

Five. GNU GPL

1. Grafting an Origin into a Stream of Events

Let us remind ourselves a brief sketch of the events that related GPL with Linux kernel. We consider this convergence as a break-point in the history of FLOSS. Opinions abound that GPL was never that important for Linux, that Linux could very much happen without GPL, that is, with another kind of software license. Anyway, this whole discussion is meaningless in the sense that, this whole ‘Linux’ thing is a living process and not a theoretical category, a real living process of becoming, and Linux now is something that Linux became, and Linux became because of what it was at the level of possibility. And Linux kernel *is* GNU-Linux, licensed under GNU GPL, from kernel version 0.11. Without the childish game of prognosticating in the subjunctive, and trying to guess what could happen with *another* Linux that is not GNU-Linux, it can be very well said that, the break in history, the thing that we want to understand theoretically in this book, does precisely reside in the very liaison between GNU and Linux. And obviously, this break could never happen if there was no GPL. This book specifically talks about the philosophical, ethical, economic and political implications of this break: a break that would become a continuity in terms of carrying on the FLOSS tradition from its primitive phase to the declared phase.

Both the mails from Torvalds to comp.os.minix in the last chapter show that the efforts of Stallman were already very much present in this reality, when Torvalds and other fellow hackers were trying to build the new OS. The efforts of Stallman and all, in the form of GNU, and the development tools that GNU was building, had created an ambiance. This ambiance did involve an implicit value judgment. It was not just the efficiency and possibility of the hacking tools that GNU was generating. Thanks to all the efforts of Stallman, the hacker’s viewpoint was already witnessing a spectrum of possibilities in terms of Father’s rules or their absence. All these efforts under the leadership of Stallman and GNU were generating small supplements in the form of resistance towards Father’s rules. These supplements would all later precipitate and find their ultimate shape in GPL. Father’s rule: this is just a metaphoric way of mentioning the omnipotent rules of capital and market. This metaphor suggests the possibility of a kind of father-hostility immanent in the ‘flower-power’ times, and makes it metaphorically possible to posit a counter-father in theories of counter-hegemony, that we will explore later, after our analysis of GPL is over.

There already was a divide. For some, the Father’s rules were just to follow and foster. And for some others, the omnipotence of the rules did hardly rule out their questionability. Many innumerable small events, which are now a part of the folklore of computing, were generating this divide. One example can be cited here, in the form of a mail, on 3rd February, 1976, by William Henry Gates III, later to be known as Bill Gates, one of the most authentic representatives of Father’s rules. Father’s rules are the rules of capitalist rationality: the rules of a rationally behaving market generated by capital. This is the very rationality which we will question here in the later parts of this chapter. We will show how, all the sub-processes generated through the stance of resistance towards these rules merged into the emerging text of GPL, and GPL then proceeded to build another alternative rationality much bigger than capitalist one. We prefer to call these sub-processes as supplements, in compliance to the theoretical scheme of a subversion of the context-text-supplement politics elaborated in chapter two. These supplements then started snowballing,

and gathering momentum, all contributing into the root process of GPL, which was in the process of making. So many very dramatic and deep-acting things happened around GPL, that we are going to know now. Though, for their proper philosophical interpretation we will have to wait till the next chapter. That GPL did it unknowingly, without a knowledge of what is going to happen in the future, is the very beauty of the thing. This, in a way, vouches for the rootedness of this whole phenomenon in the heterogeneous and polymorphic social reality.

Anyway, let us come back to the mail of Bill Gates. This mail from Gates had a heading ‘An Open Letter to Hobbyists’. This word ‘hobbyist’ was the the then word for the sense in which we are using ‘hacker’ – one who creatively works with source-code and programs. Here, in this mail, Bill Gates took a position against the hackers, whom he called ‘hobbyists’. Gates, or more formally, William Henry Gates III, in this mail was extremely rational, and correct too, in his own way, in terms of the rationality of capital and market. Let us just quote here a question from this mail, Gates 1976. This very question has a point for us.

What hobbyist can put 3-man years into programming, finding all bugs, documenting his product and distribute for free?

The question is quite credible and justified too, in terms of a reality that knows only one justice, that of market and capital. We wait till the next chapter to see that, how through layers after layers, this justice of market and capital materializes into the very institution of state and civil society. Anyway, this sense of justice came from the sense of reality that Gates represents. In the reality of Gates, this is quite unjustified. And exactly here rests the differend of GPL, as we said in chapter three – the differend between two fighting senses of justice which can never meet or coincide. The justice that GPL grappled with, the justice of freedom and sharing of knowledge, is something altogether different. It is so different that this reality of Gates does not even have any clue to anticipate, understand or foresee it. See, FLOSS is the very answer to Gates question. FLOSS *has* demonstrated, exactly the answer of this question, that yes, all these FLOSS hobbyists can put millions of man-years into programming, finding all bugs, documenting the products, and distribute them for free. Gates framed the question this way, because it was an impossibility in terms of his reality and his understanding of that reality. And it is a very living truth in terms of the reality and understanding of the reality that GPL represents.

Now, at present, this dichotomy of the reality, the divide beyond which an impossibility becomes an actuality, is complete. We have seen what the rules of market *cannot* and what FLOSS *can*. But, in this chapter, we are talking about a time when this context of FLOSS was a long way to come, even GPL was not written. But, even in this reality, this divide of differend was getting bigger every moment. The distance between the two versions of reality was increasing. We are calling it ‘version’, because actually it is the same reality, so the thing that finally matters is the understanding this reality. The understanding of the reality, together with the logic and the sense of justice inherent in it, then started talking back on the real existence. Years later all this accumulated talking back will take the form of GPL and FLOSS. But, in that time, many small events, like this mail of Gates, were happening every day in the real existence. These small events were generating the polarization that we mentioned.

The polarization of the hacker community for and against this divide is very important here. By the time the hour of choice arrived, the spectrum of choice was already created for the hackers, about which way to take. GPL enabled this final moment of choice to emerge. The choice was between two ways of seeing, and hence, two methods of learning to live in this reality. Actually, the number of possibilities is always potentially infinite, potentially plural. But, at a particular moment of history, the choice happens between the ruling way of life and one of its many plural alternatives, the one alternative that becomes the most possible alternative at that time. Here, the existing way of life was the way that went with Father's rules, rules that overlook one very important injustice against freedom and openness and intends to become 'rational' in a capitalist way – or, the other way, the 'irrational' way that wants to correct this injustice and so goes on against the Father's rules. It was for the hackers to decide now. And as we will see through this chapter, it was GPL that enabled this FLOSS way to become the most possible alternative through accumulation of innumerable number of small supplements, some of which we discuss in this chapter.

This choice thing is important, as we said. At any point of time, the number of possible actions is potentially infinite. From this infinite spectrum of choice, which way to choose depends more upon the moorings of the time, because there is no predestined path to follow. The efforts like GNU Project or Emacs, in the form of supplements of resistance, were tweaking the possibility horizon, thus making and strengthening *freedom* into a major variable, and narrowing down the choice horizon around the issue of freedom. In some other case, without this history of polarization working in the backdrop, the ruling variable could become, say, 'fun and fun only', or 'money', or 'crack government computer systems' or something like that. A choice that way would never mean that the person making the choice is 'too casual' or 'too selfish'. In actual history, in the aftermath, it may indeed turn out to be 'too selfish' or 'too casual'. But, that is the *aftermath*. At that moment, in real living time, it may very well happen that his thought plane, the possibility horizon that he could envision, did not have some other possible alternative variable at all, or at least not with a sufficient strength to pursue him into that direction. His choice *was* casual, because he could not *see* any serious alternative, maybe.

And once choice is made, it pushes and tweaks the reality in a minuscule way. That then opens up another spectrum of choice. This goes on in a series. In real life, this actually happens so many times. This seems more like a science fiction if narrated like this, that, one small action gets attached with another, another, and so on it continues, till millions of more actions get connected, and make something happen that anyone could never anticipate beforehand: the famous 'butterfly effect'. But it really happens that way. And when some tangible amount of accumulated tweaks come up in the form of a changed context, we start to discover the change, and then finally we try understand the history of this change. Now, histories start getting written with the end-product in mind, as if there was all through a teleology, as if all through the real process the end-point was the precise thing that motivated everyone towards it. But it happens hardly that way. Things go on happening more according to the seemingly important spectrum of choices at every moment, and if some of the actions, the products of these randomly made decisions, happen to fall in the same direction, a direction that would, later, after many more actions, facilitate the end-product, it becomes known and famous as 'The Origin'. All origins are

mythologies in that sense.

