Introduction to the GNU/Linux operating system

Josep Jorba Esteve

PID_00148470

UOC
Universitat Oberta de Catalunya
www.uoc.edu

Index

Introduction

GNU/Linux systems [Joh98] are no longer a novelty; they have a broad range of users and they are used in most work environments.

Their origin dates back to August 1991, when a Finnish student called Linus Torvalds announced on a news list that he had created his own operating system and that he was offering it to the community of developers for testing and suggesting improvements to make it more usable. This was the origin of the core (or kernel) of the operating system that would later come to be known as Linux.

Separately, the FSF (Free Software Foundation), through its GNU project, had been producing software that could be used for free since 1984. Richard Stallman (FSF member) considered free software that whose source code we could obtain, study, modify and redistribute without being obliged to pay for it. Under this model, the business does not reside in hiding the code, but rather in the complementary additional software, tailoring the software to clients and added services, such as maintenance and user training (the support we give) whether in the form of materials, books and manuals, or training courses.

The combination of the GNU software and the Linux kernel, is what has brought us to today's GNU/Linux systems. At present, the open source movements, through various organisations, such as the FSF, and the companies that generate the different Linux distributions (Red Hat, Mandrake, SuSe...), including large companies that offer support, such as HP, IBM or Sun, have given a large push to GNU/Linux systems to position them at a level of being capable of competing and surpassing many of the existing closed proprietary solutions.

GNU/Linux systems are no longer a novelty. GNU software started in the mid-eighties, the Linux kernel, in the early nineties. And Linux is based on tested UNIX technology with more than 30 years of history.

In this introductory unit we will revise some of the general ideas of the Open Source and Free Software movements, as well as a bit of the history of Linux and its shared origins with UNIX, from which it has profited from more than 30 years of research into operating systems.

1. Free Software and Open Source

Under the movements of Free Software and Open Source [OSIc] [OSIb] (also known as open code or open software), we find various different forms of software that share many common ideas.

A software product that is considered to be open source implies as its main idea that it is possible to access its source code, and to modify it and redistribute it as deemed appropriate subject to a specific open source license that defines the legal context.

As opposed to a proprietary type code, whereby the manufacturer (software company) will lock the code, hiding it and restricting the rights to it to itself, without allowing the possibility of any modification or change that has not been made previously by the manufacturer, open source offers:

a) access to the source code, whether to study it (ideal for education purposes) or to modify it, to correct errors, to adapt it or to add more features;

b) software that is free of charge: normally, the software, whether in binary form or source code form, can be obtained free of charge or for a modest sum to cover packaging and distribution costs and added value;

c) standards that prevent monopolies of proprietary software, avoiding dependency on a single choice of software manufacturer; this is more important for a large organisation, whether a company or a state, which cannot (or should not) put itself in the hands of a single specific solution and depend exclusively upon it;

d) a model of progress that is not based on hiding information but on sharing knowledge (like the scientific community) so as to progress more rapidly, and with better quality since decisions are based on the community's consensus and not on the whims of the companies that develop proprietary software.

Creating programs and distributing them together with the source code is nothing new. Since the beginnings of IT and the Internet, things had been done this way. However, the concept of open source itself, its definition and the drafting of the conditions it has to meet date back to the middle of 1997.

Eric Raymond and Bruce Perens promoted the idea. Raymond [Ray98] was the author of an essay called The Cathedral and the Bazaar, which discusses software development techniques used by the Linux community, headed by Linus Torvalds, and the GNU community of the Free Software Foundation (FSF), headed by Richard Stallman. Bruce Perens was the leader of the Debian project, which was working on creating a GNU/Linux distribution that integrated exclusively free software.

Note

See The Catedral and the Bazaar text at:

http://www.catb.org/~esr/writings/cathedral-bazaar/cathedral-bazaar/

Note

Two of the most important communities are the FSF, with its GNU software project, and the Open Source community, with Linux as its major project. GNU/Linux is the outcome of their combined work.

An important distinction between these communities lies in the definitions of open source and free software. [Deba] [PS02]

The Free Software Foundation [FSF] is a non-profit corporation founded by Richard Stallman, who believes that we should guarantee that programs are within everyone's reach free of charge, freely accessible and for use as each individual sees fit. The term *free* caused some reticence among companies. In English, the word can mean "without cost or payment" or "not under the control or in the power of another". The FSF sought both, but it was difficult to sell these two ideas to businesses; the main question was: "How can we make money with this?" The answer came from the Linux community (headed by Linus Torvalds), when they managed to obtain something that the GNU and FSF community had not yet achieved: a free operating system with an available source code. It was at that moment that the community decided to unite the various activities within the free software movement under a new name: open source software.

Open Source was registered as a certification brand, to which software products complying with its specifications could adhere. This did not please everybody and there tends to be a certain divide or controversy over the two groups of Open Source and FSF (with GNU), although really they have more things in common than not.

