   published by the Free Software Foundation; either version 2, or
   (at your option) any later version.

This program is distributed in the hope that it will be useful, but
WITHOUT ANY WARRANTY; without even the implied warranty of
   MERCHANTABILITY or FITNESS FOR A PARTICULAR PUR-
   POSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License
   along with this program; if not, write to the Free Software
   Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA. */

#include <stdlib.h>
   #include <string.h>

/* Return the next prime greater than or equal to N. */
   int
   nextprime (int n)
   {
      static int *q;
      static int k = 2;
      static int l = 2;
      int p;
      int *m;
      int i, j;

      /* You are not expected to understand this. */

      if (!q)
         {
         /* Init */
         q = malloc (sizeof (int) * 2);
         q[0] = 2;
         q[1] = 3;
      }
```

Derived from the FSF's Hurd development project, which is its kernel project, the code is a prime number generator. Programmers have told me that the technical details are fairly intricate, so I refrain here from providing an explanation of the actual mechanics of the code, and for the sake of analysis it is not necessary. The important element is the author's comment: "/* You are not expected to understand this. */." It reveals how boasting is an open invitation to engage in technical jousting—a playful taunt that explicitly encourages the technical comeback that proves the expectation wrong.

The author's intentions are pretty clear in the code, but here is his retroactive explanation: "At this point I offered the function as a challenge to Jim Blandy. [. . .] That the function was intended to produce prime numbers was never hidden; the challenge was to explain its technique." Blandy took the call to technical arms and responded with his own exegesis of the algorithm. When the original author of the prime number function updated the code, he changed the taunt to "/* See the comment at the end for an explanation of the algorithm used. */," and at the end of the code, stated, "Jim produced the following brilliant explanation" and included it within the code (and again, I have indicated the relevant section in italics).

```
/* Prime number generation
   C [ . . . ]
   #include <stdlib.h>
   #include <string.h>
   /* Return the next prime greater than or equal to N. */ [ . . . ]
   /* See the comment at the end for an explanation of the algo-
   rithm
   used. */
   if (!q)
       {
       /* Init */ [ . . . ]
   * [This code originally contained the comment "You are
   not expected to understand this" (on the theory that ev-
   ery Unix-like system should have such a comment some-
   where, and now I have to find somewhere else to put it).
   I then offered this function as a challenge to Jim Blandy.
   At that time only the six comments in the function and
   the description at the top were present.
   Jim produced the following brilliant explanation.]
   The static variable q points to a sorted array of the first l natural
   prime numbers. k is the number of elements which have been
   allocated to q, l <= k; we occasionally double k and realloc q ac-
   cordingly to maintain this invariant.
   The table is initialized to contain a few primes (lines 26, 27,
   34-40). Subsequent code assumes the table isn't empty.
```

> When passed a number n, we grow q until it contains a prime
> >= n
> (lines 45-70), do a binary search in q to find the least prime >=n
> (lines 72-84), and return that. [. . .]

If some hackers are ready to pounce on what they deem as the idiocy of others, they are also as likely to dole out recognition where they see fit. Hence, even while hackers are on a path toward *self-development*, this self-fashioning is intimately bound to others, not simply because of a love of tinkering or the dependence derived from collaboration, but because any meritocratic order based on expertise fundamentally requires others for constant evaluation as well. Hackers use the path of humor, taunt, jousting, boasting, and argument for such expressions of technical taste and worthiness, and in the process, cultivate themselves as expert hackers.

Just Freedom

Given hackers' proclivity for expressing cleverness, acknowledgment that they build on the shoulders of giants, need to garner recognition from others, and dual penchant for lauding populist collectivism and individual self-determination, what might these attributes reveal about hacker notions of personhood, creativity, and authorship?

It is not surprising that in so much of the literature, hackers are treated as quintessentially individualistic (Levy 1984; Turkle 1984). "The hacker," Turkle (1984, 229) writes, "is the defender of idiosyncrasy, individuality, genius and the cult of individual." Some authors argue that this individualism is a close variant of a politically suspicious libertarianism (Borsook 2000). Hackers are perpetually keen on asserting their individuality through acts of ingenuity, and thus these statements are unmistakably correct. In most accounts on hackers, however, the meaning of this individualism is treated as an ideological, unsavory cloak or is left underspecified. Why the pronounced performance of individualism? What does it say about how hackers conceptualize authorship? What tensions does it raise?

