8.1.2 Fuck the System

In Chapter 7 I argued that the mix up between social systems and operating systems was not just an arbitrary overlap or accident. Thus behind the RO versus RW cultural dichotomy of Lessig, using file system permissions as an analogy to describe cultural processes, I have briefly explained that computer operating systems and their networking can provide different models of social organisation, with different levels of transparency and policing, from small-scale emulations of property-less pseudo-secret societies to panopticonesque chroot jails. This led me to use the term sandbox to refer to these different architectures that have increasingly relied on techno-legal templates, and most notably in the context of this text, those derived from free and open source software licensing. If this approach allowed me to create a counter argument—by simply looking at the ways cultural expressions are produced and not just accessed—to the trivial pro free culture binary RO versus RW, I now want to discuss the refusal to engage with these sandboxes and their techno-legal fabric, when they create a conflict of belief, values, or ideas.

To do so, in this section I will examine the work from French noise and experimental musician and computer programmer Yves Degoyon.[28] If some are busy pondering about file permissions, Degoyon is more in favour of simply getting rid of the files and the whole OS at the same time. This is the basis for his performance *rm -rf/* :: f*** the system*—or

[28] The text from this subsection is based on a semi-structured email interview with Degoyon, that took place during April 2015 and March 2016.

/bin/rm -rf / :: f*** the system*—in which the musician performs using an audiovisual noise generating Pd patch, while at the same time opening up a terminal on his computer and runs the command `/bin/rm -rvf /*`, that will in effect recursively force-remove every file and directory under the root file system, while the names and paths of said files and directories are printed on the terminal of his GNU/Linux distro. Eventually with the file system emptied and only a handful of programs and data left in the RAM of the machine, the computer crashes, sometimes with unexpected behaviour, and with it ends the performance.

Degoyon told me that the work is mainly an experiment in chaos and the instability of computer systems. However he also admits that the title hints obviously towards a double meaning, and the action that needs to be taken to get rid of a system before it alienates you. Degoyon grew up listening to post-punk bands such as Wire, Gang of Four, and This Heat, which while having widened the cultural scope of punk, have done so, according to Degoyon, notably through the generalisation of punk's DIY spirit. Here the punk connection can be deceptive, because the title of the performance is to be understood differently from the way English punk singer Johnny Rotten claimed to have fucked up the system, when he was part, with other proto punks and early punks, of what has been described as a working class Bohemia.[29] Instead, a more abiding connection would be the 1967 pamphlet *Fuck the System* by American political and social activist Abbie Hoffman, a text filled with tips and advice to

[29] Simon Frith, *Sound Effects: Youth, Leisure and the Politics of Rock* (London: Constable, 1987), 266.

organise and survive in the "city jungle" and the development of a "freer more humanistic" society.[30] So it should not be surprising that in his approach, Degoyon feels more connected to the early days of British collective and arnarcho-punk band Crass, which he often quoted and referred to during heated mailing list discussions, where the link to avant-garde art and anarchist political movements was not a trivial appropriation as it was in other early punk bands, but was more seriously explored via direct action and zine publishing, so as to advocate animal rights, anti-war, anti-consumerism, vegetarianism, environmentalism and feminism.[31]

When Degoyon started to use and write free software, it is through this art punk anarchist inspiration that he engaged with this particular digital form of knowledge sharing. During our exchange, he refereed to the Spanish video collective R23,[32] founded by artist and computer scientist Lluis Gomez i Bigorda, as an example of introducing such elements into media art practices. Degoyon contributed to R23 DIY streaming media projects and network mapping in the early noughties, and admitted to enjoying the perturbation generated with the introduction of "some spirit of activism in the polished world of media art,"[33] at a time where

[30] The text also paved the way for a better known work by Hoffman, the 1971 *Steal this Book*, which I mentioned in Chapter 2 in connection to open design and DIY. Of course Hoffman is not the only connection to be made here. Sixties anarchist guerrilla street theatre group the Diggers were early explorers of ideas of anonymity, freedom of association, and societies free from private property, using a wide range of practices from direct action and art happenings, to the publication of leaflets and manifestos. See Emmett Grogan, *Ringolevio : A Life Played for Keeps* (1972; repr., New York: New York Review Books, 2008).

[31] See Johan Kugelberg, *In All Our Decadence People Die: An Exhibition of Fanzines Presented to Crass Between 1976 and 1984* (New York: Boo-Hooray, 2011).

[32] R23.cc, "r23.cc," 2005, https://web.archive.org/web/20050312155147/http://r23.cc/community/.

[33] Email to author, April 8, 2015.

the mix of free software and art offered a self-organised and decentralised alternative to artistic media labs.[34] However, what was first perceived as an ideological alignment between Degoyon's beliefs and the technological environment he was contributing to, unfortunately quickly turned into something very illustrative of the alienation expressed in his performance.