To some extent, for the exponents of free software (such as the FSF), open source is a false step, because it means selling out its ideals to the market, leaving the door open for software that was free to become proprietary. Those who back open source see it as an opportunity to promote software that would otherwise only be used by a minority, whereas through its worldwide diffusion and sharing, including with companies wishing to participate in open source, we find sufficient strength to challenge proprietary software.

However, the idea pursued by both movements is to increase the use of free software, thus offering an alternative to the sole solutions that large companies wish to impose. The differences are more than practical.

Having established the basic ideas of the open source community, we reached the point where we needed to clarify the criteria a software product should meet in order to qualify as open source. We had to base it on the definition of open source [OSIb] that was originally written by Bruce Perens in June 1997 in response to comments by developers of the Debian Linux distribution, which was subsequently re-edited (with minor changes) by the Open Source Initiative organisation (OSI). This body is responsible for controlling the open source definition and licenses.

Note

See the original definition of Open Source at:
http://www.opensource.org/docs/definition.php
In re-edition at:
http://www.opensource.org

Note

Open source is regulated by a public definition used as the basis for drafting its software licenses.

A small summary (interpretation) of the definition: Open source software [OSIb], or software with an open source code, must fulfil the following requirements:

1) The software may be copied, given away or sold to third parties, without requiring any payment for it.

2) The program must include source code and must allow distribution in source code as well as in compiled form. Or, in all events, there must be a well-publicised means of obtaining the source code (such as downloading via the Internet, for example). Deliberately obfuscated or intermediary forms of source code are not allowed. The license must guarantee that changes can be made.

3) The software license must allow modifications and derived works, and must allow them to be distributed under the same terms as the license of the original software. It allows the original code to be re-used.

4) The integrity of the author's source code may be required, in other words, modifications may be presented in the form of patches to the original code, or may be required to carry a different name or version number from the original. This protects which modifications can be attributed to the author. This point depends on what the software license says.

5) The license must not discriminate against any person or group of persons. Access to the software must not be restricted. In some cases there may be legal restrictions, as in the case of the United States for technology exports to third countries. If there are restrictions of this type, they must be mentioned.

6) No discrimination against fields of endeavour. The software can be used in any field of endeavour, even if it was not designed for that field. Commercial use is allowed; nobody can stop the software from being used for commercial purposes.

7) The license applies to everyone who receives the program.

8) If the software forms part of a larger product, it must keep the same license. This makes sure that parts are not separated in order to form proprietary software (in an uncontrolled manner). In the case of proprietary software, it must inform that it contains parts (stating which parts) of open source software.

9) The license must not restrict any incorporated or jointly distributed software, in other words, its incorporation should not act as a barrier for another jointly distributed software product. This is a controversial issue since it appears to contradict the preceding point, basically it says that anyone can take open source software and add it to their own software without this affecting its license conditions (for example proprietary), although, according to the preceding point, it would have to inform that there are parts of open source.

10) The license must be technology neutral, i.e. not restricted to certain devices or operating systems. It is not allowed to mention exclusive distribution means or to exclude possibilities. For example, under the open source licence, it is not possible to restrict the distribution to CD, FTP or web form.

This definition of open source is not a software license in itself, but rather a specification of the requirements that an open source software license must fulfil.

In order to be considered an open source program, the program's license must comply with the above specifications. The OSI is responsible for checking that licences meet the specifications. On the Open Source Licenses web page you can find the list of licenses [OSIa], of which one of the most famous and extensively used is the GPL (GNU Public License).

Note

Open Source Licences:
http://www.opensource.org/licenses/index.html

Under the GPL, the software may be copied and modified, but modifications must be made public under the same license, and it prevents the code becoming mixed with proprietary code so as to avoid proprietary code taking over parts of open source. There is the LGPL license, which is practically identical except that software with this license can be integrated into proprietary software. A classic example is the Linux C library (with LGPL license); if it were GPL, only free software could be developed, with the LGPL it can be used for developing proprietary software.

Many free software projects, or with part open source and part proprietary code, have their own license: Apache (based on BSD), Mozilla (MPL and NPL of Netscape) etc. Basically, when it comes to identifying the software as open source we can make our own license that complies with the above definition (of open source) or we can choose to license it under an already established license, or in the case of GPL, we are obliged for our license also to be GPL.

Having studied the concepts of open source and its licenses, we need to look at to what extent it is profitable for a company to work on or produce open source. If it were not attractive for companies, we would lose both a potential client and one of the leading software producers at the same time.

Open source is also attractive for companies, with a business model that emphasises a product's added value.