This same thing happened everywhere, and particularly, in both the groups of actions that led to GNU and Linux respectively. Let us check a few elements of history of how this ambiance was created that made Torvald's choice possible, explicitly from version 0.11 of the Linux kernel. Even before that, implicitly the choice was there in Torvald's own license attached with an earlier version of the Linux kernel. The GNU website <http://www.gnu.org> and the FSF website <http://www.fsf.org> provide all necessary materials from where we are going to collect a few elements here. In the definition 'What is Free Software', Lee 2010, the FSF document, mentions the start of the whole thing like this:

The free software movement was started in 1983 by computer scientist Richard M. Stallman, when he launched a project called GNU, which stands for "GNU is Not UNIX", to provide a replacement for the UNIX operating system – a replacement that would respect the freedoms of those using it.

Now, let us compare this time-stamp '1983' of the action that "would respect the freedoms of those using it" with portions of an email from Richard Stallman, sent to the newsgroups *net.unix-wizards* and *net.usoft*, on 27th September 1983, at 12:35 AM. This is a long mail, with sub-headings marking sections, after a subject of "new Unix implementation", and a heading 'Free Unix!' for the first part. But we are quoting here just some small strings relevant to us, getting rid of all the formatting, and plucking out the portions relevant to our discussion through a generous use of ellipsis. This document, Stallman 1983, is available in many places, in the form of an essay, both in hard and soft copies.

Starting this Thanksgiving I am going to write a complete Unix-compatible software system called GNU (for Gnu's Not Unix), and give it away free to everyone who can use it. ... GNU will be a kernel plus all the utilities needed to write and run C programs: editor, shell, C compiler, linker, assembler, and a few other things. ... everything useful that normally comes with a Unix system, and anything else useful. ... GNU will be able to run Unix programs, but will not be identical to Unix. We will make all improvements that are convenient, based on our experience with other operating systems. ... Both C and Lisp will be available as system programming languages.

I am Richard Stallman, inventor of the original much-imitated EMACS editor, now at the Artificial Intelligence Lab at MIT. ... I consider that the golden rule requires that if I like a program I must share it with other people who like it. I cannot in good conscience sign a nondisclosure agreement or a software license agreement. So that I can continue to use computers without violating my principles, I have decided to put together a sufficient body of free software so that I will be able to get along without any software that is not free. ... I'm asking individuals for donations of programs and work. ... Individual programmers can contribute by writing a compatible duplicate of some Unix utility and giving it to me. ... If each contribution works with the rest of Unix, it will probably work with the rest of GNU.

So, what Stallman looks forward to is something that is, so importantly, as mentioned in the very name, ‘Not-Unix’. Because, that OS called Unix has gone the proprietary way by that time, as we know. And for this OS, proposed by Stallman, the goal is to “give it away free to everyone who can use it.” And this he intends to be a full-fledged OS, and hence it will need the kernel and all the things from the ‘central layer of system programs’ as we mentioned in chapter three. This is essential for this OS, because the goal is to “make improvements” of this Unix-like OS, “based on our experiences of other operating systems”. And in order to make these improvements possible one needs a continued process of hacking on the OS for which one needs those tools. Stallman, the inventor of ‘EMACS’, then announces the ‘golden rule’ too: “share it with other people”. Then this mail says a few things about the license issues that will get clearer from our later discussions, and invites people for “donations of programs and work.” The very last statement is very important to understand the special status of the GNU way of doing things. This GNU way is already assuming a *surrogate fatherhood* of all the loose *fatherless supplements of resistance* by this time, 1983. This envelope status of the GNU way will reflect in the chain of events that leads to Torvald’s choice of GPL as the license of the Linux kernel, rejecting the first one written by himself for the earlier versions. This envelope or blanket function of GNU was already there, that will actualize to its fullest in the form of GPL, a bit later. Here, in this mail, Stallman projects this GNU dream as something where all hacking will start to find the haven of its cause and effect, the cause that motivates a hacker, and the effect or the work of hacking that he does. “It will probably work with the rest of GNU”, such that, relating with the rest of GNU, it becomes a part of a *surrogate* whole, in place of the *original* whole put forward by the ‘rules of father’, something that they have already rejected.

The role of Emacs in the context of this mail is not only to legitimize the worth of the man writing the mail in an envelope effort of such a scale. Emacs had a very crucial role in forming and formatting the thoughts about the licensing process. Emacs has a very long history. The development of Emacs started in the seventies, and it continues till now, together with XEmacs, a fork that started from the GNU Emacs from 1991. And a lot of things that went into the making of GNU came from Emacs. But, first let us start from some elements of the history of GNU as written by Richard Stallman in his essay ‘The GNU Operating System and the Free Software Movement’, Stallman 1999. In this essay Stallman mentions MIT Artificial Intelligence Labs, where he joined in 1971, as the first Software-Sharing Community. And *this* is important. We can use here an analogy from Indian folk and mythology, where the demon’s life is kept somewhere else, outside the body of the demon, usually in the form of a bee in a secretly and well guarded coffer. In the age after Nietzsche, Nietzsche 1990, when we are no more innocent devotees of the enigma called knowledge. We know it fully well that the will to knowledge is actually a will to power. We consider knowledge itself as power. And in that sense, the soul of the Father, the soul of capitalist hegemony, is kept elsewhere, in the high-flying research centers and academies. And, at that moment in history, there was no higher-flier than MIT AI Labs, the coffer where Father’s soul was kept. And the very fact that it worked as a software-sharing community means that till that time the gaze of capital, the gaze or *Panopticon*, had not yet zeroed in on hacking. The ‘panopticon’ is the gaze of power working ceaselessly at every moment everywhere on everything within the jurisdiction of power. Here we mean the gaze of capital, working through market and civil society, by the

institution of state. Why and how we are relating market, civil society and state with gaze of capital will be elaborated in the next chapter, where we use Hegel's logic to understand the phenomenon of GPL. This concept of *panopticon* actually belongs to Bentham. Foucault gave it a philosophical interpretation, Foucault 1977, and applied it in context of capitalist society.

2. Stallman's Text and Evolution of Resistance

Let us remember the flow of time. Stallman joined MIT AI Labs in 1971. The attacks of AT&T and Sun and others on the primal and innocent freedom and openness of hacking was yet to happen. But, very shortly, within a few years, power would start enclosing these common pastures, like the hackers community residing in AI Labs. We are using here the age-old analogy of 'common pasture' to remind us the Marxian Discourse of 'primitive accumulation'. In this theory, the primitive accumulation of wealth takes off into the process of becoming capital through enclosing the common pastures and thus accelerating the process of the expropriation of the peasants and artisan producers. Through this expropriation comes the so-called 'double alienation'. One, alienation of the primary-producer from his self-owned means of production like land. And two, alienation of the laborer from his own labor-power. From now on he will sell his labor-power as a commodity, which earlier was controlled by his own free will. The similarity of the two situations, enclosing the land and enclosing the till-then free and open knowledge in computer science is too strong to overlook it. This maybe another reason why so many times, and so wrongly, the 'Marxist' tag get attached to a lot of FLOSS theory. In the coming chapters of this book we will return to one such big mistake by the master programmer Eric Raymond. Anyway, let us return to what we were saying. At the time Stallman was working in MIT AI Labs, the whole software-sharing community, together with Unix, was going through a very important change. This change will later become so integral to everything that is relevant to FLOSS tradition in particular, and computer science in general. During 1972-73 Unix was getting written in C. The ready significance of this is that Unix becomes portable. But there are even deeper impacts of this than just portability.

As we have discussed in earlier chapters, *portability* means the ability to run a software on a second set of hardware. OS is the primal program that runs on a hardware and enables all other later programs to run. This primal program of OS has to now run on the changed hardware, and enable all other following programs to run on it. Running OS on a new hardware means a new set of drivers that talks with a new set of devices, and a set of libraries that can enable this transport, and everything else related with it. And once OS is running there on the new set of hardware, all other programs follow in line. In times before portable OS, this would mean rewriting the individual programs. But, in case of portable OS, rewriting is no more needed, because, they are already once written for the same OS. So, if OS runs, the program will run too. It is no more needed to rewrite the programs from one system to another system. The job is now to make the system, Operating System, OS, flexible enough to run with different sets of hardware. And the programs are just made for this OS. So, in a way, the very concept of 'system' is undergoing a change here. Essentially, the ground of the definition is shifting from hardware to software. In earlier

cases, all individual programs had to cope with a change of hardware. Now only OS has to do it. And all other programs, once recompiled on this new system, enabled with all the new drivers, will run this new system too. So the thing that these individual programs have to keep in account is OS, not hardware any more. Tanenbaum 2005, “Structured Computer Organization”, elaborates a lot more finer details on this issue.

