What the Are Linux and Free Software?

A few weeks ago, during one of my lectures, I experienced a minor technological glitch as I was trying to make the classroom's projector recognize my laptop and show the slides properly. As I quickly attempted to fix the problem, I apologized to my students for the delay and I told them that those kinds of inconveniences sometimes happen when you use Linux as your operating system. Their reaction: blank stares. Not even one sympathetic smile. “Linux, you know? Free software,” I said. Nothing. “Open source? Anybody?” Dead silence. Granted, my students look at me like I'm speaking in Klingon when I try to explain Marx's theories to them (and, the pinnacle of nerdiness, he even looks excited!), but I'm talking catatonic, hard-to-tell-they're-alive faces here. I then realized that, after 10 years using Linux, I assume that people know what free software is (and, shame on them, they stick to Windows or Apple because they're evil), when in reality many of them are not even aware of its existence. A friend reminded me of this when I made some high minded remark about Steve Job's legacy after his passing a couple days ago, and he suggested that I write a short note laying out what free software and Linux are. So here I am (although this has ended up not being short at all; oh well).

The free software movement was started in the mid-1980s by a software developer, Richard Stallman, a controversial figure with a rare talent for making enemies who is now as much revered as vilified. Stallman had a problem getting his printer to work properly due to some design flaw in its drivers, the software that our computers run to communicate with printers and other external devices. In order to fix the problem, Stallman contacted the printer's manufacturer and asked it for the source code of the drivers so that he, an experienced programmer, could fix it. But the manufacturer refused to send it to him. You see, most of the computer code at the core of the software that we use is written in 'compiled languages' (like the überpopular C++), which means that the instructions that make up the program are 'compiled', i.e. translated, into a language that the computer's processor can understand (but which is largely unintelligible to humans). The result is an executable file (such as the files with .exe extensions in Windows) that we run to install and run our programs. What this means is that when a user runs a specific application on her computer she does not get access to the instructions, or source code, that make up the program. The source code is thus 'closed', so you can see what the program 'does' but not how it does it. Now why would the printer's manufacturer refuse to send the source of code of its drivers to Stallman? The argument is a commercial one: as long as the source code is kept secret, nobody can copy it and produce copycat programs that will compete with yours, thus giving you the possibility of cashing in on your invention. The rationale is not in essence any different from that supporting the existence of copy rights and patents: if you invest in generating an innovation you should be guaranteed control of your invention and given the possibility of benefiting from it commercially without facing the competition of free riders who copy your work without incurring the risks and costs of a resource-consuming and uncertain creative process.

Yet there are some problems specific to closed source, or, as it is often called, proprietary software. The rights of the producer are definitely important, but what about the rights of the user? If you buy a computer program, should you not be able to, at the very least, know what it is actually doing? Years ago, some people realized that when they were running Windows on their computers, there was heavy online traffic that they were not generating (i.e. they were not using their web browsers or email clients at the time). What was going on? The fears were that Windows was gathering information about the user and sending it to Microsoft's headquarters without the her knowledge. This is not to say that Microsoft's intentions were evil and that they were trying to hurt the user. Most likely, they just wanted to know their users better in order to more accurately tailor their or other firms' (to which they could sell the data) products. But the point is that, not being able to access the source code, it was impossible to know what the software was doing. Could there be firms selling (or even freely giving) us software

which is actually stealing our data and even using it for detrimental or illegal purposes? Another problem is the one Stallman confronted. What if we have spent money on software that does not do what we want or need? Maybe a slight modification of the code would be sufficient and, after all, we have purchased the software. So why shouldn't we be able to modify it as we want, in the same way that we can alter any of the products that we acquire and thus belong to us? Finally, there is a line of argument that affirms that proprietary software stifles innovation. As I mentioned above, a key reason for the existence of copy rights and patents is that they are necessary to foster innovation. Why would anyone spend time and money on creating something if they are not able to financially benefit from their investment afterwards? But there is a strong argument to be made for the case that innovation happens when we can build on previous knowledge without having to start from scratch. This is a particularly important issue in software production, where the re-use code is essential in order to avoid the duplication of efforts and the wastage of resources. And how can we innovate and bring the frontier of software development forward when we cannot read the code that others are writing?