Open source offers various attractive benefits where companies are concerned:

a) For software developer companies, it poses a problem: how to make money without selling a product. A lot of money is spent on developing a program and then profit has to be made on top. Well, there is no simple answer, it is not possible with any type of software, the return lies in the type of software that can generate profit beyond the mere sale. Normally, a study will be made as to whether the application will become profitable if developed as open source (most will), based on the premises that we will have a reduced development cost (the community will help us), a reduced cost of maintenance or bug correction (the community can help with this quite quickly) and taking into account the number of users that the open source will provide, as well as the needs that they will have for our support or documentation services. If the balance is positive, then it will be viable to do without revenue from sales.

b) Increasing the number of users.

c) Obtaining greater development flexibility, the more people who intervene, the more people will be able to detect errors.

d) Revenue will mostly come from support, user training and maintenance.

e) Companies that use software need to take many parameters into consideration before choosing a software for managing tasks, such as performance, reliability, security, scalability and financial cost. And although it would seem that open source is already an evident choice on the cost basis, we must say that there is open source software capable of competing with (or even surpassing) proprietary software on any other parameter. Also, we need to take care with choosing the options or proprietary systems of a single manufacturer; we cannot rely solely on them (we may recall cases such as Sony's beta format video versus VHS, or the MicroChannel architecture of IBM for PCs).

We need to avoid using monopolies with their associated risks: lack of price competition, expensive services, expensive maintenance, little (or no) choice of options etc.

f) For private users it offers a large variety of software adapted for common uses, since a lot of the software has been conceived and implemented by people who wanted to do the same tasks but could not find the right software. Usually, in the case of a domestic user, a very important parameter is the software cost, but the paradox is that precisely domestic users are more prone to using proprietary software. Normally, domestic users will use illegal copies of software products; recent statistics show levels of 60-70% of illegal domestic copies. Users feel that merely by owning a home PC they are entitled to using the software in some countries for it. In these cases, we are dealing with illegal situations, which although they may not have been prosecuted, may be one day, or are attempted to be controlled through license systems (or product activations). Also, this has an indirect negative effects on free software, because if users are extensively using proprietary software, it forces everyone who wants to communicate them, whether banks, companies or public administrations, to use the same proprietary software too, and they do have to pay the product licenses. One of the most important battles for free software is to capture domestic users.

Note

Illegal domestic copies are also sometimes known as pirated copies.

g) Finally, states, as a particular case, can obtain important benefits from open source software, since it offers them quality software at ridiculous prices compared to the enormous cost of licenses for proprietary software. Moreover, open source software can easily integrate cultural aspects (of each country) such as language, for example. This last case is fairly problematic, since manufacturers of proprietary software refuse to adapt their applications in some regions – small states with their own language – or ask to be paid for doing so.

2. UNIX. A bit of history

As a predecessor to our GNU/Linux systems [Sta02], let's recall a bit about the history of UNIX [Sal94] [Lev]. Originally, Linux was conceived as a Minix clone (an academic implementation of UNIX for PC) and used some ideas developed in proprietary UNIX; but, in turn, it was developed in open source, and with a focus on domestic PCs. In this section on UNIX and in the following one on GNU/Linux, we will see how this evolution has brought us to current day GNU/Linux systems that are capable of competing with any proprietary UNIX and that are available for a large number of hardware architectures, from the simple PC to supercomputers.

Linux can be used on a broad range of machines. In the TOP500 list, we can find several supercomputers with GNU/Linux (see list on webpage top500.org): for example, the MareNostrum, in the Barcelona Supercomputing Center, a cluster, designed by IBM, with 10240 CPUs PowerPC with GNU/Linux operating system (adapted to the requirements of these machines). From the list we can see that overall supercomputers with GNU/Linux make up 75% of the list.

Note

We can see the TOP500 list of the fastest supercomputers at:
http://www.top500.org

UNIX started back in 1969 (we now have almost 40 years of history) in the Bell Telephone Labs (BTL) of AT&T. These had just withdrawn from a project called MULTICS, which was designed to create an operating system so that a large computer could support thousands of users simultaneously. BTL, General Electric, and MIT were involved in the project. But it failed, in part, because it was too ambitious for the time.

While this project was underway, two BTL engineers who were involved in MULTICS: Ken Thompson and Dennis Ritchie, found a DEC PDP7 computer that nobody was using, which only had an assembler and a loading program. Thompson and Ritchie developed as tests (and often in their free time) parts of UNIX, an assembler (of machine code) and the rudimentary kernel of the operating system.

That same year, in 1969, Thompson had the idea of writing a file system for the created kernel, in such a way that files could be stored in an ordered form in a system of hierarchical directories. Following various theoretical debates (which took place over about two months) the system was implemented in just a couple of days. As progress was made on the system's design, and a few more BTL engineers joined in, the original machine became too small, and they thought about asking for a new one (in those days they cost about 100,000 US dollars, which was a considerable investment). They had to make

up an excuse (since the UNIX system was a free time development) so they said they wanted to create a new text processor (an application that generated money at that time), so they were given approval to purchase a PDP11.

UNIX dates back to 1969, with over 30 years of technologies developed and used on all types of systems.

When the machine arrived, they were only given the CPU and the memory, but not the disk or the operating system. Thompson, unable to wait, designed a RAM disk in memory and used half of the memory as a disk and the other half for the operating system that he was designing. Once the disk arrived, they continued working on both UNIX and the promised text processor (the excuse). The text processor was a success (it was Troff, an editor language subsequently used for creating the UNIX man pages), and BTL started using the rudimentary UNIX with the new text processor, with BTL thus becoming the first user of UNIX.