We have discussed, how at one stage of the evolution of hardware, everything was done the hardware way, plug-boards and all. Gradually the hardware evolved enough to do the execution through software, that is, in terms of commands that will be executable by the hardware. And Unix getting written in C was the next and final step. The whole OS, together with all the later programs to run on it, now enjoys an added fluidity of a *discursive space*. This is a very interesting and crucial point. A few important steps are involved here. One, the whole space of source code becomes a space written in the same language, C. And so, the whole space has an uniform discursive structure, where all the details of the hardware have become differences of variables and parameters. Two, the whole discourse has two distinct halves, the OS half, and the Application half, each half with a relative autonomy and specialization of its own. While the focus of the OS half is on manipulating hardware, the attention of the Application half is concentrated on the running of the application programs, taking the OS half source code as given. Three, the OS half has its discursive fluidity in the sense that, with a change in hardware, the whole source code can get edited and tweaked to make the OS run. The sole motto of this half is to run the OS. Four, the Application half takes off with the OS half taken for granted, so each can specialize on a zone of its own. For the OS half, this zone is hardware, and for the Application half, this zone is the given OS. There can obviously be minor exceptions of this discursive structure, when any half has to take into account something from outside its zone. But, certainly, this whole discursive space, written in C, with its two relatively separate halves, has an added fluidity, as we said. The total portability of both the halves, made the whole discourse of source code very general and all-pervasive. Take any hardware, and the OS half, through editing and compilation, will make the Unix OS run on that. And the Application half can forget the OS half altogether, starting from their project of enabling the application programs to run on Unix.

The written source code is a piece of text. All the statements there stand for real actions, to happen on a real machine with a real set of hardware. The whole collection of actions, including both the domains of OS and Application, is now text. Every action is a text, written in a high-level language meant for human beings. It is a text that can be read, annotated, analyzed, edited and debugged – everything that can be done with any text. And most importantly, after becoming a text, it could be transferred anywhere from the machine it was running on. Think of a real OS running on a real machine, one cannot abstract the OS from the machine in its real form. But, as it becomes a text, it can be abstracted. And likewise, anything can be done with it, exactly like a text. This process of a collection of real events transforming into a collection of editable texts, together with the availability of the mother text, the source code of Unix, signal the very break that makes source code so crucially important in the history to follow.

Now that the whole thing became a collection of texts, it became infinitely easier for Unix source code to go beyond all the barriers of hardware. It could now even get transmitted

over the Net too. It could go anywhere and start getting transformed. And more than that, now it could go beyond itself, and go on reproducing itself in a multitude of polemical genealogies, and continuously go on generating innumerable subtexts. Hacking was the machinery: the act of writing of new offspring texts, through which the mother text could not only now mutate, but evolve beyond itself into newer and newer species. This is what happened in the case of the whole FLOSS genealogy. It is not that this evolution would be absolutely impossible without this discursive rendering of the hacking space. But, in patterns emerging over time, it has facilitated the tendency of factors coming together in ceaselessly generating new possibilities. The availability of the mother text, Unix source code, opened up a Pandora's box, in a very positive way of course. So positive that maybe Forrest Gump could call it his *Mama's Box of Chocolates*. Everyone now started writing new texts by reading, understanding, editing, cloning, copying, modifying, improving, partitioning, and augmenting the mother text. Let us quote once again a paragraph from Stallman's experience in MIT AI Labs. As we said, we are going to quote quite a lot from this essay, 'The GNU Operating System and the Free Software Movement', Stallman 1999.

We did not call our software "free software", because that term did not yet exist; but that is what it was. ... If you saw someone using an unfamiliar and interesting program, you could always ask to see the source code, so that you could read it, change it, or cannibalize parts of it to make a new program.

So, as we said, FLOSS was already there, without the name though. The motifs were all coming up. All these motifs would come together to form the very context where GPL gets written and becomes the choice license for Linux kernel. Before C and Unix, the FLOSS practice was already there. Though all those activities were without a focus. They were happening in scattered ways, with different programs. Because, a full scale OS was still lacking. And as we said, the whole discursive space of hacking does not get its full bloom before the mother discourse of the OS is available, before the double birth of Unix and C. So, it was hacking alright, FLOSS hacking too, but, it still lacked the whole vitality that we witness there later, with the arrival of Unix source code in C. We have reported the details of the stream of events around the birth of Unix, in the last chapter. Once Unix source code in C was available, the realm of hacking was potentially complete. It became ready to evolve, the mother text being available and the hackers ready to generate generations of offspring. Much before the name 'FLOSS' came, this was the realm of FLOSS hacking, going in full blast, with Unix source code in C, complete with all the necessary component parts. And then entered the rules of market and capital into this realm, starting to take away the free flow of source code, thus truncating the process of evolution and all. The primitive FLOSS started breaking down. So came the resistance against this taking away of the primitive freedom and openness. The supplements of this resistance started accumulating. This accumulation of the supplements will finally lead to GPL, through which the final context of FLOSS will come into being. This FLOSS will no more be primitive, it will be very adequately guarded and protected by GPL.

The second section of Stallman 1999 is titled "The collapse of the community", with an account of how the "situation changed drastically in the early 1980s." We know from the last chapter, the inception of this process of drastic changes started in late seventies, in

terms of transformations happening in both hardware and software, even in the market of OS. Stallman writes here, “The AI lab hacker community had already collapsed.” This was the exile of primitive FLOSS from the strongroom of power. This process of exile would then foster through the whole space of computing. The primitive accumulation of the primitive nameless FLOSS was complete. It was time now for the merchants to rule, loaded with the value-judgments of capital and market. Stallman gives us an account of a series of events through which a situation got created where “first step in using a computer was to promise not to help your neighbor.” Primitive accumulation was over, and it was time for the competitive market to rule, where every entity is defined in terms of a competitive positing of other entities. This is a space where enmity rules, the day of friendship is gone. We would come back to this theme of enmity and friendship later, in the next chapter, where we will relate GPL to a context created by GPL, where the two binary opposite categories of ‘enmity’ and ‘friendship’ no more remain binary and start to overdetermine each other. But, now let us quote once again from the same text, Stallman 1999.

The idea that the proprietary-software social system – the system that says you are not allowed to share or change software – is antisocial, that it is unethical, that it is simply wrong, may come as a surprise to some readers.

We see here a definite pronouncement of the differend of social justice, that we are talking about from chapter two onwards. We chose this portion for another reason too. There is a suggestion of resonance of a Marxist logic here. This is the sentiment that the FLOSS logic shares with Marxism. We deliberately chose here the phrase ‘FLOSS logic’ in place of ‘FLOSS philosophy’ to point it out that this FLOSS logic is a practical workaround of a fix, and it is no way a conscious journey into philosophy. The FLOSS tradition was trying to find a practical solution of a very painful real life problem. But, in achieving the solution of this problem, GPL actually goes through a violence on the logical categories of the institution of state. We are coming to that later in the book. But, let us remember, this whole thing happened in a ‘practical’ way, not a ‘theoretical’ one in the sense that, it was not that someone was trying to solve a crisis deep down into the logical categories and got GPL. And so we prefer to attach the tag ‘unconsciously’ to the action of arriving at GPL. The arrival at GPL was a long process of continued efforts, a long series of supplements, and quite a lot of these efforts shared the sentiment reflected in the above quote. This sentiment goes quite in line with a similar sentiment reflected in a lot of Marxist positions. The Marxist position proclaims a similar indignation at some people being so very innocent in not understanding that the very system of justice in a market society is founded on a basic injustice. We already said a few things about it in chapter one, but let us mention it here once again.