Because hackers do not automatically treat software as solely derivative of one laboring mind but instead see it as derivate of a collective effort, the constant drive to perform ingenuity reflects the formidable difficulty of claiming discrete inventiveness. After all, much of hacker production is based on a constant reworking of different technical assemblages directed toward new purposes and uses—a form of authorial recombination rarely acknowledged in traditional intellectual property law discourse.

Because of the tendency, especially now more than ever, for hackers to recognize the reality of collaboration, it may seem that they are moving toward the type of politics and ethics of authorship that flatly reject the ideal

of individualism altogether—a rejection famously explored in the works of Roland Barthes, Michel Foucault, and Dick Hebdige. In the F/OSS domain, hackers have not moved, even an inch, to decenter the persona of the author in the manner, say, most famously exemplified by Barthes, who in 1967 sought to dethrone the authority of an author: "To give a text an Author is to impose a limit on that text, to furnish it with a final signified, to close the writing."[19]

Instead, among hackers the authorial figure seems to speak slightly louder, clamoring for and demanding credit and recognition, established through oral histories of software or etched into the infrastructure of production. Hackers record contributions and attributions in common files included with source code, such as the Authors and Contributors files (Yuill 2008). This archival drive helps partially explain why certain hackers can also receive the legendary status they do. This everyday discourse and inscription develops a shared historical awareness about who contributed what—one that brings attention to the conditions of production or the nature of the contribution. Furthermore, accountability and credit are built into many of the technical tools that facilitate collaboration, such as CVS and Subversion—software systems used to manage shared source code. These systems give developers the ability to track (and potentially revert to) incremental changes to files and report the changes to a mailing list as they are made, and are often used concurrently by many developers. Since developers all have accounts, these technologies not only *enable collaboration* but also provide *precise details of attribution*. Over time, this record accumulates into a richly documented palimpsest. Though individual attribution is certainly accorded, these technological palimpsests reflect unmistakably that complicated pieces of software are held in place by a grand collaborative effort that far exceeds any one person's contribution.[20]

In contrast to many accounts on authorship, I find that a short description about the aesthetics of jazz and its "cruel contradiction" is eerily evocative of the hacker creative predicament:

> There is a cruel contradiction implicit in the art form itself. For true jazz is an art of individual assertion within and against the group. Each true jazz moment (as distinct from the uninspired commercial performance) springs from a context in which each artist challenges all the rest, each solo flight, or improvisation, represents (like the successive canvases of a painter) a definition of his identity: as individual, as member of the collectivity, and as link in the chain of tradition. Thus, because jazz finds its very life in an endless improvisation upon traditional materials, the jazzman must lose his identity even as he finds it. (Ellison 1964, 234; quoted in Gilroy 1993, 79)

Among hackers this cruelty, this difficulty in establishing discrete originality, is in reality not so cruel. It is treated like any interesting problem: an enticing

hurdle that invites rigorous intellectual intervention and a well-crafted solution within given constraints. Hackers clearly define the meaning of the free individual through this very persistent inclination to find solutions; they revel in directing their faculty for critical thought toward creating better technology or more sublime, beautiful code. The logic among hackers goes that if one can create beauty, originality, or solve a problem within the shackles of constraints, this must prove a *superior* form of creativity, intelligence, and individuality than the mere expression of some wholly original work.

Not every piece of technology made by hackers qualifies as a hack. The hack is particularly the "individual assertion within and against the group" (Ellison 1964, 234), which may be easily attached to an individual even though it is still indebted to a wider tradition and conversation. Hackers certainly engage in a creative, complex process partially separated from hierarchy, enfolding a mechanics of dissection, manipulation, and reassembly, in which various forms of collaboration are held in high esteem. Much of their labor is oriented toward finding a good enough solution so they can carry forth with their work. But their form of production is one that also generates a practice of cordial (and sometimes not-so-cordial) one-upping, which simultaneously acknowledges the hacker's technical roots and yet at times strives to go beyond inherited forms in order to implement a better solution. If this solution is achieved, it will favorably reveal one's capacity for original, critical thought—the core meaning of individuality among hackers.

Hackers recognize production as the extension or rearrangement of inherited formal traditions, which above all requires access to other people's work. This precondition allows one to engage in constant acts of re-creation, expression, and circulation. Such an imperative goes against the grain of current intellectual property law rationalizations, which assume that the nature of selfhood and creativity is always a matter of novel creation or individualized inventive discovery.