At that time, the UNIX philosophy started to emerge [Ray02a]:

- Write programs that do one thing and do it well.
- Write programs to work together.
- Write programs to handle text streams.

Another important characteristic was that UNIX was one of the first systems conceived to be independent of the hardware architecture, and this has allowed it to be carried over to a large number of different hardware architectures.

In November 1971, as there were external users, the need to document what was being done resulted in the UNIX Programmer's Manual signed by Thompson and Richie. In the second edition (June 1972), known as V2 (the edition of the manuals was made to correspond with the UNIX version number), it was said that the number of UNIX installations had already reached 10. And the number continued to grow to about 50 in V5.

Then, at the end of 1973, it was decided to present the results at a conference on operating systems. And consequently, various IT centres and universities asked for copies of UNIX. AT&T did not offer support or maintenance to UNIX, which meant that users had to unite and share their knowledge by forming communities of UNIX users. AT&T decided to cede UNIX to universities, but did not offer them support or correct errors for them. Users started sharing their ideas, information on programs, bugs etc. They created an association called USENIX, meaning users of UNIX. Their first meeting in May 1974 was attended by a dozen people.

Note

See: http://www.usenix.org

One of the universities to have obtained a UNIX license was the University of California at Berkeley, where Ken Thompson had studied. In 1975, Thompson returned to Berkeley as a teacher bringing with him the latest version of UNIX. Two recently-graduated students, Chuck Haley and Bill Joy (nowadays one of the vice-presidents of SUN Microsystems), joined him and started to work together on a UNIX implementation.

One of the first things that they were disappointed with were the editors; Joy perfected an editor called EX, until transforming it into VI, a full screen visual editor. And the two developed a Pascal language compiler, which they added to UNIX. There was a certain amount of demand for this UNIX implementation, and Joy started to produce it as the BSD, Berkeley Software Distribution (or UNIX BSD).

BSD (in 1978) had a particular license regarding its price: it said that it corresponded to the cost of the media and the distribution it had at that time. Thus, new users ended up making some changes or incorporating features, selling their remade copies and after a certain amount of time, these changes were incorporated into the following version of BSD.

Joy also made a few more contributions to his work on the vi editor, such as handling text terminals in such a way that the editor was independent of the terminal where it was being used; he created the TERMCAP system as a generic terminals interface with controllers for each specific terminal, so that programs could be executed irrespective of the terminals using the interface.

The following step was to adapt it to different architectures. Until 1977, it could only be run on PDP machines; that year adaptations were made for machines of the time such as Interdata and IBM. UNIX Version 7 (V7 in June 1979) was the first portable one. This version offered many advances, as it included: *awk, lint, make, uucp*; the manual already had 400 pages (plus two appendices of 400 pages each). It also included the C compiler designed at BTL by Kernighan and Ritchie, which had been created to rewrite most of UNIX, initially in the assembler and then into C with the parts of the assembler that only depended on the architecture. Also included were an improved shell (Bourne shell) and commands such as: *find, cpio* and *expr*.

The UNIX industry also started to grow, and versions of UNIX (implementations) started to appear from companies such as: Xenix, a collaboration between Microsoft – which in its early days it also worked with UNIX versions – and SCO for Intel 8086 machines (the first IBM PC); new versions of BSD from Berkeley...

However, a new problem appeared when AT&T realised that UNIX was a valuable commercial product, the V7 license prohibited its study in academic institutions in order to protect its commercial secret. Until that time many universities used the UNIX source code in order to teach operating systems, and they stopped using it to teach only theory.

However, everyone found their own way of solving the problem. In Amsterdam, Andrew Tanenbaum (prestigious author of theory books on operating systems) decided to write a new UNIX-compatible operating system without using a single line of AT&T code; he called this new operating system Minix. This is what would subsequently be used in 1991 by a Finnish student to create his own version of UNIX, which he called Linux.

Bill Joy, who was still at Berkeley developing BSD (already in version 4.1), decided to leave to a new company called SUN Microsystems, where he finished working on BSD 4.2, which would later be modified to create SUN's UNIX, SunOS (around 1983). Every company started developing its own versions: IBM developed AIX, DEC - Ultrix, HP - HPUX, Microsoft/SCO - Xenix etc. As of 1980, UNIX began as a commercial venture, AT&T released a final version called UNIX System V (SV), on which as well as on the BSD 4.x, current UNIX are based, whether on the BSD or the System V branch. SV was revised several times and, for example, SV Release 4 was one of the most important ones. The result of these latest versions was that more or less all existing UNIX systems were adapted to each other; in practice they are versions of AT&T's System V R4 or Berkeley's BSD, adapted by each manufacturer. Some manufacturers specify whether their UNIX is a BSD or SV type, but in reality they all have a bit of each, since later several UNIX standards were drawn up in order to try and harmonise them; among these, we find IEEE POSIX, UNIX97, FHS etc.

Over time, the UNIX system split into several branches, of which the two main ones were AT&T's UNIX or System V, and the University of California's BSD. Most current UNIX systems are based on one or the other, or are a mixture of the two.