One of the software project actively developed by Degoyon at the time was PiDiP,[35] which stands for PiDiP is Definitively into Pieces, a BSD-style licensed Pd external that brings extra video processing capabilities and builds upon the GPL'ed Pure Data Packets (PDP) Pd video processing objects by Belgian software and hardware developer Tom Schouten,[36] and also sharing some code with GPL'ed real-time video effect software EffecTV, originally developed by Japanese programmer Kentaro Fukuchi. But two events made Degoyon question the relationship between his political views and free and open source software production. He explained to me that the first event was a conversation with a CCTV company in 2004, that was present in an international meeting of activists in Switzerland, and that was interested in using free software technology for motion detection. The second event occurred at a free software meeting in Brazil in 2005, where representatives from the army were assessing the viability of using free software in their surveillance systems. Degoyon told me

[34] See Annet Dekker, Angela Plohman and Irma Földényi, "Interview with Dave Griffiths, Aymeric Mansoux and Marloes de Valk," in *A Blueprint for a Lab of the Future*, ed. Angela Plohman (Eindhoven: Baltan Laboratories, 2011).
[35] See Yves Degoyon, "PiDiP Is Definitely in Pieces," 2011, http://ydegoyon.free.fr/pidip.html.
[36] Tom Schouten, "Untitled Page," 2012, http://zwizwa.be/pdp/.

that he obviously could not accept that, and was the reason he first first decided to add a clause to his BSD-style license "NOT FOR MILITARY OR REPRESSIVE USE !!!", and later on take a more radical step by releasing PiDiP under his own license in 2010:

> to cut with all legal blah-blah, this license will be made short.
>
> the code published here can be studied, modified, used by anyone that provides all the original credits and sources in derivative projects.
>
> there are restrictions on its use, it cannot be used for :
>
> - military amd/or repressive use
> - commercial installations and products
> - any project that promotes : racism, nationalism, xenophobia, sexism, homophobia, religious hatred or missionarism .. (expandable list)
>
> this is not a standard license.
>
> sevy & authors.[37]

These two changes in PiDiP's licensing terms are an interesting case of fucking up the sandboxing system. Degoyon, who told me he had originally chosen a copyfree[38] BSD style license because it was like Pd's own license, was in fact releasing a software containing an assortment of code from copyleft'ed EffecTV, bits and bytes from other sources and collaborations, and also his own code written from scratch. By initially releasing PiDiP with a non-copyleft non-GPL compatible license and yet using some copyleft'ed parts, he was breaking the GPL and misusing the

[37] LICENSE.txt file from the PiDiP CVS repository, revision 1.1.1.1, commitid: MR5avkuVSyEPgbZ, 2010-12-06 06:31:45. The typo will be fixed with commit aOzDtQZu7yTgVL9v, in February 2011 for version 1.2.

[38] For an an explanation on copyfree licensing see Chapter 5, The Double Misunderstanding with Copyleft.

copyright of others. A GPL-respectful way to publish PiDiP should have been for instance either under the GPL, or as two collections of source files, the GPL modified ones under the GPL and the others under the BSD style license or else, assuming Degoyon did not use other chunks of source code licensed differently, in which case further fragmentation of the software would have been necessary in case of license incompatibilities. But Degoyon cared little about that fact and in 2006 stated on the Pd mailing list, in a very art punk anarchist way, that people should not forget that PiDiP contributors like to "confuse lawyers and boring people first."[39] Funnily enough, the original mis-licensing—when PiDiP was distributed as BSD yet including GPL code from EffecTV—did not prevent the software to be successfully validated by FSF employees and listed in 2003 by the FSF directory with other *useful* free sofware—as I discussed earlier in Part 2—which shows that traceability and transparency in free and open source software has its limits.

Regardless of Degoyon's little interest in respecting licensing terms—a situation which shows some ressemblance with Stallman's early EMACS days where code circulation was more important for the hacker than diligent respect of copyright laws[40]—was an important figure of the Pd community, whose software was used by several artists and packaged or distributed by other developers. However, this started to change in 2005 when the licensing issue was brought up in the Pd mailing lists. The issue dragged on for years with extremely heated discussions on the user

[39] Yves Degoyon, "[PD-Ot] Pidip Inherits Gnu Gpl from Effectv," 2006, https://lists.puredata.info/pipermail/pd-ot/2006-01/001377.html.
[40] See Part 1.