This injustice, from the point of view of a Marxist reading of history, comes in layers. One, the injustice that the people whose labor create new things, new goods, new wealth, are just paid an amount proportional to the amount of labor given. This is just the amount needed for their subsistence, and maybe marginally a bit more, depending on the power of negotiation between labor and capital. And, by virtue of this subsistence payment, the owner of capital, takes away everything from the owner of labor-power. Two, the means of production or capital is, in the true sense, nothing but accumulated labor. Logically this is

quite clear. Human labor started generating the first ever tools and machines working on the raw nature. These tools then helped labor to create the second set of machine, and this went on for millions of years. Through this time, all these machines, and all the wealth generated through these machines, got accumulated in the body of capital. And so, a gross injustice resides there in a personal, not social, ownership of capital. Capital is, by definition, social: accumulated social labor through millenniums of civilization. This unjust ownership of capital now gives the owners of capital the opportunity of exploiting the people that are selling labor-power from the products of their own labor. Three, so in a market society built on this major injustice, the very values this society upholds become some false values. For example, the slogan of freedom proclaimed by this society becomes a freedom for the seller of labor power to starve and die if he does not go the way the owner of capital wants him to go. And it remains a freedom to hire and fire at will for the owner of capital. This list may go on, but the point is, the Marxist position carries a similar sentiment of disapproval at the ‘stupid innocence’ of the people who do not understand that the market society is based on the breach of those very principles that it preaches. Maybe this similarity of sentiment reflected here in the quote of Stallman led so many people to believe that there is something Marxist there in GPL and all. Anyway, we would come back to this in a big way, in our coming chapters, where we show that the new social categories imbibed in GPL are not only extremely different to Marxism, but actually far bigger and richer in scope than the whole Marxist cannon.

Carrying this indignation towards the so-called ‘state-of-the-things’, Stallman now goes on to list the unstated social assumptions implicit in the actions of companies owning, and thus ruling, software. The list of unstated assumptions goes like this. One, the assumption that, companies can own software naturally. Stallman states this to be untrue, and according to the view of US constitution and legal tradition, copyright is an “artificial government-imposed monopoly that limits the users’ natural right to copy”. Two, the assumption that, the only role a piece of software has is to do the job it is supposed to do, and it has no social, ethical or political part to play. Very soon, all Stallman’s efforts in GNU and FSF, and then the whole FLOSS flow of events, will challenge this mechanical view about a piece of software. Three, the assumption that, without a company to create them and thus assert a monopoly power over the use of them, there would be no software. MIT already was demonstrating the negative of it in a small scale, and very soon FLOSS was going to disprove this assumption entirely.

Stallman invites the readers to decline to accept these unstated implicit assumptions, and takes resort to, as he says it, “commonsense”. But, very soon the efforts of Stallman are going to outlive all forms of commonsense. GPL is soon going to attack these implicit assumptions and the system residing in these assumptions with a ploy that applies a very uncommon sense of recursive logic. The use of the phrase “unstated assumptions” is very important here, prefiguring the jump of GPL from the realm of real existence into its play with logical categories. This same thing is going to happen in the very real situation of learning to live and hack in a world that the hackers did not make. It would come as a ploy to cope with this world of ‘unstated assumptions’ where the hacker cannot live anymore. Either, bow down to Father’s rules, or leave the game altogether – “Another choice, straightforward but unpleasant, was to leave the computer field.” In the use of the phrase “unstated assumption” by Stallman, there remain some implicit pointers to the logical ploy.

When something is called ‘unstated’, there remains a concept of stating it somewhere, in some place where it is supposed to be stated – in some discourse. Later, we will discuss in full details this discourse, the discourse of the institution of state that grows around the entity of ‘private property’, as we will show in the next chapter. These assumptions are all parts of that discourse of private property and state. GPL is going to challenge this very discourse by a self-recursive loop in the category of private property. Maybe Stallman is not at all conscious in this philosophical way, but, the pointers are all spontaneously there, scattered in these supplements, ready to be read by anyone who can read them.

When all these supplements of resistance are starting to accumulate, in terms of looking in the eyes of the Father, or the hegemony of capital, there was already a model, that was the Marxist or Communist cannon. A cannon that projects an alternative father, say father[/]. This father[/] is the project of counter-hegemony, as we said in the first chapter. The project of all counter-hegemonic politics of the Marxist order resides in replacing father with father[/], and rebuilding the society around this father[/]. As we will see in the coming chapters of this book, in a counter-hegemony, in the act of replacing father with father[/], father is not inherently transformed, just the ownership of capital changes hands. So it becomes capital[/] owned by father[/]. We will call this property as property[/], later in the book, in contrast to the category of ~~property~~ generated by GPL. We will put this category father[/] to a lot of use, when we come back to the concept of counter-hegemony later in the book.

Such a project of the communist politics is nothing unusual. In the fight against father, against his injustice, communist party wants to create an alternative father[/], a structure beyond this injustice. The embryo of this would-be structure, for the time being, is situated in the party system, giving an address to all the forces of resistance growing within this society. Grounding on this embryonic structure, the communist party then plans to go into a full-fledged war against father. And, thus comes revolution, through which father[/] becomes the winner, and hence replaces father. In case of communist party, father[/] is socialism, for the time being, to evolve into communism in the long run. Till father is overthrown, the embryonic structure of father[/], residing in the communist politics, is the haven of justice and struggle against injustice. Father[/] is a blanket concept, an envelope, that holds everything that is good and productive till the hour of overthrow, and after it, the shape of things to come. The party is the tool, the resort, the struggle, and the address of the future that the party itself is going to make happen. Now let us quote a few lines from the same text, Stallman 1999, that may remind us something like this envelope concept of the projected father[/] in terms of hacker’s fight against the market rules .

The answer was clear: what was needed first was an operating system. That is the crucial software for starting to use a computer. With an operating system, you can do many things; without one, you cannot run the computer at all. With a free operating system, we could again have a community of cooperating hackers—and invite anyone to join. And anyone would be able to use a computer without starting out by conspiring to deprive his or her friends.

Stallman mentions it as the moral choice on which GNU was founded, the projection of the things to become. The genetic traits of father[/] shows there clear and distinct. But, then something happened there very dramatic. And this drama was so full of self-recursive logic

that it reminds us of nothing but the very acronym GNU: GNU's-Not-Unix. The twist becomes more distinct if we write it as a statement: GNU is GNU's-Not-Unix. The statement starts to say what GNU is, and ends by declaring what GNU is not. A similar twist resides in the very working of GPL, it defines property by what property is not: we will discuss it in the next chapter. The dramatic twist residing in GNU GPL changed the whole projection like anything: something unprecedented in the rich history of human struggle against power. Something that would transform the projection from father to ~~father~~: this is something never seen before in human history, as we will elaborate later. This enabled GPL to become a real blanket of all counter-culture, without becoming a practice of the culture of father'. We are coming back to that later. We used these character-icons to remind us of the difference between the Marxist projection and the GPL projection: father' in the Marxist case, and ~~father~~ in case of GPL. While in the Marxist case resistance replaces father with an alternative one, GPL projects an alternative that builds from some internal subversion, an extremely deep-acting subversion, and start tweaking father from within. But, to discuss the logical details of this politics of subversion, we have to first go through the history of the rise of GPL.

3. History of GPL

Stallman faced the workings of proprietary licenses in quite a hard way in the episode of the printer, given in details in the Chapter 1 of the book, "Free as in Freedom: Richard Stallman's Crusade for Free Software", Williams 2002. An incident that showed the potential danger implicit within an apparently innocent nondisclosure agreement. A nondisclosure agreement is connected with some sort of trade secret. It is a pretty common ploy in the realm of market. But, as the event demonstrated, it carried within it a serious breach of the human right to freedom of knowledge. It took a push from the hacker world mind-setting to reveal this breach. When Stallman wanted to correct some flaws in the control program of a printer, the nondisclosure agreement blocked it. And the concern selling the printer refused to divulge the necessary information. What was actually blocked here? It blocked the continuity of knowledge, and the hacker mindset of Stallman, that believed in freedom and cooperation in the realm of knowledge, reacted against this block. Someone more market-minded in that sense, like the mindset implicit in the mail of Gates we quoted before, may very well could take it for granted. But, Stallman started interrogating it, and that too, quite actively. Let us remember whatever we said about the accumulation of a series of supplements, in each of which a fatherless bastard text, a small quantum of resistance, started accumulating. Each of them tried to resolve the differend that cannot be resolved, at least within the given legal framework. All these quanta collectively contributed into the later process of FLOSS to emerge, as we shall see.