However, at that time, AT&T (SVR4) was undergoing legal proceedings as a telephone monopoly (it was the leading, if not the only, telephone company in the US), which forced it to split into several smaller companies, causing the rights to UNIX to start dancing between owners: in 1990 it was shared 50/50 by the Open Software Foundation (OSF) and UNIX International (UI), later, UNIX Systems Laboratories (USL), which denounced the University of Berkeley for its BSD copies, but lost, since the original license did not impose any ownership rights over the UNIX code. Later, the rights to UNIX were sold to Novell, which ceded a share to SCO, and as of today it is not very clear who owns them: they are claimed through different fronts by Novell, the OSF and SCO. A recent example of this problem is the case of SCO, which initiated a lawsuit against IBM because according to SCO, it had ceded parts of the UNIX source code to versions of the Linux kernel, which allegedly include some

original UNIX code. The result as of today is that the matter remains in the courts, with SCO turned into a pariah of the IT industry threatening Linux, IBM, and other proprietary UNIX users, with the assertion that they own the original rights to UNIX and that everyone else should pay for them. We will have to see how this case evolves, and the issue of UNIX rights along with it.

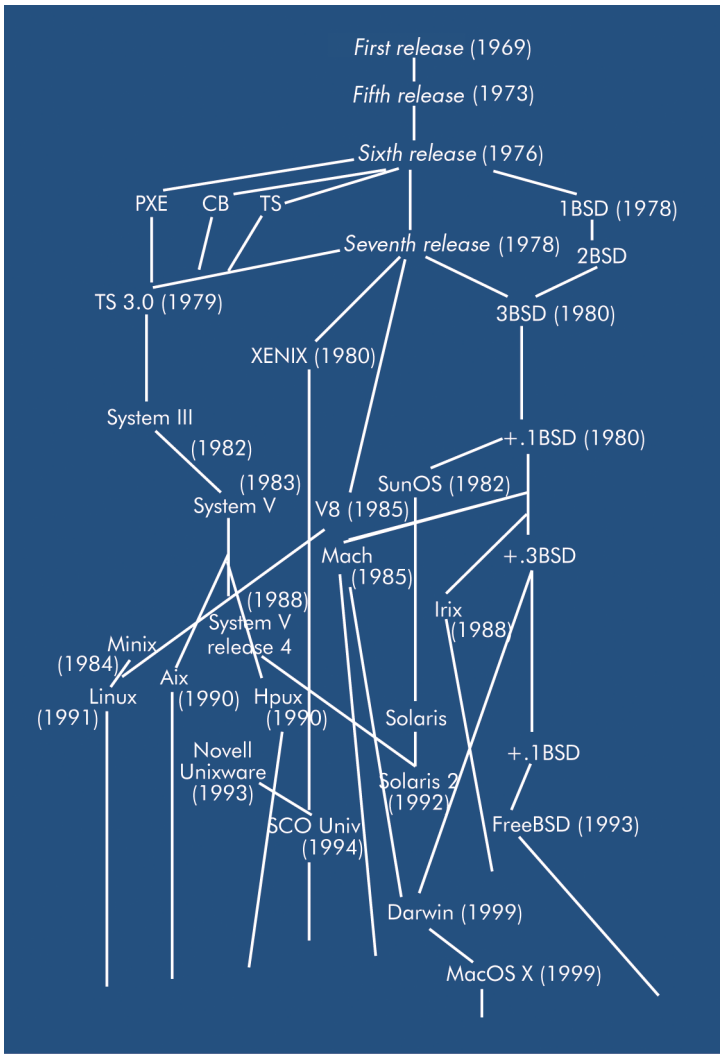

Figure 1. Historical summary of the different versions of UNIX

The current scenario with UNIX has changed a lot since Linux appeared in 1991, since as of 1995-99 it became a serious alternative to proprietary UNIX systems, due to the large number of hardware platforms that it supports and the extensive support for its progress of the international community and companies. Different proprietary versions of UNIX continue to survive in the market, because of their adaptation to industrial environments or for being the best operating system in the market, or because there are needs that can only be covered with UNIX and the corresponding hardware. Also, some proprietary UNIX are even better than GNU/Linux in terms of reliability and performance although the gap is shortening all the time, since companies with their own proprietary UNIX systems are showing more and more interest in GNU/Linux and offering some of their own developments for inclusion in

Linux. We can expect a more or less slow extinction of proprietary UNIX versions towards Linux-based distributions from manufacturers adapted to their equipment.

Overview of these companies:

Note

Many companies with proprietary UNIX participate in GNU/Linux and offer some of their developments to the community.

- SUN: it offers a UNIX implementation called Solaris (SunOS evolution). It started as a BSD system, but is now mostly System V with parts of BSD; it is commonly used on Sun machines with a SPARC architecture, and in multiprocessor machines (up to 64 processors). They promote GNU/Linux as a Java development environment and have a GNU/Linux distribution known as Java Desktop System, which has been widely accepted in a number of countries. Also, it has started using Gnome as a desktop, and offers financial support to various projects such as Mozilla, Gnome and OpenOffice. We should also mention its initiative with its latest version of Solaris UNIX, to almost totally free its code in Solaris version 10. Creating a community for Intel and SPARC architectures, called OpenSolaris, which has made it possible to create free Solaris distributions. On a separate note, we should mention recent initiatives (2006) to free the Java platform under GPL licenses, such as the OpenJDK project.