There is a variety of reasons, then, to argue that the source code of computer programs should not be kept secret, and that users who legally purchase software should be able to modify it and use it in whatever way they want. After all, they bought the programs, didn't they? Hmm... no. Notice that every time you install a piece of software you have to go through a step in which you are asked to read a long text and 'agree' to it before you can move on. That long text is a EULA, an End-User License Agreement that stipulates that what you have acquired is not a particular computer program, but the right to use it under the terms of that specific license. And if you go and read that license you will realize that you are basically agreeing to give up any rights on your purchase beyond its mere use. EULAs are so detailed that you are almost even obligated to go down on your knees and thank god that Windows, Apple, or [write the name of your corporation of choice here] allows you to use their work. You cannot modify, share, copy or distribute the software in any way, and if the program kills all your data, sends your information to a Russian porn website and your money to a Nigerian prince, or just destroys your computer, you are on your own and they are not liable in any way.

In realizing all of these issues through the printer driver's incident, and in opposition to the notion of proprietary software that he identified as the source of the problem, Richard Stallman came up with the concept of software freedom. As he is fond of saying, the freedom in free software does not refer to monetary cost as in 'free beer' (and, in fact, Stallman has no problem with people making money from writing and selling software), but to freedom as in 'free speech'. People should have the right to inspect, use, copy, distribute, and improve the software that they acquire in any way they want, and limiting their ability to do so implies a limitation of their freedom. Thus he called software developers to produce software that gave people these freedoms. In order to ensure that this would be the case, Stallman and the foundation he created (the Free Software Foundation) came up with a new type of software license that guarantees not only that the software produced will remain free, but also that any software based on free software will remain free software. This means that, contrary to what happens with unlicensed works, nobody can take a free software-licensed program, modify it, and sell it as proprietary software: if you have built on free software (and therefore taken advantage of its free character), you have to make your work available as free software as well.

The emergence of free software, coupled with the modular and digital character of computer programs and the growth of the internet, spawned the birth of a community and a movement that would have been impossible only a few years earlier. Developers from all over the world began to write code and share it with others who would in turn improve, expand and share it as well. The result was that complex programs that, before, were only possible within large corporations, were now being developed cooperatively by programmers who had never met personally and who had no other ties than

their desire to collaborate. The Free Software Foundation thus spearheaded a project to collaboratively write an operating system that would be free software. They called this project GNU, a recursive acronym that stands for Gnu's Not Unix (in reference to the type of operating system that they were trying to model their work on, Unix). By the early 1990s, the GNU Project had successfully created many of the components of a modern operating system, but it still did not have the piece that lies at the core of and controls the system, usually known as the 'kernel'. It was then that a young Finnish computer science student called Linus Torvalds announced on the internet that he intended to write a Unix-like kernel which, when it was finished, came to be known as Linux. Soon after he released the kernel, other developers put all the pieces of the puzzle together and created the first complete free software-inspired operating system: GNU/Linux, which most people refer to simply (and inaccurately, as Stallman likes to remind us) as Linux.

But let's remember: Linux is made of a number of different components, and developers are able to access and modify it in any way they want. Very soon, then, we started to see the emergence of different versions or flavors of Linux. Some people built minimalistic versions with just the most basic components, others began adding new tools, for instance, to facilitate the installation of new software in the system. The end result was that within a few years it was impossible to talk of Linux as 'one' operating system. Hundreds of versions of Linux emerged, and these came to be referred to as 'distributions'. Linux systems at that point were almost exclusively used by software programmers. All the software was run by writing instructions (in what is known as the command line), similarly to how the pre-Windows DOS operating system worked. In the late 1990s, however, different groups of programmers started to work on the creation of windows-like graphic environments (nowadays called desktop environments or simply desktops) to add to the bare-bones Linux systems. Following the logic of operation of free software communities, several of these desktops emerged, the most popular of them being Gnome and KDE. This added another layer of complexity to the Linux ecosystem. You do not run just Linux on a computer. You first need to choose a distribution, and then you can choose what graphical windows environment or desktop you want to run on top (nowadays some distributions come with one specific desktop, while others come in different flavors for different desktops).