And the thing to note here is that, already the legal and ethical are coming to the differend. As we described in the context of Lyotard 1988 in chapter one, the ethical claims of the hacker world is already considering itself a *victim* in the *differend* of the legal mechanism of a pretty common nondisclosure agreement, when someone like Stallman pushes the contract hard enough. And this nondisclosure agreement was a legal ploy. Let us attend this point: a legal ploy is a social and ethical apparatus, one of many through which justice is distributed among people. That very justice mechanism was creating injustice here, self-

recursively. And in this case it was a terrible irony, operating against a community that would later generate GNU: a whole organization defined self-recursively. An organization that would employ another self-recursive loop in generating GPL that goes on snatching the whole institution of capital and state from within. GPL is the biggest self-recursive logical ploy since August Kekulé's self-devouring snake that bred carbon bonds, the self-recursion in-built in the *viral nature* of GPL.

When using this phrase *viral nature*, I am very conscious that it is a phrase popularized by a section of media under direct supervision of monopoly powers in software industry, as part of a diatribe against GPL. Here the use is a counter-gesture, as a tribute to the sheer force implicit in GPL, a force of such an extent that can only be given a biological analogy – the silent invincibility of a virus. Let us come back to the context of Stallman's fight for the lost freedom of the hacker world. So, once it was getting taken away, it was time now for hackers like Stallman to reclaim this freedom in the field of software creation. But, what exactly was the shape of this freedom? In Stallman's own formulation, this freedom is actually a composite freedom. We will come back to this in details later, but for the time being let us listen to what Stallman considers as 'freedom' in Stallman 1999.

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

But, the whole event chain that led to the confluence of Linux kernel and GPL, and thus made FLOSS happen, that would serve as a context of reading the text of GPL in a book like this, was still a long way to go. And this particular formulation of freedom was written by Stallman much later. In the true sense of the term this is a formulation of freedom from the future, when we already know what has happened. I could not be sure about the exact time of writing of Stallman 1999. The O'Reilly book "Open Sources: Voices from the Open Source Revolution", DiBona, Ockman, Stone 1999, mentions the year 2000 as the copyright year, while the GNU web-document mentions it as 1998. So, whatever it may be, 2000, 1999 or 1998, it can very well be presumed that it was written much after the organization of resistance against the taking away of the primitive freedom in hacking. Obviously there must be a process, made of many small steps, through which GNU reached this definition of freedom. But, this document does not mention these steps.

This document does not mention any history of reaching at this definition of freedom in the world of software creation. After reading this document, it may seem that, as if this four-point formulation of freedom was instantaneous, which, I believe was not the real case. Such a terse and elegant formulation to grow, I presume, it takes quite some real time and experience. History does hardly happen in a dramatic way. There remain infinite number of small events. A context makes some of them fall in a line, and hence, a tentative time-line gets created. Some more events gather, and so on it goes, till the whole shape of real history has incarnated a new idea, and now this idea leads us in searching back the history.

In the real history the context was always already there, in the voice of dissent, in the waves of counter-culture, through the whole decades of sixties and seventies, specifically in those universities where the young minds were giving shape to the still unnamed FLOSS tradition, without knowing what exactly they were doing. The context loaded them with some tendencies of time, and they were marking out the footprints of those tendencies in the hackers' space. Some of these accumulating supplements of resistance we have already mentioned, and there are still some more events to come, that have to fall in line to create the bedrock where the confluence of GPL and Linux kernel will happen: let us retrace that.

In 1984 Stallman quit MIT AI Labs so that AI Labs cannot interfere in the works of the GNU project of free software, and cannot claim any right of distribution on any work that Stallman does there. This 'free' of 'free software' from now on signifies the four-point freedom of hacking mentioned above in the quote from Stallman. Another major step here was the GCC project, the GNU-Compiler-Collection. We have already discussed the importance of GCC and all the development tools created by GNU. The quote from Stallman above concerning the need of an OS also mentions this importance. Stallman started working on GCC in 1985. Here happened another glitch about license concerning the primary development of the compiler with the author of VUCK, Free-University-Compiler-Kit. Finally the first beta version of GNU-Compiler-Collection was announced by Stallman in an email on Sunday, 22nd March 1987, 10:56 AM. One necessary note here, before reading the quote from this email: 'Alpha' and 'beta' are just two stages of software development before the software is released. Usually 'alpha' build is delivered to internal software testers other than the software engineers in the process of preparation, and 'beta' is the stage when the software has passed the alpha stage, and is now ready to be tested and used by a limited number of users. As the email announced, the beta version GCC was made available on the Net by FTP, File-Transfer-Protocol, such that other hackers can access and use and hack it. And, in the email, for the people who cannot access the FTP, Stallman offered to sell it, Stallman 1987.

If you can't ftp, you can order a compiler beta-test tape from the Free Software Foundation for \$150 (plus 5% sales tax in Massachusetts, or plus \$15 overseas if you want air mail).

This quote demonstrates an important point, marking out the true significance of the term 'free' in context of software. The term 'freedom' in Stallman's texts does not imperatively mean 'free as in free beer'. Obviously one can make it that way, but the imperative meaning of 'free' is 'free as in freedom of speech', as formulated by Stallman in the four-point freedom of hacking. This price/sale aspect will become crucially important for GPL to integrally weave the 'freedom' aspect into the fabric of ownership of commodities and wealth. We are coming to that in due course.

The same price and sale thing happened with Emacs too. Actually, now, with a dispassionate historian's way of seeing, it seems quite like a preconceived project. As if, the reality was putting forward every loose end such a way that makes the future of this whole endeavor prepared for everything that can take place in this world. As if it was preparing the endeavor for every glitch of every order. And GPL was finally the name of

the completeness that emerged through this endeavor. The endeavor called GNU came through all these knots and that is why it could come up with an apparatus like GPL. We have already preferred biological analogies to depict the immense vitality inherent in GPL. Reemploying another biological analogy, we can say, these were all acting as antigens, against which the antibodies were getting stored. All these will finally actualize in the organism of GPL. These were all small supplements, with very little importance in itself in a stand-alone way. But, together, in an accumulated way they were just waiting for GPL to arrive.

Initial development of Emacs started during the seventies in the AI Labs. The very primary EMACS, Editor-MACroS, for an editor called TECO, was written by Richard Stallman together with some other people, while he was working in AI Labs. Macro-s are automated job-doers in the form of rules or patterns that specify how certain kinds of input will be produced into some specified form of output. Then this Emacs went through a lot of evolution and phases of development, and a lot of different versions of Emacs or Emacs-clones came up. This led to a problem of too much customization and forking. There was some glitch with the proprietary rights of Emacs too, we are coming to that later. Around 1984 Stallman started writing GNU version of Emacs that came to an usable shape in 1985. Let us hear about the price dimension from Stallman 1999 once again, the way in which this software was made available to people.

I could have said, "Find a friend who is on the net and who will make a copy for you." Or I could have done what I did with the original PDP-10 Emacs: tell them, "Mail me a tape and a SASE, and I will mail it back with Emacs on it." But I had no job, and I was looking for ways to make money from free software. So I announced that I would mail a tape to whoever wanted one, for a fee of \$150. In this way, I started a free software distribution business, the precursor of the companies that today distribute entire Linux-based GNU systems.

The run of the logic we have already witnessed, but this is one explicit mention of the economic aspect of FLOSS software, FLOSS as a source of wealth and property. And the way this Emacs event handled this property aspect is going to be the embryo of property aspect of GPL.

One more aspect important for us concerning GPL is the protection of FLOSS as FLOSS. Stallman discusses this aspect concerning the fate of X-Windows in Stallman 1999. We mentioned in chapter three, X-Windows is the basic core of GUI or Graphical-User-Interface implementation on which were built all the later FLOSS GUI things.

Developed at MIT, and released as free software with a permissive license, it was soon adopted by various computer companies. They added X to their proprietary Unix systems, in binary form only, and covered by the same nondisclosure agreement. These copies of X were no more free software than Unix was.

This is a problem. Someone creates a piece of FLOSS software, then deliberately forgoes all the rights on it, making it absolutely 'free' when making it available. Now anyone can

take that up and put an enclosure around it, and make it closed. Stallman here mentioned the X-Windows incident. Another very important event of this nature happened in case of Mac OS Ten and FreeBSD. FreeBSD is, we said before, one BSD variant of the Unix clones. Mac OS Ten, or MacOS X, is meant for the Mac machines. Mac OS X is a proprietary software. The Mac machines are also proprietary hardware: 'proprietary hardware' in the sense that only one concern Apple Inc. manufactures them, unlike the PC clones made by numerous manufacturers after IBM made the blueprint public, as we mentioned before. Let us declare it before reporting this event, that, this report is more of a conjecture than some verifiable argument. Because, it is in some cases extremely difficult, if not altogether impossible, to prove undeclared code-appropriation due to both logical and legal reasons. But a lot of knowledgeable people conjecture in favor of this argument that we present here. As some educated opinions go, Mac OS Ten incorporates some source code from FreeBSD and NetBSD. Both these BSD variants are distributed under a FLOSS license called BSD. But while FreeBSD and NetBSD were FLOSS, Mac OS X was a property of one of the monopoly giants in computer industry. This is another example how the common pastures can get enclosed by private capital in the knowledge-land of software industry.