Among F/OSS hackers, the moral economy of selfhood is not easily reducible to modern "possessive individualism" (Graeber 1997; Macpherson 1962). Nor does it entirely follow the craftsperson or the stand-alone romantic author figured by intellectual property jurisprudence but rather evinces other sensibilities that point to competing liberal concepts of individualism and freedom. While hackers envisage themselves as free and rational agents, in the context of free and open-source hacking, most hackers place less emphasis on the freedom to establish relations of property ownership and exchange. Instead, they formulate liberty as the condition necessary for individuals to develop the capacity for critical thought and self-development.[21]

While the hacker interpretation of labor, creativity, and individuality strays from influential liberal understandings of personhood—possessive individualism—it does not represent a wholly novel take on these themes.

It aligns with the type of person presupposed in free speech theory, perhaps most lucidly in Mill's writings, which influenced the shape, content, and philosophy of free speech jurisprudence as it now exists in the United States (Bollinger and Stone 2002; Passavant 2002). Mill, influenced by the Romantic tradition (Halliday 1976), defines a free individual as one who develops, determines, and changes their own desires, capacities, and interests autonomously through self-expression, debate, and reasoned deliberation (Donner 1991). It is a vision that fuses utilitarian and romantic commitments, and is built on the idea of human plasticity and development—the ability of the self to grow and develop through creative expression, mental activity, and deliberative discussion, usually by following one's own personally defined path. As Wendy Donner argues, this form of liberal self-cultivation also requires the establishment of standards by which to judge the development of the human faculties. Mill's "transformed conception of utility necessitates a new method of value measurement which relies heavily on the judgment of competent agents," writes Donner (1991, 142), "and thus essentially rests on a doctrine of human development and self-development." What is notable is how Mill ([1857] 1991, 93) contends in his famous *On Liberty* that an individual must follow their own path of development, because "persons [. . .] require different conditions for their spiritual development." Even if this Romantic inclination prioritizes the individual, one can only develop the critical faculties along with moral and aesthetic standards via a process of training and open-ended argumentation in debate with other similarly engaged individuals.

Much of free software legal philosophy and moral sensibilities bear remarkable similarities to this Millian (and thus also Romantically informed) vision of personhood, self-development, and liberty, although there are differences and specifications tied to hacking's unique relations between persons, labor, and technology. Hackers place tremendous faith in the necessity and power of expressive activity that springs from deep within the individual self—an expression that acts as the motor for positive technical change. Progress depends on the constant expression and reworking of already-existing technology. Thought, expression, and innovation should never be stifled, so long as, many developers told me during interviews, "no one else is hurt"—a sentiment that is part and parcel of Millian free speech theories.

Free software developers have come to treat the pursuit of knowledge and learning with inestimable high regard—as an almost sacred activity, vital for technical progress and essential for improving individual talents. As one software developer observed, "I can use the code for my own projects and I can improve the code of others. I can learn from the code so that I can become a better programmer myself, and then there is all my code out there so that you can use it. It is just freedom." The spirit of this statement is ubiquitous among F/OSS developers. A utilitarian ethic of freedom and

openness is increasingly seen as not only obvious but also indispensable in order to develop the "state of the art."

For developers, technical expression should always be useful. If it isn't, it denies the nature of software, which is to solve problems. Yet hackers also place tremendous value on the aesthetic pleasures of hacking, producing technology and software that may not have any immediate value but can be admired simply on its own elegant terms—as a conduit for personal self-expression.

Over years of coding software with other developers in free software projects where discourses about liberty run rampant, many developers come to view F/OSS as the apex of writing software, as we will see in the next chapter. It has, they say, the necessary legal and material features that can induce as well as fertilize creative production. In contrast to the corporate sphere, the F/OSS domain is seen as establishing the freedom necessary to pursue *personally* defined technical interests in a way that draws on the resources and skills of other individuals who are chasing down their own interests. In other words, the arena of F/OSS establishes all the necessary conditions (code, legal protection, technical tools, and peers) to cultivate the technical self and direct one's abilities toward the utilitarian improvement of technology. While many developers enjoy working on their corporate projects, there is always a potential problem over the question of sovereignty. One developer told me during an interview that "managers [. . .] decide the shape of the project," while the F/OSS arena allows either the individual or collective of hackers to make this decision instead. F/OSS allows for technical sovereignty.