and developer lists of the software. Degoyon's contributions to the debate tended to add oil to the fire as he explicitly grounded his refusal to change his license based on political motivations—with even more oil poured when he started to change the BSD license into a non-copyfree non-military license—whereas those asking him to conform acted as a sort of neighbourhood watch system, trying to enforce the cyber constitution of the Pd sandbox. I use the words neighbourhood watch here, because in fact *only* Kentaro Fukushi, and possibly other contributors of EffecTV, were the ones who could require their licensing to be enforced. As it turned out, Degoyon and Fukushi had already met on several occasions previously, and the EffecTV author knew of PiDiP and appreciated the fact that his work had been ported to Pd. His silence on the mis-licensing matter may have seemed to indicate he cared little about the potential licensing problem with PiDiP. However, as Degoyon was further pushed in to a corner within the Pd community, which in turn led to the radicalisation of his licensing strategy, PiDiP started to break more constitutive mechanisms of other sandboxes, such as operating systems like Debian, or free and open source software hosts like SourceForge. Simply put, by means of TOS, social contracts, or other usage agreements, these platforms and operating systems can implement their own definition of software freedom, which help decide which licenses they allow, ultimately shaping the software culture they distribute. PiDiP's new license was incompatible with many of such definitions. Eventually PiDiP became, in 2010, a *software non grata* removed from the Pd repositories and distributions.[41] At time of writing, PiDiP, the impossible *copypunk* source code, only exists in a limbo of various repositories outside and disconnected from the Pd community, but it is still listed in the EffecTV project links as well as in the FSF free software directory.

Within the free cultural techno-legal template, the practice and intention that led to the creation of PiDiP, a software that grew organically from the encounter of the author with other artists and developers—and the source code they wrote, notably within the projects of the R23 collective—became incompatible with its technical and legal framework. It challenged the definition of freedom carried by the sandbox it was born within, and illustrated the non-trivial interaction between the changes through the years of an author's thoughts, the fluidity of the digital medium his creation was written in, and the rigidity of its legal framework. In such a situation, PiDiP, published by a rather proud outlaw,[42] nonetheless found a deadlock and the execution of its *legal* instructions became eventually incompatible within the system it was developing, as opposed to its perfectly running *software* instructions. This example shows once again the strength of the techno-legal template, and its dual level of interpretation by machines, and humans, initially discussed in Chapter 1. To be sure, Degoyon's stand should not be marginalised or neglected because it was the response of an artist in the context of a niche software community. In fact, similar responses

[41] This removal was effective with commit r14502 from the Pure Data SVN code repository, which motivated Degoyon to start hosting his own public code repository on giss.tv and change the license even more, as discussed previously.
[42] In reference to Yves Degoyon, "[PD] Percolate," 2007, https://lists.puredata.info/pipermail/pd-list/2007-03/047953.html.

and critiques towards free and open source projects have also been articulated notably by Felix von Leitner, a German IT security expert and ex-member of the Chaos Computer Club:

> This is what we get when our free software licenses lack a 'not for military purposes' clause: DARPA presents a weapon control system on the basis of Android tablets http://www.darpa.mil/NewsEvents/Releases/2015/04/06.aspx. Linux is now killing people.[43]

More recently, in 2015, one of von Leitner's own GPL licensed free software projects, dietlibc, a popular lightweight C standard library,[44] was shown to have been used in products sold by Hacking Team, the Italian Information Technology company specialised in providing corporations and governments with intrusion and surveillance technology. Next to the breach of the GPL copyleft, this situation further prompted Leitner to call for a NOMIL/NOINTL license, and started to put in motion a modification of the AGPLv3 as a foundation for such a license.[45] von Leitner's effort is not singular, and there has been in the past several projects that became non-free and non-open source software, in spite of the availability of the source code, simply because they used statements,[46] or licensing techniques that exclude military usage like the Peaceful Open Source

[43] 'Das haben wir jetzt davon, dass wir in unseren freie-Software-Lizenzen keine "nicht für militärische Anwendung"-Klausel haben: DARPA präsentiert ein Waffensteuerung auf Basis von Android-Tablets. Linux tötet jetzt Menschen.' Translation Florian Cramer. Felix von Leitner, "Fefes Blog," 2015, https://blog.fefe.de/?ts=abda600a.

[44] Felix von Leitner, "diet libc - a libc optimized for small size," 2016, https://www.fefe.de/dietlibc/.

[45] Felix von Leitner, "Fefes Blog," 2015, https://blog.fefe.de/?ts=ab645846.

[46] See Roedy Green, "Non-Military Use Only," 2017, http://mindprod.com/contact/nonmil.html.

License.[47]

PiDiP, whose name indeed announced its demise, precisely shows what happens when the license as community law take over the values it was thought to be defending. Accepting to use a specific license against one's own beliefs brings the risk of creating cognitive dissonance, and Degoyon avoided this by putting his beliefs before the sandbox's rules when he noticed the contradiction created by the situation. But even though passion and affects are crucial in creating allegiance to democratic values,[48] they must be removed from the rationalised model of free culture for the latter to operate smoothly, and could explain why some participants of free and open source projects present their work detached from political intentions.[49] This is not just an issue of social dynamics within small communities, but it is also visible in the way the infrastructures that support free culture operate. To give a short example, in 2009, the jsmin-php software was banned from Google Code because the software had inherited the license of jsmin.c it was based on, a license that was a modified version of the free and open source software MIT license. The modification was one line stating "The Software shall be used for Good, not Evil", which made the software non-free and gave the "Don't be evil" company a reason to exclude the code from its free and open source software hosting platform.[50] Interestingly enough, and

[47] Linkesh Diwan, "Peaceful Open Source License," 2014, https://web.archive.org/web/20140924010836/http://wiseearthpublishers.com/sites/wiseearthpublishers.com/files/PeacefulOSL.txt.