The term 'flame-war', popular in the mailing list domains of FLOSS, means a long and continued discussion through a war of emails in the mailing lists, and 'flaming' refers to exchanges of an unpleasant kind between Netizens on these mailing lists. One of the biggest flame-wars in FLOSS history was the GPL-BSD issue. Mails and web-pages in the Net archives focusing on this issue would be available in millions, if not more. Here, we have got absolutely no point in passing any value judgment about which one is correct and which one is not. We just want to mention what was what and how that changed the course of things to happen one after another, such that when they are written down in a text, a scheme comes out of it and gets known as history. For us these all constitute the endless series of supplements that would later find their father in the text of GPL.

The history of *counter-culture* resisting against the rules of father, and then the empire striking them back, brings out one issue very clearly: the issue of infinite openness. The issue that, maybe, some check should be there about an infinite openness. Freedom and openness without a check are indeterminate freedom and openness, and any concrete closure can close them down once again. Reiterating the debate in the GPL-BSD flame-war is pointless here, the debate about which brand of 'freedom and openness' is in the truest sense the more free and more open, GPL or BSD. The point is that the BSD license had no in-built check to bar Apple from appropriating BSD code, if Apple did it at all. Such a check could block a concern from appropriating FLOSS source code into a proprietary software generating newer properties. And, more importantly, it could block the concern from expropriating FLOSS tradition from this new software that grew on FLOSS flesh and blood. BSD has a very clear position here, the position that it wants to achieve through the BSD license. X-Windows people, too, had a very logical explanation of their own, depending on what they want to do with their software. Not that we want to proclaim here how wrong they were, or, if they were at all wrong or erroneous. Many debates floating on the Net want to scan this degree of rightness or wrongness, trying to measure the difference between the degree of freedom and openness between GPL and BSD.

The question here, in this book, is not at all about this measurement. All these debates are pointless for this book in a way that is very well understood by elementary arithmetic. These debates are wanting to compare two qualitatively different things. They are trying to calculate the proportion between an elephant and a horse, while actually the whole proportion paradigm works only between things that are alike: like two horses and three horses – the referent has to be identical. BSD license has a kind of infinitely free indeterminate openness about the ownership of the property generated from the wealth of knowledge residing in the software under BSD license. But, GPL is very determinate about which kind of ownership of property it would allow to generate from the knowledge capital licensed under GPL. It is a very determinate and finite kind of freedom and openness. And it is just meaningless for our purpose here to discuss which kind of ‘free-ness and openness’ is better for human society at large. Here we are not talking about the quality of being good, we are talking about the newer kinds of property categories generated through GPL, obviously which was not at all the case with BSD license. Unlike GPL, BSD license does not interrogate the traditional social fabric of property relations and try to dislodge them from their social ethical equilibrium, what exactly GPL does. GPL attacks these social relations from within with the newly formed categories generated by GPL. The details of this process of dislodging we are going to discuss in the next chapter, together with all their philosophical implications.

To return to the point of history, we have not yet reached GPL in terms of our retracing the events. During the period 1984 through 1988 different GNU pieces of software carried different kinds of license. All of them were micro-trials into reaching the final goal that was GPL, still in the making, and not fully known yet. And, by every step, the goal was getting more distinct and clear. They were all trial pieces of text through which the supplements accumulated into GPL. We have already said that in 1983, a plan for the mass-collaboration project of FLOSS, called GNU, was announced. And, the work of development on the planned GNU OS started in 1984. As we have already mentioned, more than once, this GNU OS project was the envelope project that Stallman was planning for a long time.

Now, once the envelope project was reached at, it crucially needed one very determinate address, to pursue the action of fight against the closure imposed on FLOSS by capital’s rules. This address came in the form of FSF, Free-Software-Foundation, founded in late 1985. Gradually, all the pieces of jigsaw were falling in their places and creating a pattern, which none of the components could foresee solely on its own. And neither was there an oracle to foresee all of it from before. For an oracle would need to see this pattern from outside the reality, with an *extra-real*, that is, unreal knowledge. The oracle, at that point in time, would have to know beforehand, where all these supplements would lead to, that is the text of GPL. The oracle will have to know the very context called FLOSS that the text of GPL would create. The context that is working as a backdrop of writing and reading this book. The oracle will have to know what we know in 2010.

In fact, the pattern was emerging with such a force that all the requirements for creating an OS, as proclaimed by Stallman in that envelope concept – all the tools, all the software gadgets, were already there, by the time Torvalds and the other hackers together created the kernel. The only one thing that these FSF and GNU endeavors could not come up with was

a kernel. Now, Torvald's Linux kernel was the last remaining piece of the pattern. But this whole pattern that started to emerge quite distinctly from 1985 onwards still had one big clue missing. That was GPL, General-Public-License from GNU. This is the point that will make the whole envelope so crucially important. It is the envelope that would fill all FLOSS hacking with so much vitality, that it can return the gaze of father, that it can stare in the eyes of the rules of capital. And that too in a very different way, different from all the prior ways in which power of capital was fought against. GPL was the point that enabled the envelope become the haven of all counter-culture things, enabled it to become the surrogate father of all the bastard supplements, that too without going into any duel with capital, avoiding that very duel that all the Marxist and communist paradigms opt for. That is the very novelty of the logical twist involved in GPL.

As we said, from 1984 to 1988 GNU pieces of software were attached with different kinds of license. The web resource Tai 2001 reports in details about the events leading to the birth of GPL, quoting from many Stallman documents. One important bit here was the glitch about license and right over Emacs, that we already mentioned in a passing way. After Stallman wrote the first Emacs in 1976, James Gosling, the so-called father of Java, wrote Emacs in C for the first time in 1982. It was called Gosling Emacs and it ran on Unix. The source code of Gosling Emacs was primarily under free distribution and that was used in the first version of GNU Emacs in March 1985. Then Gosling Emacs was sold in 1983 to a corporation called UniPress. This version became known as UniPress Emacs and came under a proprietary license. UniPress asked Stallman to stop distributing GNU Emacs source code that contained codes by Gosling. Now GNU had to go through the process of expunging all bits of code written by Gosling, and four months after the primary release of the tainted version, Emacs version 16.56 was released in July 1985, where Stallman replaced every bit of Gosling code with codes of his own.

This was the last and the most important bit in the pattern. The last supplement that would augment the process of reaching GPL. This generated a search for getting some way to block the recapturing of that, what has been freed and opened up, against the onslaught of the market rules of capital in general, and proprietary software in particular. The last and final lap, in this marathon preparation of all the elements that will go into the making of GPL, is at last complete. The Emacs incident, exactly like the X windows one, and arguably like the Mac OS Ten and BSD episode, and some other incidents of the same nature, pointed out the fact that for FLOSS to become and remain FLOSS, it needed something more: it needed some check that could block the recapturing of the freed land, using our old metaphor of primitive accumulation and land ownership. This check came in the form of 'copyleft', the single sleight of hand that would go on destructing and reconfiguring the age old social ethical and economic categories, in an extremely unprecedented and immensely philosophical way. This would actually ruin the age old concepts of both friendship and war, and, for the first time in the history of civilization, would breed any real concept of struggle without a war, something much beyond the traditional communist and Marxist war paradigms of struggle.

Let us cite here another quote just to remind us one point that we have hinted more than once. This quote shows beyond doubt that whatever magic happened there in GPL was way beyond the philosophical contemplation of Stallman himself. GPL was paving the

ground of a new age philosophy beyond the concept of war. And Stallman was still using the traditional paradigm of war, hero, coward and all, things that are so usual in any diatribe, be it petty political, or FLOSS or of the Marxist genre. He was talking about Gosling, in context of the Gosling Emacs incident. This quote is from Stallman 1986. This 1986 speech by Stallman was given at the Royal Institute of Technology, Sweden. Tai 2001 quotes from it too. Though there are some minor editing differences.