The hacker formulation of individuality, as the pursuit of one's interest for the mutual benefit of each other and society, is an apt example of the general characterization of modern individualism as defined, according to Taylor (2004, 20), by "relations of mutual service between equal individuals." While much of liberal thought understands mutual service in terms of economic exchange, hackers relate to it through the very act of individual expression and technical creation—the only sound ways to truly animate the uniqueness of one's being.

Conclusion

As noted in the previous section, even though hackers tend to approach other hackers as equals, they also construct themselves as high-tech cognoscenti creating the bleeding edge of technology. This elitism follows from their commitment to the organizational ideal of meritocracy, a performance-based system that applauds individual skill, encourages respectful competition between peers, and sanctions hierarchies between developers, especially in the F/OSS project to be discussed at length in the subsequent chapter.

The meritocratic ideal, ubiquitous in liberal thought, has particular resonance in the US popular imaginary. The United States is often thought of as a living embodiment of meritocracy: a nation where people are judged on their individual abilities alone. The system supposedly works so well because, as the media myth goes, the United States provides everyone with equal opportunity, usually through public education, to achieve their goals. As such, the hierarchies of difference that arise from one's ability (usually to achieve wealth) are sanctioned by this moral order as legitimate.

In many senses, hackers have drawn from what is still a prevalent trope of meritocracy to conceptualize how they treat one another and self-organize. In his classic account of hackers, Levy (1984, 43) includes this principle as one of the six elements that define the hacker ethic, noting that "hackers should be judged by their hacking, not bogus criteria such as degrees, age, race, or position," in which "people who trotted in with seemingly impressive credentials were not taken seriously until they proved themselves at the console of the computer."

Though written twenty years ago, this commitment to meritocracy still holds undeniable sway in the way F/OSS hackers construct norms of sociality and envision selfhood, not because it exists in the same exact way, but rather because hackers have given it new meaning by organizationally building the institution of the free software project guided by a dedication to meritocracies. Hackers who participate in free software projects routinely asserted that F/OSS projects are run as meritocracies. The doors are open to anyone, they insist; respect and authority are accorded along the lines of superior and frequently individual technological contribution. As we will see in the next chapters, F/OSS hackers may not build perfect meritocracies and yet they are certainly motivated to implement them.

For F/OSS hackers, it is imperative to constantly and recursively equalize the conditions by which other hackers can develop their skills and prove their worth to peers. As part of this equalization process, one must endow the community of hackers with resources like documentation and the fruits of one's labor: source code. The free software hacker does not privatize the source of value created, even those exceptional pieces of code that are undeniably one's own and seen to emerge from sheer technical ability. Within F/OSS, this value is fed back and circulated among peers, thereby contributing to an endowed and growing pool of resources through which other hackers can constantly engage in their asymptotic process of self-cultivation.

This constant recirculation of value is one way in which hackers can explicitly downplay their elitism and display their sound technical intentions to their peers. Their implementation of meritocracy contrasts markedly with the ideal of it in capitalist societies, where the privatization of value is legitimate as long as one generates wealth (or gains other forms of status) through one's personal ability. In fact, numerous issues over who and what are responsible for equalizing the terrain of competition plague

liberal democracies marked by a meritocratic ideal. This leveling is often seen as secured through such avenues as public education. That, in turn, raises questions like, Should capitalist philanthropists (such as John Rockefeller in the past and Gates in the present), individuals, governments, or property tax fund public education? With hackers, these sets of thorny issues are minimized, partially resolved by their constant recirculation of value, notably software and documentation, as well as debates and conflicts over mentorship and helping.

Still, the predominant sentiment is that once knowledge has been released to the collective of hackers, individuals must, on their own two feet, prove their worth by creating new forms of value that can be fed back recursively to the community. If one seeks too much help, this violates the hacker implementation of the proper meritocratic order, and one might be subjected to a stylized rebuff such as the common RTFM.

Among hackers, the commitment to elitism and meritocracy historically has run fairly strong. There is still an ambivalent relationship to elitism and this meritocratic ideal, however, as I will explore in more detail in the next chapter. I will show how those vested with authority on software projects, because of their success, are usually met with some degree of suspicion, and thus jokes and sometimes accusations of cabals run rampant among hackers. This requires them to constantly perform their trustworthiness and demonstrate their good technical intentions to the community at large. I now turn to the institution, the free software project, where technological production unfolds, and where commitments to free speech and meritocracy are further specified under the aegis of a tremendously varied set of ethical practices.