How much coherence is there in all this diversity? Because all these systems share the same core, they basically work in the same way, they have similar file system structures, and the commands are usually interchangeable (even if there are differences here and there). You can run programs built for a given desktop on a different one, although that usually involves a less than seamless integration (some programs might look different or weird, there might be some graphical glitches or functions that do not work, etc.)

For a long time, making a Linux system work properly involved major hassles. Sometimes you would not get your your mouse or printer recognized, others you were unable to connect to the internet, or there would not be appropriate tools to change the configuration of your keyboard. All of these problems were solvable: after all, the code was open, and if you knew how to write programs or were skilled at looking for solutions on online boards you could get a pretty decently working system (and feel very proud of it). But, for about five or six years now, Linux has been mature enough that some distributions are installed and run as easily, and work with as much functionality and eye-candy, as Windows or Mac systems. This does not mean that there no problems. The process of creating free software is very different from that of producing proprietary software. Microsoft and Apple spend millions of dollars creating software in-house, and they test it and improve it until it is ready for release. Linux development takes place in the open. Distributions and projects within the community share their software openly, and if you want you can update your system daily to try and use the latest developments. This implies using highly unstable software that often creates more problems than it

solves. Many users are not very knowledgeable of these processes, and sometimes install software that is still in the process of development and thus become very frustrated, which leads them to criticize Linux, abandon it and vow to never return to it again. There are now many distributions whose goal is to put together rock-solid, stable systems that are supposed to work without a hitch. Yet distributions compete with each other, and are often put together by people who are eager to include the latest versions and updates of programs that might not be ready for prime time, resulting in annoyed users.

To the extent that using Linux involves problems nowadays, however, these are mostly the result of the proprietary nature of the drivers released by the manufacturers of hardware. When a company manufactures a graphics card, a screen, a keyboard, network card, a printer or a mouse, they also write the drivers that allow computers to communicate with those devices. However, because most end users run Windows systems, many manufacturers only write drivers for Windows, making it impossible for Linux users whose computers include those devices to make them work. Now, the Linux community has been very good at reverse-engineering and writing drivers for most of the devices out there. But this process often depends on the voluntary work of programmers, which means that it can take some time for Linux drivers to be released and that the drivers might not work properly right away, or have reduced functionality. These problems are usually solved with time, but it is not uncommon that someone buys a new laptop and finds out that the latest Linux distributions do not work because the drivers for a particular device are not yet available. Distributions are getting better and better at including drivers, and we increasingly find manufacturers that release Linux drivers for their devices, yet in this respect Linux is still not on par with Windows (and, of course, with Apple, whose operating system can only be run on Apple machines, thus ensuring the proper working of all devices in all cases).

As Linux has become more and more popular, and as more and more manufacturers produce drivers and other software for Linux, a different issue has emerged. It is perfectly possible to run proprietary software on Linux. As long as the code that you write is targeted to a Linux system, you can compile it and release it with a non-free license. Now, the impetus behind the creation of Linux was to have an operating system that is free software, and thus this confronts distributions and users with a choice: should you install in your system (or include in your distribution) a program or driver that is not free software? Or should you keep your system free, but then lose functionality (i.e. have a device that does not work or that does not work properly, or not use the program that would be your preferred way of playing music or watching videos)? Different distributions and users make different decisions about this, based on their values and their needs.