Gosling originally had set up his Emacs and distributed it free and gotten many people to help develop it, under the expectation based on Gosling's own words in his own manual that he was going to follow the same spirit that I started with the original Emacs. Then he stabbed everyone in the back by putting copyrights on it, making people promise not to redistribute it and then selling it to a software-house. My later dealings with him personally showed that he was every bit as cowardly and despicable as you would expect from that history.

It is not the case that we want to pass any opinion about the justification of this anger. We only want to point out the fact that this is petty pedestrian in comparison to what GPL is going to achieve, both logically and in terms of real existence. It seems pretty unfitting for a person to go down in history as the author of GPL to talk in terms of such war cry, and deploying phrases that better describe a war situation than something absolutely beyond everything that we knew about war and friendship through the ages. One more thing can be mentioned here that this 'diatribe' mode is very much common there in Stallman's texts and speeches, even till the recent times, like in the case of his case against Eric Raymond's concept of Open Source Software in comparison to his own concept of Free Software.

It was actually quite conscious and deliberate that, all through this book, time and again 'free and open source software' was repeatedly used while talking about FLOSS. Without declaring it beforehand, we wanted to chisel it out: we are talking about something that is far bigger than these vitriol things. We cannot go into the details of Free Software versus Open Source Software debate here, that would be beyond the scope of this book. But one thing is very clear to us, the logical things that we are going to discuss in the coming chapters, were indeed beyond the comprehension of the author of GPL. Maybe it is the way such revolutionary texts go, much beyond the comprehension of the very authors that write those, the authors themselves being products of the very time in which they get written. And the time GPL was getting written, was the time in which the supplements were accumulating. The full meaning of GPL can only come out when read from the context of FLOSS. And this context, at that time, was yet to happen. As we said, in chapter one, GPL is going to be the example which brings deconstruction and differend together, creating a reading strategy that goes from supplements to text to context. It would involve the very deep-acting twist of GPL on the very social ethical and economic categories, but we are yet to elaborate that twist in terms of Hegel in the coming chapter. One thing is pretty sure here: the reading of GPL that we are going to present in this book was something that Stallman himself did not read, it was wildly beyond his way of thought. The implications of GPL in terms of Hegel's logic and the full implication of GPL in terms of all philosophies of struggle will be the subject matter of the coming chapters.

Now to proceed forwards any more it will be a help to get a little familiar with GPL and BSD licenses. Let us cite some references here. Both GPL and BSD licenses are available in hard print with different books, essays, and software distro-s. But, the best way to get them is the Net, because all the recent changes and discussions about them are available then and there. The current Version of GNU GPL is 3, the release date was 29th June, 2007. The release date of GPL Version 1 was February 1989. And, the release date of GPL Version 2 was June 1991. All the texts are available online, in the GNU website, <http://www.gnu.org>, and it is a very interesting study to go through the changes in the text of GPL from one version to the next. One excellent website is there concerning any necessity of understanding any aspect of GPL in particular, or any legal aspect of any license in general, that is Groklaw, <http://www.groklaw.net>. And one particularly interesting page is there comparing different elements of GPL Version 2 and 3, in Jones 2006. BSD Licenses are actually a family of software licenses that intend to seek what we are calling FLOSS rights in this book. The mother license of this entire BSD license family was the original one that was used with BSD or Berkeley Software Distribution, produced and distributed from University of California, Berkeley. The BSD distro evolved with time, and the accompanying license evolved too. One important variant of the family of BSD Licenses is FreeBSD License. While I am writing this book, version 4.4 of FreeBSD license is available at the FreeBSD website, <http://www.freebsd.org>.

Though we must keep in mind, just a reading of the text of GPL is almost bound to miss the subtle points. And because it was a legal text, a text on which many a legal battle were fought and won, it has its legal intricacies. But that is not the whole point here. In this book, we are not going to read GPL from a legal viewpoint. We intend to read GPL in a particular light of Hegel's theory to get the reading that we want to present in this book. GPL or BSD License, or any license for that matter, is not a piece of text of literary value. It is a 'real' text in the sense that, this kind of text undergoes direct application in real life and works as a bridge between the theoretical space of computer science and the world of legal rights of private property. As a text it is something very different from the literary texts. The literary text is always at one step remove from the real world, and works through the consciousness that it generates in the thought of the reader. That is true for GPL, and something more is true too. GPL is a text that contains manual of some legal actions, real legal actions executed in the real socioeconomic space. So in the last section of this chapter, we go into some very necessary legal details about GPL in particular and licensing in general. The version 1 of GPL was the oldest one, then came later versions, in tandem with a changing real world of property, capital and knowledge. And there were things like LGPL, Library-GPL too. But those details, just like the Free Software and Open Source Software debate, are beyond the scope of this book. For the points we want to make here, GPL Version 1 will be sufficient.

4. GPL and Linux Kernel

GPL Version 1 was released in February 1989. It was written by Stallman to be used as a blanket license for the different licenses given with GNU development tools like Emacs, GNU Debugger – a developing tool, and most importantly, GCC. GPL version 2 was released in June 1991. There were some changes between Version 1 and 2, none of these

changes are going to affect the reading of the ‘copyleft’ concept in GPL in our book. One very important change in GPL Version 2 actually strengthened the non-recapture status. Some code that is already made free and open through GPL, if gets used in another piece of software, this piece of software can never be closed down. And if it is indeed closed down, the piece of software, under such closed license, cannot be distributed at all. This is what GPL Version 2 proclaimed. And that actually is something that goes in favor of the logic that we are developing in this chapter and the next one. So it strengthens all the things that we are going to say in the coming pages of this book. This GPL Version 2 is the version of GPL under which Linux kernel was released.

As we said, after Linus Torvalds reported about the kernel in a post to comp.os.minix, in September 1991, Linux kernel version 0.01 was released. Version 0.02 came in October that year. The website <http://kernel.org> contains all the materials and resources that we are using in this section. All the references cited here are available on this website, in the section on old-versions. The release note attached with the kernel 0.01 contained the license declaration that we are quoting here, exactly the original, including the typo, Torvalds 1991. It came in the Section 2, titled ‘Copyrights etc’ of this document.

This kernel is (C) 1991 Linus Torvalds, but all or part of it may be redistributed provided you do the following:

- Full source must be available (and free), if not with the distribution then at least on asking for it.
- Copyright notices must be intact. (In fact, if you distribute only parts of it you may have to add copyrights, as there aren't (C)'s in all files.) Small partial excerpts may be copied without bothering with copyrights.
- You may not distribute this for a fee, not even "handling" costs.

We can see some of the points we already discussed about the FLOSS tradition repeated here in this informal license. Only the last part, ‘not distribute this for a fee’, is different from our experience with GCC and Emacs. We are coming to that. One other thing we should quote here from this release note, Torvalds 1991, that is the lamentation for the lack of an OS till then, and calling for use of different GNU tools that were already in circulation among the hackers.

Sadly, a kernel by itself gets you nowhere. To get a working system you need a shell, compilers, a library etc. These are separate parts and may be under a stricter (or even looser) copyright. Most of the tools used with linux are GNU software and are under the GNU copyleft. These tools aren't in the distribution - ask me (or GNU) for more info.

The Linux kernel was getting distributed under the license written by Torvalds till version 0.11. Version 0.12 came in February, for the first time the kernel was under GPL. We have already said, by that time GPL was version 2. The release note with the Linux kernel version 0.12 had a small section under the heading ‘COPYRIGHT’. Let us quote this section, Torvalds 1992.

The Linux copyright will change: I've had a couple of requests to make it compatible with the GNU copyleft, removing the "you may not distribute it for money" condition. I agree. I propose that the copyright be changed so that it confirms to GNU - pending approval of the persons who have helped write code. I assume this is going to be no problem for anybody: If you have grievances ("I wrote that code assuming the copyright would stay the same") mail me. Otherwise The GNU copyleft takes effect as of the first of February. If you do not know the gist of the GNU copyright - read it.

As we can see, the 'not distribute this for a fee' part is taken away by Torvalds, and he resorts to GPL that is already there. So, all the factors that were there to fall in a line and generate the pattern that we now call FLOSS at last came together and created the pattern and FLOSS was born. So, now the endless series of accumulated father-less supplements, together with their father text, GPL, complete the chain. It is time now for the process of birth of the context called FLOSS. Now to get the full significance of GPL, let us go into a bit of details about license, copyright, and intellectual property rights in the next section, which in the true sense should be called an appendix.