NOTES

Introduction: A Tale of Two Worlds

1. https://www.gnu.org/copyleft/gpl.html (accessed September 22, 2011).
2. It is now routine for anthropologists to unpack the effects of liberal formations by attending to the fraught politics of multiculturalism and secularism, the establishment of publics, the coconstruction of markets, marketing, and consumer desire, and the political changes wrought by new national constitutions and neoliberal policies (see Comaroff and Comaroff 2000, 2003; Ferguson and Gupta 2002; Haydn 2003; Mahmood 2004; Ong 2006; Povinelli 2002, 2006; Scott 2011). Despite this rich literature, the influence of liberal values in the context of Anglo-European societies still tends to figure thinly or inconsistently, either as an external economic influence that shapes cultural expressions, or more richly, as relevant to the discussion of secularism, religion, publics, and most especially, multiculturalism. The study of privacy and free speech, for instance, has tended to come in normative, philosophical, and legal terms (Bollinger and Stone 2002; Nissenbaum 2009; Rule 2009; Solove 2010). There is, however, a small but growing body of anthropological literature on liberalism and technology (Helmreich 1998; Malaby 2009) as well as the anthropology of the press and free speech (Boyer 2010; Keane 2009). For an enlivening historical account on liberalism as a lived set of principles in mid-Victorian Britain, see Hadley 2010.
3. Because the bulk of my research was conducted on Debian, a free software project, and with developers involved with other free software projects, my analysis also tilts in the direction of free over open-source software. And given how much attention has already been placed on open-source over free software, it is key to add this neglected perspective. But much of this book clearly applies to open source, for while even if open-source developers and projects de-emphasize a moral language of freedom (Chopra and Dexter 2007), they still routinely advance liberal ideals in, for example, their commitments to meritocracy and rational, public debate.
4. I am indebted to the stellar cultural analysis of liberalism offered by Stuart Hall (1986), who makes the compelling case that liberalism is not only a set of political creeds but also exists as cultural common sense composed of a set of interconnected principles that "hang together." Hall's definition is useful because he highlights some core features (such as a mistrust of authority and an accentuated commitment to individualism), yet he is careful not to pose a single logic to liberalism. He also argues that in its historical and lived dimensions, liberalism has incarnated into what he calls "variants of liberalism," replete with differences and contradictions. These differences and contradictions are still part and parcel of liberalism's life, and are evident among hackers.
5. A less humorous consequence of this ambivalence is the limited funding options available to students and researchers who choose to remain in North America for fieldwork (with the exception of those studying indigenous communities). Not only are existing funds nearly impossible to live on; there are few overall funding sources as well. So

even if we have managed to enlarge our field of inquiry, this is a case in which economic constraint works to discourage researchers from walking down a recently opened path.

6. For thoughtful contemplations on the method of participant observation and fieldwork, see Clifford and Marcus 1986; Comaroff and Comaroff 1992; Faubion and Marcus 2009.
7. Digital Millennium Copyright Act, 17 U.S.C. 1201(a)(1)(a).
8. One of the most crystalline examples of this utilitarian justification is provided in *Harper and Row, Publishers, Inc. v. Nation Enterprises*, a Supreme Court case deliberated in 1985. The question at hand was whether the magazine, the *Nation*, was entitled under the fair use doctrine to publish a three-hundred-word excerpt, in a thirteen-thousand-word article, from President Gerald R. Ford's twenty-thousand-word memoir published by Harper and Row. The court ruled in favor of Harper and Row, upholding the ideal that property rights promote a public benefit by inducing creation. Sandra Day O'Connor delivered the majority opinion portraying copyright as "the engine of free expression." Versions of this utilitarian rationale, in which Internet protocol (IP) is the basis for harvesting "knowledge," continue to be expressed and hold sway within the context of an heightened neoliberal expansion of intellectual property rights, making existing tensions between expressive and IP rights more palpable and acute than ever.
9. The Silicon Valley geek entrepreneur, who I am not addressing in this book, aligns quite closely with neoliberal aspirations. For a discussion of Web 2.0 technologies, entrepreneurs, and neoliberalism, see Marwick 2010.