[48] Mouffe, "For an Agonistic Model of Democracy (2000)," 199–200.

[49] Coleman, "The Political Agnosticism of Free and Open Source Software and the Inadvertent Politics of Contrast."

[50] See Ryan Grove, "JSMin isn't welcome on Google Code," 2009, http://wonko.com/

linking back to my earlier neighbourhood watch analogy, Google did not scan their repository for non-compliant licenses, they were simply informed by another user in the main discussion forum of the Google Code virtual community.[51]

As shown with these examples, there is only a thin balance between the free software *Gemeinschaft* emulation, and the implementation of a cyber disciplinary society. Free culture in this context is far from being the liberating and pluralistic tool it seemed to be, or to be more precise and to refer to the first part of this thesis, I have shown with this example that the aggregative and deliberative democratic models of free culture, have risen at the cost of antagonism and radicalisation of cultural practices, by limiting rapid cycles of hegemonic and counter-hegemonic efforts, that used to be more prominent during the chaotic era of proto-free culture. As a result, free culture sandboxes become absolute democracies in which not only artists such as Degoyon, but any participant in fact, are effectively forbidden "to engage with a multiplicity of agonistic democratic struggles to transform the existing hegemonic order,"[52] because their software becomes a threat to a public space that according to the defined free culture can only exist as a consensual thing, and that is defined by certain parameters that rely on the exclusion of others.

post/jsmin-isnt-welcome-on-google-code.

[51] Adam Goode, "jsmin-php not open source," 2009, https://groups.google.com/forum/#!topic/google-code-hosting/F8P68oKPXA8.

[52] Mouffe, "Cultural Workers as Organic Intellectuals (2008)," 215.

8.2 Fork the System

Next to complete obedience or complete resistance, one particular side-effect of a free cultural mechanism that promotes the circulation of information over the context of its production and usage, allows a third approach to engage with sandbox dynamics: forking.

Forking can be described as the process by which the source code of a piece of software can be modified, so as to make, for instance, new software integrating modifications, minor or major, that would not have been accepted by the author(s) and community from which the fork stemmed, or simply to explore transformations unforeseen by the original authors of a work.[53] The divergence of source code and the proliferation of concurrent versions of the same software is not specific to free and open source software and became an important aspect of source code sharing in the early days of UNIX , as it was discussed in Chapter 1. It has also been argued that copyleft development could either deter forking motivated by competition, and allow merging back at a later stage if forking occurs.[54] However, the rationalisation of source code sharing with the creation of free and open source software licenses, can also be interpreted as taking a radical path towards divergence, a "right to fork,"[55] regardless if open forms of developments are made mandatory as with copyleft li-

[53] For a general explanation regarding forking in free and open source software culture, some historical references, and a case study with the Debian and Ubuntu operating systems, see Benjamin Mako Hill, "To Fork or Not to Fork: Lessons from Ubuntu and Debian," 2005, https://mako.cc/writing/to_fork_or_not_to_fork.html.

[54] Andrew M. St Laurent, *Understanding Open Source & Free Software Licensing* (Sebastopol: O'Reilly, 2004), 171–73.

[55] Weber, *The Success of Open Source*, 159.

censes. In that sense license-assisted forking can be seen more as a liberal remix-culture-oriented free culture approach, than a community-binding copyleft mechanism. Both are in fact different materialisations of the rules of software freedom. Due to the difference of context in which such materialisation occurs—acquiring existing work versus contributing to existing work—forking originally had as a result, a very bad reputation. Yet, it has risen today to become a very important mechanism central in the writing of free and open source software, in the age of connected machines and users, and an important component in sandbox dynamics and underlying mechanics of the constant becoming in free and open source software communities.

Before elaborating on the details of such a mechanism—notably with the software `git` that I will introduce later in this section—I must first briefly explain how forking has co-evolved with the different generations of tools which have facilitated the writing of software. What is interesting in this co-evolution is the apparent contradiction between the desire to develop a very liberal approach to producing and distributing software, but done so through the very techno-legal means and methods that will later be feared by those defending such a liberal system. In particular, libertarian computer programmer Eric S. Raymond, who famously articulated the negative consequences of forking:

> Nothing prevents half a dozen different people from taking any given open-source product (such as, say the Free Software Foundations's GCC C compiler), duplicating the sources, running off with them in different evolutionary directions, but all claiming to be the product.
>
> This kind of divergence is called a fork. The most important characteristic of a fork is that it spawns competing projects that cannot

later exchange code, splitting the potential developer community.[56]