This brings us to the distinction between free software and open source software. In the mid-1990s, some developers became uncomfortable with what they perceived as the too ideological and extreme stance espoused by Richard Stallman and the Free Software Foundation. For these critics, the key advantage of making the source code of software available was the collaborative, distributed system of software production that it spurred. This was seen as a particularly efficient way of writing and improving software code, and thus valuable from a merely pragmatic perspective. From this viewpoint, then, opening the source code of software does not have to have anything to do with the notion of freedom. These critics thus started a parallel movement based not on the notion of free software but on the concept of open source. The Open Source Foundation was created, and as the Free Software Foundation had done before, it also came up with a series of open source software licenses that promote and defend the production of open source software. In practice, free software and open source licenses are mostly indistinguishable, and there are few real examples in which a particular program would be considered free software and not open source. This is the reason why people often use the two terms interchangeably, and why we often use the term FOSS (free and open source software) to

refer to the collection of software that meets the demands of both concepts. Yet this does not imply that there aren't stark advocates of one view or the other, and that conflicts are not common within the larger FOSS community.

A further cleavage comes from the role of for-profit corporations in free and open source software as compared to that of the community. Large corporations are playing an increasingly important role in free and open source software. The Android operating system for mobile devices, for instance, is open source, yet it is developed in-house by Google with little outside input. The most popular Linux distributions, such as Ubuntu, Fedora, or openSUSE, are backed by for-profit corporations that sell services to other actors based on their knowledge and expertise of the software. The development process of the software varies case by case, with some of these firms engaging in 'open governance', meaning that the community has full input and decision power, while others keep the products much closer to their chest. Of course, as long as the software is free or open source, anyone is free to take it and, through what is known as a 'fork', branch out from what the corporation does and continue the development independently. But, in practice, software development requires considerable resources and corporations are playing a larger and larger role in free software. There are, however, large open source projects that are managed by the community of developers and users in a non-hierarchical, participatory way. And there are other cases in which free software or open source projects are hosted by non-profit foundations, such the Firefox web browser, which is managed by the Mozilla Foundation.

Having said all this, and given the complexity of the free software and open source ecosystem, how should someone go about using a Linux distribution in his/her computer? How can we deal with the potential problem of having devices that will not work on Linux? First of all, most distributions (or at least the most popular and mainstream ones) make their software available in the form of Live CDs (or DVDs). A Live CD means that the entire operating system can be run from the CD (or even a flash drive). You put it in your CD tray and turn on your computer, just as you would to install Windows, but instead of going straight to installing the system, you get the option of trying it out first. You can then see how it looks, try the programs and, more importantly, make sure that all your devices work properly before installing it on your hard drive. Obviously, with a Live CD the system works much more slowly than it would if it were properly installed, but you do get to see it work in its full functionality before making the decision to adopt it. Secondly, installing Linux does not necessarily mean that you need to get rid of the operating system that you already have in your computer. For years, I had Linux installed side by side with Windows on my laptop (even if I barely used the latter), just in case I ever had any problems with Linux or if there was a Windows program that I needed to use and was not available for Linux. Most distributions, when you are installing them, are able to identify that there is already an operating system installed on your computer, and they offer the option of wiping it or of installing Linux by its side. If you choose the latter option, when you turn on your computer you will be able to choose from a menu which one of the operating systems you want to boot up. Moreover, Linux is able to 'see' the Windows partition (the part of the hard drive where Windows is installed), so you will be able to access your Windows files from Linux (the other way around, that is, accessing your Linux files from Windows, is not yet possible in a straightforward way as far as I know). Linux has software that can read most of the files used by the most popular Windows or Mac proprietary applications. For instance, there is no Microsoft Office for Linux, but you can use LibreOffice, which is a fully functional office suite that reads and saves Office files and displays them quite accurately (although not completely, particularly for documents with elaborate formatting). So should be able to access your office documents, music or videos from Linux without problems. Finally, if you really need to use a Windows program that is not available for Linux, there is a software for Linux called WINE which, when installed, allows you to install and run Windows programs within Linux. WINE is work in progress and not all Windows software runs seamlessly in Linux, but you can find out if the program

you are interested in runs well in Linux by checking the database available on the WINE website (http://appdb.winehq.org/). There are also commercial versions of WINE that focus on making sure that certain very popular Windows programs work without problems (such as CrossOver, http://www.codeweavers.com/ or Cedega, which focuses on running Windows games on Linux, http://www.cedega.com/).