- IBM: it has its proprietary version of UNIX called AIX, which survives in some segments of the company's workstations and servers. At the same time, it firmly supports the Open Source community, by promoting free development environments (eclipse.org) and Java technologies for Linux, it incorporates Linux in its large machines and designs marketing campaigns to promote Linux. It also has influence among the community because of its legal defence against SCO, which accuses it of violating intellectual property alleging that it incorporated elements of UNIX in GNU/Linux.

- HP: it has its HPUX UNIX, but offers Linux extensive support, both in the form of Open Source code and by installing Linux on its machines. It is said to be the company that has made the most money with Linux.

- SGI: Silicon Graphics has a UNIX system known as IRIX for its graphics machines, but lately tends to sell machines with Windows, and possibly some with Linux. The company has been through difficulties and was about to break up. It offers support to the Linux community in OpenGL (3D graphics technology), different file systems and peripheral device control.

- Apple: joined the UNIX world recently (in the mid-nineties), when it decided to replace its operating system with a UNIX variant. The core known as Darwin derives from BSD 4.4; this Open Source kernel together with some very powerful graphic interfaces is what gives Apple its MacOS X operating system. Considered today to be one of the best UNIX and, at

least, one of the most appealing in its graphics aspect. It also uses a large amount of GNU software as system utilities.

- Linux distributors: both commercial and institutional, we will mention companies such as Red Hat, SuSe, Mandriva (formerly known as Mandrake), and non-commercial institutions such as Debian etc. These (the most widespread distributions) and the smallest ones are responsible for most of the development of GNU/Linux, with the support of the Linux community and the FSF with GNU software, in addition to receiving contributions from the abovementioned companies.

- BSD: although it is not a company as such, BSD versions continue to develop, as well as other BSD clone projects such as the FreeBSD, netBSD, OpenBSD (the UNIX considered to be the securest), TrustedBSD etc. These operating systems will also result in improvements or software incorporations to Linux sooner or later. Additionally, an important contribution is the Darwin kernel stemming from BSD 4.4, which Apple developed as the Open Source kernel of its MacOS X operating system.

- Microsoft: apart from hindering the development of UNIX and GNU/Linux, by setting up obstacles through incompatibilities between different technologies, it has no direct participation in the world of UNIX/Linux. However, in its early days it developed Xenix (1980) for PCs, based on an AT&T UNIX license, which although not sold directly was sold through intermediaries, such as SCO, which acquired control in 1987, and was renamed SCO UNIX (1989). As a curious side note, later it bought the rights to the UNIX license from SCO (which in turn had obtained them from Novell). Microsoft's motives for this acquisition are not clear, but some suggest that there is a relation with the fact that it supports SCO in the lawsuit against IBM. In addition, recently (2006), Microsoft reached agreements with Novell (current provider of the SuSe distribution and the OpenSuse community), in a number of bilateral decisions to give business promotion to both platforms. But part of the GNU/Linux community remains sceptical due to the potential implications for Linux intellectual property and issues that could include legal problems for the use of patents.

Note

Open letter from Novell to the GNU/Linux community
http://www.novell.com/linux/microsoft/community_open_letter.html

Another interesting historical anecdote is that together with a company called UniSys, they launched a marketing campaign on how to convert UNIX systems to Windows systems; and although its purpose may be more or less commendable, a curious fact is that the original web server of the business was on a FreeBSD machine with Apache. Occasionally, it also pays "independent" companies (some would say they are not very independent) to conduct comparative performance analyses between UNIX/Linux and Windows.

As a general summary, some comments that tend to appear in UNIX bibliography point to the fact that UNIX is technically a simple and coherent system designed with good ideas that were put into practice, but we should not forget that some of these ideas were obtained thanks to the enthusiastic support offered by a large community of users and developers who collaborated by sharing technology and governing its evolution.

And since history tends to repeat itself, currently that evolution and enthusiasm continues with GNU/Linux systems.

3. GNU/Linux systems

Twenty years ago the users of the first personal computers did not have many operating systems to choose from. The market for personal computers was dominated by Microsoft DOS. Another possibility was Apple's MAC, but at an exorbitant cost in comparison to the rest. Another important option reserved to large (and expensive) machines was UNIX.

A first option to appear was MINIX (1984), created from scratch by Andrew Tanenbaum, for educational purposes in order to teach how to design and implement operating systems [Tan87] [Tan06].

MINIX was conceived for running on an Intel 8086 platform, which was very popular at the time as it was the basis for the first IBM PCs. The main advantage of this operating system stemmed from its source code, which was accessible to anyone (twelve thousand lines of code for assembler and C), and available from Tanenbaum's teaching books on operating systems [Tan87]. However, MINIX was an educational tool rather than an efficient system designed for professional performance or activities.