Here it becomes clear that the fork is more than a threat to these communities, it is a threat to the mechanism of reciprocity which is central to the gift economy,[57] and which inspired Raymond to describe free and open source software community as gift culture.[58] Of course, as I explained previously in this third part, and regardless of the desires and mechanisms of reciprocity put in place, it is to be expected that a system deeply inspired by classic liberal dynamics will create competition between different actors trying to maximise profit, whatever this profit is, either financial or based on the free circulating information they can access to. In that sense, forking can become a tool to accelerate competition. Raymond however seems to preemptively defuse the problem by arguing that there is a discrepancy between what he calls *the yield* implied by free and open source licenses, which according to him is only *use*, and the yield of participation in the production of free and open source software that is "peer repute in the gift culture of hackers, with all the secondary gains and side-effects that implies."[59]

In this context indeed, forks are therefore negative for the community as they "tend to be accompanied by a great deal of strife and acrimony between the successor groups over issues of legitimacy, succession, and design direction."[60] The fork here is seen as a form of failure to reach con-

[56] Eric S. Raymond, "Homesteading the Noosphere," *First Monday* 3, no. 10 (1998), http://firstmonday.org/ojs/index.php/fm/article/view/621/542.
[57] Mauss, *The Gift*.
[58] Raymond, "Homesteading the Noosphere."
[59] Ibid.
[60] Eric S. Raymond and Guy L. Steele, "THE JARGON FILE, VERSION 4.2.2," 2000, http:

sensus around a common techno-legal authority, that should in theory satisfy all the inhabitants of the sandbox. But given its political power, the threat of forking could also work as part of a strategy to influence the direction of a project, and has been described as similar to a " 'vote of no confidence' in a parliament,"[61] a convenient way to work around the effective vote-less rough consensus found in some of these communities.[62] Therefore in the early days of free and open source software development, the fear of forking may have worked as a glue to assemble and maintain large software community sandboxes, where the desire for liberal and libertarian structures was nuanced by the necessity to maintain cohesion in these world of techno-legal social systems, leading to a sort of macro liberalism. Another account is to note that in certain cases, the trademarking and other means of protecting the name of a project has helped discourage the creation of competing projects.[63] Lastly, it was argued that the trading aspect of free and open source software development shared resemblance with iterated games around reputation, and thus the fear of forking introduces a reputation risk.[64] Said differently, it may have not been the threat of schism, name protection, or reputation, that limited the proliferation of radical software freedom, that is forking, but simply that the act of forking took significantly more effort than solving issues within an existing community. However, another expla-

//catb.org/jargon/oldversions/jarg422.txt, *forked* entry.

[61] David A. Wheeler, "Why Open Source Software / Free Software (OSS/FS, FLOSS, or FOSS)? Look at the Numbers!" 2015, https://www.dwheeler.com/oss_fs_why.html#forking.

[62] Stadler, *Digital Solidarity*, 39.

[63] Andrew M. St Laurent, *Understanding Open Source & Free Software Licensing*, 173.

[64] Weber, *The Success of Open Source*, 159.

nation could simply be that the development platforms available at the time were simply not flexible enough to facilitate forking, therefore prevented a more radical take on software freedom and the free circulation of information.

In the history of software engineering, tools such as version control systems (VCS), also known as revision control and source control, have allowed developers to keep track of changes in software. When Marc J. Rochkind started research on VCS in 1972 at Bell Labs with the project Source Code Control System (SCCS),[65] running both on IBM 370 OS and PDP 11 UNIX, the idea to approach software development to reflect on the continuous and concurrent nature of software engineering was deemed radical,[66] but it was not entirely new, because IBM had already been working on a way to facilitate and control software engineering with their 1968 CLEAR-CASTER system—the combination of the Controlled Library Environment and Resources (CLEAR) and the Computer Assisted System for Total Effort Reduction (CASTER)—so as to provide a unified programming development support system and batch processing system. In the CLEAR-CASTER system, changes to source were notably detached from the actual source text to facilitate the keeping track of changes as well as providing contextual documentation for the software.[67] These VCS and others from the first generation, to borrow from Raymond's classification

[65] Marc J. Rochkind, "The Source Code Control System," *IEEE Transactions on Software Engineering* 1, no. 4 (1975): 369.
[66] Ibid., 368.
[67] John N. Buxton and Brian Randell, "Software Engineering Techniques," Report on a conference sponsored by the NATO Science Committee, Rome, 1969 (NATO Science Committee, 1970), 5.3 Support Software for Large Systems.

of such tools,[68] worked by sharing the same file system, but with the rise of computer networks and remote access to computational facilities, VCS eventually evolved to adopt a client-server model. This shift occured with the Unix tool Revision Control System (RCS) created in 1982 by German computer scientist Walter F. Tichy,[69] first following a local data model, the functionality of which was enhanced in 1985 by Dutch computer scientist Dick Grune[70] so as to facilitate collaboration across several users. Grune's work eventually led to the creation of the Concurrent Versions System (CVS), that existed, not without some irony, as two concurrent projects.[71]