So you have decided to try a Linux distribution. Which one should you choose? In part, the answer depends on what your values are, what kind of user you are, what type of software you like best, etc. Remember that the choice of distribution is different from the choice of desktop environment, and the latter is the one that determines how you interact with your computer and what software you run on a regular basis. In order to make decisions about these issues, you can search online and you will see that there are gazillions of opinions on what distribution and what desktop environment is best. If you do not want to make any decisions, I would tell you to go ahead and try Ubuntu (http://www.ubuntu.com). Ubuntu is the most popular distribution nowadays. It should give you a taste of what Linux has to offer and work out of the box. If you want to make a more informed decision, here you have a list of some alternatives and their general characteristics:

- Ubuntu: backed by a for-profit corporation called Canonical. Because it is so popular, it gives you ready access to many programs from the internet (through its included software center), and even free webspace to store your music and files in the cloud. Its popularity also means that if you have any problems, there's a myriad online resources where you can find help. Some people object to how Ubuntu does not give enough voice to the community and to how it does not contribute very much to the advancement of the goals of free software. Now Ubuntu comes with a desktop environment called Unity, but there are other flavors with other desktops as well, such as Kubuntu (with the KDE desktop, my favorite and probably the one that offers more configuration options), Xubuntu or Lubuntu (with the XFCE and the LXDE desktops respectively, which are light-weight, fast and simple).
- Linux Mint: it is not backed by any corporation, although it is based on Ubuntu (i.e. they basically take Ubuntu and configure it further). The goal of Linux Mint is to make it even simpler than Ubuntu, so it incorporates more non-free software, but everything should work like a charm and you should have to make very few decisions to get it working.
- Debian: Ubuntu (and therefore Linux Mint) is based on Debian, which can be thought of as 'the' distribution. It is one of the oldest ones, and it is managed completely by a community of developers in a decentralized and democratic way. It is a rather 'purist' distribution that attempts to follow the values of free software as far as possible (although you can install non-free software on it as well), to the extent that its version of Firefox is called Iceweasel, with a different icon, because of the copyright that the Mozilla Foundation tries to enforce on the Firefox logo. Debian comes in three different flavors: stable, testing, and unstable. Stable, as its own name says, is the one that you are supposed to be able to use without experiencing any problems. This means that is it thoroughly tested, and each version is not released until all major problems have been fixed. The result of this is that the stable Debian does not include the latest software and that releases are not very frequent (Ubuntu, in contrast, has a new release every six months). Testing and unstable include much more recent versions of all programs, and are updated almost daily, but this also implies that you are at risk of things breaking at any moment (and believe me, it happens).
- openSUSE: this distribution belongs to a different family of distributions from the previous three (which are all Debian based), so it has its own tools for installing and managing programs. It is backed by Novell, a corporation that sells a polished version of the distribution to firms.

Novell plays an important role in the development of openSUSE, but the distribution as a whole seems to be more community-driven than, for instance, Ubuntu. When you install it you can choose between the two major graphical desktop environments (KDE and Gnome), although the default is KDE.

- Fedora: Red Hat is the largest Linux-based corporation in the world, and they sell a distribution (and commercial support) to firms called Red Hat Linux Enterprise. Their free version for the community is Fedora, which is also part of a different family from the two mentioned above.
- Gentoo: this one was the first distro I used, which was probably a huge mistake. I chose it because it is completely community driven. It has an awesome way of installing packages through which you do not install the compiled file, but download the source code and compile it directly in your computer. This means that you get a system that is fully tailored to your machine, and thus runs faster and better. However, this also means that installing a package can take a **very** long time (since compilation for big programs is a slow process). It usually requires tweaking and extra work to make everything run properly, so unless you're adventurous and have lots of time to waste I would stay away from it (I still love it though).

There are hundreds of other options, and part of the fun of being a Linux user is to explore what is available and try different distributions until you find the one that is right for you. In the end, I have found that, although there is a learning curve as you have to adapt to a new operating system, Linux pays off as it gives you much choice and provides tools to do anything you are interested in on you computer at no cost. And, if you are partial to the ideals of free software (or open source), it offers you a way of being a consumer of software that promotes them.