In the nineties, the Free Software Foundation (FSF) and its GNU project, motivated many programmers to promote quality and freely distributed software. And aside from utilities software, work was being done on the kernel of an operating system known as HURD, which would take several years to develop.

Meanwhile, in October 1991, a Finnish student called Linus Torvalds presented version 0.0.1 of his operating system's kernel, which he called Linux, designed for Intel 386 machines, and offered under a GPL license to communities of programmers and the Internet community for testing, and if they liked it, for helping with its development. There was such enthusiasm that in no time a large number of programmers were working on the kernel or on applications for it.

Some of the features that distinguished Linux from other operating systems of the time and which continue to be applicable, and others inherited from UNIX could be:

a) It is an open source operating system: anyone can have access to its sources, change them and create new versions that can be shared under the GPL license (which, in fact, makes it free software).

b) Portability: like the original UNIX, Linux is designed to depend very little on the architecture of a specific machine; as a result, Linux is, mostly, independent from its destination machine and can be carried to practically any

architecture with a C compiler such as the GNU *gcc*. There are just small parts of assembler code and a few devices that depend on the machine, which need to be rewritten at each port to a new architecture. Thanks to this, GNU/Linux is one of the operating systems running on the largest number of architectures: Intel x86 and IA64, AMD x86 and x86_64, Sun's SPARC, MIPS of Silicon, PowerPC (Apple), IBM S390, Alpha by Compaq, m68k Motorola, Vax, ARM, HPPArisc...

c) Monolith-type kernel: the design of the kernel is joined into a single piece but is conceptually modular in its different tasks. Another school of design for operating systems advocates microkernels (Mach is an example), where services are implemented as separate processes communicated by a more basic (micro) kernel. Linux was conceived as a monolith because it is difficult to obtain good performance from microkernels (it is a hard and complex task). At the same time, the problem with monoliths is that when they grow they become very large and untreatable for development; dynamic load modules were used to try to resolve this.

Note

Original Mach project:
http://www.cs.cmu.edu/afs/cs/project/mach/public/www/mach.html

d) Dynamically loadable modules: these make it possible to have parts of the operating system, such as file systems, or device controllers, as external parts that are loaded (or linked) with the kernel at run-time on-demand. This makes it possible to simplify the kernel and to offer these functionalities as elements that can be separately programmed. With this use of modules, Linux could be considered to be a mixed kernel, because it is monolithic but offers a number of modules that complement the kernel (similar to the microkernel concepts).

e) System developed by an Internet-linked community: operating systems had never been developed so extensively and dispersely, they tend not to leave the company that develops them (in the case of proprietary systems) or the small group of academic institutions that collaborate in order to create one. The phenomenon of the Linux community allows everyone to collaborate as much as their time and knowledge will permit. The result is: hundreds to thousands of developers for Linux. Additionally, because of its open-source nature, Linux is an ideal laboratory for testing ideas for operating systems at minimum cost; it can be implemented, tested, measures can be taken and the idea can be added to the kernel if it works.

Projects succeeded each other and – at the outset of Linux with the *kernel* – the people of the FSF, with the GNU utility software and, above all, with the (*GCC*) C compiler, were joined by other important projects such as XFree (a PC version of X Window), and desktop projects such as KDE and Gnome. And the Internet development with projects such as the Apache web server, the Mozilla navigator, or MySQL and PostgreSQL databases, ended up giving the initial Linux kernel a sufficient coverage of applications to build the GNU/Linux systems and to compete on an equal level with proprietary systems. And to convert the GNU/Linux systems into the paradigm of Open Source software.

GNU/Linux systems have become the tip of the spear of the Open Source community, for the number of projects they have been capable of drawing together and concluding successfully.

The birth of new companies that created GNU/Linux distributions (packaging of the kernel + applications) and supported it, such as Red Hat, Mandrake, SuSe, helped to introduce GNU/Linux to reluctant companies and to initiate the unstoppable growth we are now witnessing today.

We will also comment on the debate over the naming of systems such as GNU/Linux. The term *Linux* is commonly used (in order to simplify the name) to identify this operating system, although in some people's opinion it undermines the work done by the FSF with the GNU project, which has provided the system's main tools. Even so, the term *Linux*, is extensively used commercially in order to refer to the full operating system.

Note

GNU and Linux by Richard-Stallman:
http://www.gnu.org/gnu/ linux-and-gnu.html.

In general, a more appropriate term that would reflect the community's participation, is *Linux,* when we are referring only to the operating system's kernel. This has caused a certain amount of confusion because people talk about the Linux operating system in order to abbreviate. When we work with a GNU/Linux operating system, we are working with a series of utilities software that is mostly the outcome of the GNU project on the Linux kernel. Therefore, the system is basically GNU with a Linux kernel.

The purpose of the FSF's GNU project was to create a UNIX-style free software operating system called GNU [Sta02].