10. http://mbrix.dk/files/quotes.txt (accessed April 10, 2007).
11. http://www.loyalty.org/~schoen/ (accessed March 19, 2007).
12. http://www.gnu.org/gnu/manifesto.html (accessed July 30, 2007).
13. https://upload.wikimedia.org/wikipedia/commons/b/b7/Anti-sec_manifesto.png (accessed, March 26, 2012).
14. Editorial, "The Victor Spoiled," *2600: The Hacker Quarterly* 15, no. 4 (1998–99): 4.
15. Although my exploration remains hemmed to free software and may not be relevant to *all* domains of hacking, there is certainly some overlap between what I describe and instances of hacking unrelated to the world of free software.
16. Gender also receives only cursory attention. The reasons for this omission are multiple, but foremost, I believe far more substantial research on the topic is needed before qualified and fair judgments as to the complicated dynamics at play can be posed, especially since analyses must interrogate wider social dynamics such as education and childhood socialization that have little to do with free software projects. In the last two years, a series of vibrant initiatives around diversity and gender have proliferated in the context of free software, with tremendous support from the wider developer community—something I have not been able to research adequately.
17. While this book attends to a number of translocal aspects of F/OSS development, it by no means captures the reality of all different places where free software has taken hold, such as India, Vietnam, Peru, and Brazil. For instance, many free speech commitments explored in this book are shared by Brazilian developers I worked with, even while the general story of free software in Brazil and other parts of Latin America looks quite distinct from what happened in the United States given how entwined it became with national politics (Chan 2008; Schoonmaker 2009; Murillo 2009).
18. The region, despite being dominated by high-tech capitalism, is by no means monolithic. It is home to a range of distinct values, stretching from staid engineering commitments

(English-Leuck 2002), to countercultural expressions (Turner 2006) and new age currents (Zandbergen 2010), to undoubtedly liberal (Malaby 2009) and neoliberal orientations (Marwick 2010).

Chapter 1: The Life of a Free Software Hacker

1. Most of the developers I interviewed were between the ages of eighteen and thirty-five, although there were a number over thirty-five years old (there were some under eighteen who I interacted with but did not formally interview due to provisions in my Institutional Review Board application). Thus, this life history is located very much in time, with the narrative spanning the period between the late 1970s until the present.
2. Warez typically refers to commercial or proprietary software that has been cracked or pirated, and therefore illegally circulated to the larger public (in the past on BBSs and currently on the Internet). For this to happen, the software's copy protection measure must be deactivated. In contrast, shareware is copyrighted software that is released by its author initially for free on a trial basis or under some other set of conditions.
3. For decades, computer science was a branch of mathematics or class offerings were scattered in different departments. Although MIT was home to many important computer projects, for instance, it only began offering an undergraduate computer science course in 1969. The first computer science department was established in 1962 at Purdue University, and it was not until the mid- to late 1970s and early 1980s when many US universities started to establish stand-alone computer science departments (Ensmenger 2010, 120–21). See also "History of the Department of Computer Sciences at Purdue University," http://www.cs.purdue.edu/history/history.html (accessed October 23, 2011).
4. These quotes are culled from my life history interviews.
5. Efficiency can mean various things for programming/software, including running faster, using less computing resources, or both.
6. For a comprehensive history of the BBS era, see the excellent eight-part documentary *BBS: The Documentary* by Jason Scott (2005).
7. BBSs also played a prominent role among phreaks and underground hackers (Thomas 2003; Sterling 1992). Usenet, a large newsgroup service, was significant for hackers as well (Pfaffenberger 1996).
8. FidoNet, established in 1984, was an independent mail and information transport system that connected BBSs together.
9. IRC happens on IRC servers (EFnet, Freenode, etc.) that run software that allow users to set up "channels" and connect to them. There are a number of major IRC servers around the world that are linked to each other. Anyone can create a channel, and once created and populated with users, all others in the channel can see anything anyone types in a given channel. Using IRC client software, a user can connect to multiple servers at once, and join multiple channels, switching conversations by switching tabs or windows. While conversation on the channel is public, one can also initiate multiple private conversations. IRC has grown tremendously since it was first created in 1988. In July 1990, IRC averaged at 12 users on 38 servers. Now there are thousands of servers, and over 100,000 users on some servers. To give a sense of its growth, one of the more