5. Technicalities of Licenses

Shun-Ling Chen wrote an excellent thirty pages booklet on FLOSS license that discusses the technicalities in a very simple way, Chen 2006. What I did in this section is just augmenting the discussion there with some materials from "Producing Open Source Software" by Karl Fogel, Fogel 2005. Let us keep one thing in mind that, while both these texts are excellent in generating a wisdom of the readers about the broader perspective of intellectual property rights, we will be deliberately choosy and myopic in our vision. We have a very humble goal, to reach into the workings of GPL in such a way that we can cross-compare the categories of GPL with the Hegelian categories of Right and Property. Here we discuss a bit about the definitions of the different terms related to intellectual property rights, because to understand the full implications of licenses on some intangible product of human creativity like software, we need to understand intellectual property rights. Intellectual property rights can be of different types, like *Trade-secret*, *Trademark*, *Patent*, and *Copyright*. While we discussed a bit about source code, object code and their license in chapter three, where we mentioned about the protection of intellectual property rights through Berne Convention, WHO and TRIP, we need some more details about software copyright and license to understand the intervention of GPL into socio-economic categories.

Trade-Secret

Trade secret means some form of confidentiality used and guarded by a company as an advantage against competitors. It is guarded by the company as classified information. This secret may come in different forms, like a formula, or a in particular form of practice or process, or in the shape of some design or pattern or instrument, or in some particular compilation of some information like a 'know-how'. The crucial condition of a secret to become a trade-secret is that of economic benefit – it must bestow some kind of economic

advantage to the holder of the secret over all those competitors that do not share the secret. So, three things come together in creating a trade-secret. One, a trade-secret resides in some information that is not known to general public. Two, a trade-secret must generate some economic benefit. Three, some efforts are spent in keeping a trade-secret.

Trade Mark

Trade Mark is a sign of identity. This sign can take many forms like names, phrases, symbols, designs, pictures, or styles used by a business to identify itself and its products or services. Trade Marks are meant for easy identification by the consumers. For spelling out a Trade Mark usually symbols like TM and [®] are used. Certain exclusive rights are attached with a registered Trade Mark, and legal actions can be taken for any infringement on that.

Patent

A Patent means some exclusive rights on an invention granted by the state. This invention can be a new process or technology, or a machine, or some produced good, or anything that is new and useful. The exclusive rights are granted by the state on the inventor for some period of time, in exchange of the disclosure of the invention. While a Trade Secret enables a business to guard some particular information, for a disclosure of certain information the business is granted a Patent for say, twenty years.

Copyright

A Copyright is meant for a creative work, literary, musical, painting, software or something, usually © being the sign of declaring a Copyright. Unlike a Patent, a Copyright is automatic, it applies to a work upon its creation. A Copyright gives the creator of an original work exclusive rights for a certain time period over the circulation, publication, distribution of the work, including its adaptation, if any. One important thing about a Copyright is that, while some work is getting copyrighted, the ideas operating within the work, or generated through the work are not copyrighted at all. A License of Copyright often accompanies a piece of creative work. This license declares under which terms and conditions the work can be used.

Software and Copyright Regulation

A piece of software is subject to copyright laws. Even arguments in favor of enabling a patent for a piece of software has come up, even in some cases they were granted too. Though the FLOSS tradition does always consistently resist it. There are lots of details here, and thousands of web-pages and scores of books are written every year around this. As we said, there is even an website: <http://www.groklaw.net>. But, for us, as we already said, the only thing that matters is to understand the features that made GPL what GPL is.

Free Software

A piece of software that can be freely shared and modified, including its final and source code form. So, in another way, this is the same definition of free software put forward by Stallman, that freedom with software means four freedoms, 1, freedom to use the software, 2, freedom to study and change the software, 3, freedom to copy and distribute the software, and 4, freedom to improve the software and distribute the improvements. So, as

we have already mentioned, the freedom of free software has got nothing to do with ‘free’ as ‘zero-cost’. So, as we know through this text, this is the kind of software on which the FLOSS tradition is built.

Open Source Software

In the true sense, another name of ‘free software’, but the difference in name stands for a difference in philosophical and ideological interpretation of the tradition. The PDP machines were never called computers by the manufacturers, as we have discussed in Chapter Three, in order to vest the machines with a new kind of identity. Exactly the same way, the term ‘Open Source’ wants to make the term more presentable to business enterprises, ‘free software’ already having an aura of its own, in many cases which may intimidate business houses. And Open Source wants to interpret the FLOSS tradition more as a ‘development methodology’ than a political movement, the way has gone the FSF way of thought, as the OSI, Open-Source-Initiative, believes. And there is one bad connotation of the word ‘free’ too, because as many market experiences go, things that are given ‘free’ (in the zero-cost sense) are scarcely up to the mark, and this meaning of the word ‘free’ gets stuck with it, OSI believes. But, technically, there is no difference between ‘free software’ and ‘open source software’, that is, a free software license is always a open source software license and vice versa. A lot of things we can read on the Net about OSI and their own website (<http://opensource.org>) defines and discusses all aspects of OSI interpretation, but for our purpose it will suffice to know that free software is open source software too, and equally relevant for our later discussions on FLOSS.

Proprietary and/or Closed-Source Software

Proprietary and/or Closed-Source Software is exactly the opposite of ‘free’ or ‘open source’ software. When software is distributed under royalty-based licensing terms, where users pay for every copy of it, it is proprietary software. The terms of using it may differ, but, it is always sufficiently restrictive to prevent the very FLOSS dynamics that we are discussing all through this book. Even software distributed free of cost, that is, zero-cost, can be proprietary, if the license attached with it does not allow free distribution and modification. Usually all proprietary software is closed-source. The term ‘closed-source’ means the source code which was compiled to get the software cannot be seen and thus read, studied or modified. For most proprietary software the source code cannot be seen, and therefore they are obviously closed-source too. Even if in a particular case the source code is allowed to be seen by others, the issue is: what one can do with it? Can it be put into modification and redistribution of modification, which is obviously answered in the negative for this kind of software. The proprietary closed-source software can very well be commercial or not, but that is no point here, because as we said before, the FLOSS licensing system does not bar commerce, that is, selling of software. What FLOSS does is to keep the *freedom* intact.

Public Domain

A software is in Public Domain when it has no copyright holder. That means when there is no one to restrict copying of the work. An author of a piece of software can deliberately send it to Public Domain, and this sending does not change the fact that there was, after all

an author who wrote it. The author is always there, but now without a copyright. The problem with Public Domain is that when some software is sent to Public Domain, materials of this software can be modified and the resultant software can become a copyrighted software. Things like this we have already discussed in this chapter. Releasing something into Public Domain, is technically speaking, absolutely and indeterminately free, and hence its materials can be recaptured into proprietary and closed-source software. This is one crucial point of contention between GPL supporters and BSD supporters, as we have already said.

Copyleft

Copyleft is a kind of a license that is never the negative or negation of copyright. It is a popular misconception that copyleft is the case when the right is left behind. Copyleft licenses use the same copyright law that is used by the copyright licenses but to achieve an altogether different result. It ensures that the FLOSS freedom must ‘travel with the work’. That means, Copyleft licenses deploy copyright laws to enforce that all the copies, or derivatives, or copies of derivatives of the original work, fall under the same copyleft license. One is entirely free to copy, modify, distribute or sell a copyleft software, or even sell or distribute the modified copies too, but every copy of the original or modification or copy of modification must come under the same copyleft license. GNU GPL is Copyleft. So, one freedom is never there for any user of copyleft works: he is never free to take away the freedom in-built into it.

Features of GPL

The only focus of GPL was on freedom of hacking, GPL was deliberately crafted to make it impossible to create some software under proprietary licenses by adapting codes from a software that is under GPL. And so, the legal formulation of GPL works such a way that a software under GPL has two basic requirements.

One, any software that was built using any code that was under GPL must itself carry GPL license when getting distributed. Because it is derivative work of some GPL-ed code.

Two, in no way can any additional restriction be imposed on any GPL-ed code or its derivative work, which is obviously under GPL by condition one.

So, by these two conditions, what GPL ensures is continuity of freedom, or, to be specific, continuity of a very special kind of freedom that can never take away the freedom, excluding any possibility of backlash or closure on anything once free, or even its offspring. Obviously the different possibilities of these factors are unimaginably rich, any probe into GPL related literature will give a taste of that. But, we want to get back to our only concern, that is, how GPL changed the very categories of the world that we live in.