As part of a client-server VCS like CVS, or its successor subversion (SVN) introduced in 2000 to improve some of the flaws of CVS,[72] the code repository is commonly served from a single machine, the server, that keeps track of all the changes in the source code. For instance, a programmer can use a VCS client software to retrieve changes made by other programmers and which are stored remotely on a machine running the VCS server software that serves and tracks changes in the central repository. The programmer can then make further modifications locally on their personal machine, and eventually commit changes to the central repository, granted they are allowed to do so by the server. It is not

[68] Eric S. Raymond, "Understanding Version-Control Systems (DRAFT)," 2008, http://www.catb.org/esr/writings/version-control/version-control.html.
[69] Walter F. Tichy, "RCS—a System for Version Control," *Software: Practice and Experience* 15, no.7 (1985): 637–54.
[70] Dick Grune, "The relation between my CVS, Brian Berliner's cvs and GNU CVS," 1992, https://dickgrune.com/Programs/CVS.orig/CVS_BB_and_GNU.
[71] Ibid.
[72] Michael Pilato, Ben Collins-Sussman and Brian Fitzpatrick, *Version Control with Subversion* (2002; repr., Sebastopol: O'Reilly, 2008), xiii–xiv.

difficult to see that there is a lack of balance in this control structure because developers can be denied access to the central repository. But it also means, that getting access to the whole database, the history of the project, is not trivial because all of this is handled on the server side. On the other hand, because of this gated and centralised architecture, requesting access to a project VCS, to be trusted with such access, can only be done by socially interacting with the community or group working on the software. Changes to the system are therefore also scrutinised and discussed within these same groups and communities, as access to the main VCS repository of a project does not imply anything can be committed. But it is important to note that once again, those in charge of writing software within such environments are not necessarily those able to change and modify such software environments, and the writing of software can be done following many different participatory and managerial models, often referred to as *governance models* within free and open source software management discussions.[73]

In 2005 Scottish artist, writer, and programmer Simon Yuill introduced the concept and framework of Social Versioning Systems (SVS), used in his social simulation game spring_alpha,[74] where players are invited to take part in an uprising to form an alternative society to that of the capitalist, normalising and disciplinary world they've lived in so far. Next to traditional game mechanics derived from interactive fiction and open-

[73] Ross Gardler and Gabriel Hanganu, "Governance Models," 2013, http://oss-watch.ac.uk/resources/governancemodels.

[74] Simon Yuill, "SVS [about]," 2006, http://www.spring-alpha.org/svs/index.php?content=about.

ended world simulation, the novelty of spring_alpha is that players were able to re-write the code that runs the simulated world,[75] a process both facilitated and tracked by SVS. SVS and spring_apha are both inspired by, and illustrate well, the constitutive and social dimension of the free software techno-legal templates that lead to the creation of sandboxes, whereby rules can be theoretically challenged and modified following different models of participation. One aspect of SVS in particular was prompted at the time by the growing availability of tools to monitor, visualise and further track changes within version control code repositories, as well as quantify and contextualise them. Looking back today at the way the tracking tool provided by SVS pushed the idea of VCS as a glue to bridge social systems with their techno-legal frameworks, it is striking to see how some of the principles provided by this critical art and research project announced, coincidentally, an age in which VCS are nowadays combined and interleaved with discretised and "computable orderings,"[76] not however to reprogram the social systems they're used in—and this is the key difference—but rather to further order and control software work and dominant modes of production, as best exemplified with the social-coding platform GitHub.[77]

Indeed, if Yuill's ideas were rooted in the understanding that the moral and social aspects of work were not solely determined by technology, as Coleman explained with her work on free software communities as

[75] Ibid.

[76] Quinn DuPont and Yuri Takhteyev, "Ordering Space: Alternative Views of ICT and Geography," *First Monday* 21, no. 8 (2016), http://journals.uic.edu/ojs/index.php/fm/article/view/6724/5603.

[77] Ibid.

high-tech guilds,[78] and therefore whose dynamics had the potential to be internally contested and challenged with very rare occasions of forking, this was not without counting on two aspects. The first is as I described earlier with the two Pd examples, which showed that the immutability of the legal fabric of these sandboxes in practice greatly limits counter-hegemonic efforts. But most importantly here, the second aspect is that such analysis and work were highly dependent on the state of all these software frameworks that helped manage and control software production. If client-sever models of version control, for instance, introduced a great change and reinforced the role of governance models—a sort of golden age for systems based on Raymond's description of bazaar versus cathedral and benevolent dictatorship versus meritocracy[79]—the third change in the history of such tools, which I will now introduce, is without question the one that will exacerbate the tension between the two approaches to software freedom that I have introduced in this section, and as a consequence the tipping point that will change the way forking was perceived thus far.

This third alteration is the replacement of client-server architecture with that of distributed version control system (DVCS). With DVCS, there is no more central repository, and no more fixed topology for the networked organisation of software production. Because each DVCS is both client and server, every copy of the project *is* a fork and the programmer works first on their local copy before deciding to push which part of their

[78] Gabriella Coleman, "High-Tech Guilds in the Era of Global Capital," *Anthropology of Work Review* 22, no. 1 (2001): 28–32.
[79] Raymond, *The Cathedral and the Bazaar.*

changes and to which other repository. At first this model seems to suggest a less rigid relation between the embedding of moral and social aspects of work in technology, because indeed in the case of DVCS it is up to social conventions to shape the network topology of software production, and this with extremely great flexibility, with the possibility of breaking free from the more traditional models of governance. However when several DVCS implementations—such as arch, bazaar, codeville, darcs, git, mercurial—started to gain popularity in the mid-noughties they were not perceived positively,[80] precisely because "the very conveniences [DVCS] provides also promote fragmentary social behaviours that aren't healthy for [free and] open source communities."[81] It is a threat because the historical situation becomes suddenly inverted: forking takes less work and effort than interacting with an existing community. Sending changes back to other code repositories becomes optional, and depends on the willingness to interact with other developers, and of course the willingness of these to accept changes. Above all, DVCS shows that the old assumption where "it will almost always be more economical for a potential forker to try to get the technical changes he wants incorporated into the existing code base [...], rather than to split off and try to create a new community,"[82] might have been wishful thinking, or at least needs serious revision.

However, in the same way the success story of the Linux kernel project

[80] Ben Collins-Sussman, "The Risks of Distributed Version Control," 2005, http://blog.red-bean.com/sussman/?p=20.
[81] Ibid.
[82] Weber, *The Success of Open Source*, 160.

helped construct the nineties free and open source software narrative of many programmers collaborating and working together, and became a poster child for the bazaar and benevolent dictator model of free software governance, the same project is at the centre of a shift in mentality regarding forking. As mentioned in the previous part of this thesis, Android, Google's mobile operating system, relies on the Linux kernel, but due to several issues that are not so relevant here,[83] Google's work on the kernel was essentially done on a branch which grows further away from its original source, with little to no possibility of merging back changes and additions. In turn, the initial contributions, then abandoned, from Google to the mainline source code repository were removed. The conflict was initially framed as a stereotypical situation were communication is difficult but forking is easier, but what was new here, is that next to the usual knee-jerk response of forking as a threat to communities and the reciprocal blaming for which party was at the source of the situation, there was a subtle shift in the perception of forking. Chris DiBona, American software engineer and director of Open Source and Science Outreach at Google, posted during the tense exchanges of 2010:

> [...] this whole thing stinks of people not liking Forking. Forking is important and not a bad thing at all. From my perspective, forking is why the Linux kernel is as good as it is.[84]

The rise of DVCS put in motion a process in which forking transformed

[83] Steven J. Vaughan-Nichols, "Linus Torvalds on Android, the Linux Fork," 2011, http://www.zdnet.com/article/linus-torvalds-on-android-the-linux-fork/.

[84] Chris DiBona, "Greg Kroah-Hartman: Android and the Linux Kernel Community (Comment)," 2010, https://lwn.net/Articles/372419/.

from vice to virtue. because in effect it offered a way for sandboxed communities to go forth and multiply by following this radical materialisation of deregulated software freedom, and expanding the development of the *metacommunities*, "sparsely or thickly connected populations of objects, users, producers"[85], that surround code repositories. But this new approach also launched into fame a web platform such as GitHub, in leading the self-coined trend of social coding, that sits at the cross-roads of social networks, project managements tools, and revision control.[86] On GitHub, anyone is able to have several public `git` repositories, a popular revision control system, and is given the ability to fork any other repository by clicking on a button, simply called *Fork*. The button is enhanced with a counter that reveals how many forks have been made of the given repository, making explicit, within this platform, how forking ends up as a popularity contest. Users of the platform are also able to contribute back changes they make to their fork, to the parent repository, and employ a specific property of `git`, which allows them to cherry-pick changes made in other forks. These basic operations represent the so-called "social life" of code sharing on GitHub.[87] They can also simply ignore the parent repository and give a new context to their fork. In fact other features offered by both the `git` software and GitHub itself, and the ability

[85] Matthew Fuller, Andrew Goffey, Adrian Mackenzie, Richard Mills and Stuart Sharples, "Big Diff, Granularity, Incoherence, and Production in the Github Software Repository," in *Memory in Motion: Archives, Technology, and the Social*, ed. Ina Blom, Trond Lundemo, and Eivind Røssaak (Amsterdam: Amsterdam University Press, 2017), 89.
[86] Ibid.
[87] See Adrian Mackenzie, "What Is an Important Event? 175 Million Events and Counting. Notes for Public Lecture at It University of Copenhagen" (https://github.com/metacommunities/metacommunities.git, March 5, 2014).

to track all these, could have the potential to provide a rich "account of how people move through code,"[88] and generally speaking the reason that leads scholar Adrian Mackenzie to argue that "software today is less like a machine, a system or even an assemblage, and more like a crowd."[89] But given everything discussed so far in this thesis—from the proto free and open source era of computational culture, its strange modes of organisation and the UNIX fellowship, and of course the Cambrian explosion of free and open *things* triggered by free culture—this analogy to the crowd could easily apply since the early days of code sharing. In fact, private forks and exotic code-hosting platforms are nothing new, but GitHub contributes an authoritative centralisation and forced visibility of such practices. The shift is not so much from machine to crowd, but—and expanding on Mackenzie's urban metaphors—it is the transition from rural coding communities to the coding city crowd, through the means of the *Gemeinschaft* emulation originally triggered by the use of free and open source techno-legal templates. But more importantly here, this crowd is in fact trapped. While GitHub provides very effective, and easy to use, tools to facilitate the self-organisation of communities around one single repository, there is a catch. To permit the construction of extra systems on top of the `git` DVCS the repositories are forked *within* the GitHub platform, thus revealing the irony of centralising a completely distributed system into one giant... sandbox, where almost one half of

[88] Adrian Mackenzie, "Code-Traffic : Code Repositories, Crowds and Urban Life," in *Code and the City*, ed. Rob Kitchin and Sung-Yueh Perng (London: Routledge, 2016), 86.
[89] Ibid., 87.

the repositories are forks from other repositories.[90]

While networked decentralisation has been perceived as an empowering instrument, as best exemplified with Dmytri Kleiner's Peer-to-Peer Communism vs Client-Server Capitalist State,[91] the techno-legal mechanisms that permit such decentralisation have been greatly overlooked. In retrospect, it is clear that when P2P rose to popularity, it first appeared to provide a lightweight, democratic, and nomadic alternative to the client-server models of transactions and capitalisation, but that was however not counting without how this new model could also be embedded into other systems of different nature. This is once again very well illustrated with GitHub and shows that no matter what is the topology of network labour, there will always be opportunities to create overlapping structures to control and capitalise it. In the case of GitHub, this capitalisation is moved to another level. What has escaped from the control of macro-liberal/micro-communal groups is now collected by this platform, a new form of browser-assisted massive local file system source control à la CLEAR-CASTER, a shared and collaborative file-sharing app for programmers in the age of Internet turned into an Operating System,[92] worse, a download site.[93] Similarly, it is possible to witness how the *yielding* effect suggested by Raymond, can be captured by a platform

[90] Adrian Mackenzie, "Large Numbers: Imitative Fluxes in the Data-Material Imaginary. Notes for Material, Visual and Digital Culture Research Seminars 2015-16, University College London" (https://github.com/metacommunities/metacommunities.git, February 1, 2014).

[91] Kleiner, *The Telekommunist Manifesto.*

[92] In reference to Tim O'Reilly, "The State of the Internet Operating System," 2010, http://radar.oreilly.com/2010/03/state-of-internet-operating-system.html.

[93] Mackenzie, "What Is an Important Event? 175 Million Events and Counting."

like GitHub. It does not matter what the yield is and it certainly is a variable element, but while the *use* of software can escape GitHub as easily as with a clone command, its context cannot be extracted from the different additional social and technological features that GitHub has built around the popular DVCS. In the process the licenses are replaced with Terms of Services,[94] and the employees and founders of the platform, whose core components are strategically closed source,[95] are the ones to decide what projects and behaviours are acceptable. They establish a nearly feudal meta-model of governance on top of the communities and groups they host, occasionally taking advantage of their overarching landlord position, thanks to the newly-acquired virtuous property of forking, to directly tap for their own benefit into the gigantic pool of disposable code they host, regardless of the damage this creates to independent programmers turned sharecroppers.[96]

[94] A recent study in 2013, even if it was essentially simple data scraping, showed that out of nearly 1.7 million code repositories on GitHub, less than 15% had a license. See Neil McAllister, "Study: Most Projects on Github Not Open Source Licensed," *The Register*, 2013, http://www.theregister.co.uk/2013/04/18/github_licensing_study/.

[95] Tom Preston-Werner, "Open Source (Almost) Everything," 2011, http://tom.preston-werner.com/2011/11/22/open-source-everything.html.

[96] For an example of such abuse see Aymeric Mansoux, "Fork Workers," in *Are You Being Served?*, ed. Anne Laforet, Marloes de Valk, Madeleine Aktypi, An Mertens, Femke Snelting, Michaela Lakova, and Reni Höffmuller (Brussels: Constant, 2014).