In 1991, Linus Torvalds managed to join his Linux kernel with the GNU utilities when FSF still didn't have a kernel. GNU's kernel is called HURD, and quite a lot of work is being done on it at present, and there are already beta versions available of GNU/HURD distributions (see more under the chapter "Kernel Administration").

It is estimated that in a GNU/Linux distribution there is 28% of GNU code and 3% that corresponds to the Linux kernel code; the remaining percentage corresponds to third parties, whether for applications or utilities.

To highlight GNU's contribution [FSF], we can look at some of its contributions included in GNU/Linux systems:

- The C and C++ compiler (*GCC*)
- The bash shell
- The Emacs editor (GNU Emacs)
- The postscript interpreter (*ghostscript*)

- The standard C library (*GNU C library*, or *glibc*)
- The debugger (*GNU gdb*)
- *Makefile* (*GNU make*)
- The assembler (*GNU assembler* or *gas*)
- The linker (*GNU linker* or *gld*)

GNU/Linux systems are not the only systems to use GNU software; for example, BSD systems also incorporate GNU utilities. And some proprietary operating systems such as MacOS X (Apple) also use GNU software. The GNU project has produced high quality software that has been incorporated into most UNIX-based system distributions, both free and proprietary.

It is only fair for the world to recognise everyone's work by calling the systems we will deal with GNU/Linux.

4. The profile of the systems administrator

Large companies and organisations rely more and more on their IT resources and on how these are administered and adapted to the required tasks. The huge increase in distributed networks, with server and client machines, has created a large demand for a new job in the marketplace: the so-called systems administrator.

A systems administrator is responsible for a large number of important tasks. The best systems administrators tend to have a fairly general practical and theoretical background. They can perform tasks such as: cabling installations or repairs; installing operating systems or applications software; correcting systems problems and errors with both hardware and software; training users; offering tricks or techniques for improving productivity in areas ranging from word processing applications to complex CAD or simulator systems; financially appraising purchases of hardware and software equipment; automating a large number of shared tasks, and increasing the organisation's overall work performance.

The administrator can be considered the employee who helps the organisation to make the most of the available resources, so that the entire organisation can improve.

The relationship with the organisation's end users can be established in several ways: either through training users or by offering direct assistance if problems should arise. The administrator is the person responsible for ensuring that the technologies employed by users function properly, meaning that the systems satisfy users' expectations and do the tasks they need to fulfil.

Years ago, and even nowadays, many companies and organisations had no clear vision of the system administrator's role. When business computing was in its early days (in the eighties and nineties), the administrator was seen as the person who understood computers (the "guru") responsible for installing machines and monitoring or repairing them in case there were any problems. Normally, the job was filled by a versatile computer technician responsible for solving problems as and when they appeared. There was no clear-cut profile for the job because extensive knowledge was not required, just basic knowledge of a dozen (at most) applications (the word processor, spreadsheet, database etc.), and some basic hardware knowledge was enough for day to day tasks. Therefore, anyone in the know who understood the issue could do the job, meaning that usually administrators were not traditional computer technicians and often knowledge was even communicated orally between an existing or older administrator and a trainee.

This situation reflected to some extent the prehistory of systems administration (although there are still people who think that it is basically the same job). Nowadays, in the age of Internet and distributed servers, a systems administrator is a professional (employed full-time exclusively for this purpose) who offers services in the field of systems software and hardware. The systems administrator has to execute several tasks destined for multiple IT systems, mostly heterogeneous, with a view to making them operative for a number of tasks.

Currently, systems administrators need general knowledge (theoretical and practical) in a diversity of fields, from network technologies, to operating systems, diverse applications, basic programming in a large number of programming languages, extensive hardware knowledge – regarding the computer itself as well as peripherals – Internet technologies, web-page design, database management etc. And normally the profile is sought to correspond to the company's area of work, chemistry, physics, mathematics etc. Therefore, it is no surprise that any medium to large company has turned away from employing the available dogsbody towards employing a small group of professionals with extensive knowledge, most with a university degree, assigned to different tasks within the organisation.

The systems administrator must be capable of mastering a broad range of technologies in order to adapt to a variety of tasks that can arise within an organisation.

Because of the large amount of knowledge required, unsurprisingly there are several sub-profiles for a systems administrator. In a large organisation it is common to find different operating systems administrators (UNIX, Mac, or Windows): database administrator, backup copies administrator, IT security administrator, user help administrators etc.

In a smaller organisation, all or some of the tasks may be allocated to one or a few administrators. The UNIX systems (or GNU/Linux) administrators would be a part of these (unless there is one administrator responsible for all tasks). Normally, the administrator's working platform is UNIX (or GNU/Linux in our case), which requires enough specific elements to make this job unique. UNIX (and its variants) is an open and very powerful operating system and, like any software system, requires a certain level of adaptation, configuration and maintenance in the tasks for which it will be used. Configuring and maintaining an operating system is a serious job, and in the case of UNIX can become quite frustrating.

Some important issues covered include the following:

a) The fact that the system is very powerful also means that there is a lot of potential for adapting it (configuring it) for the tasks we need to do. We will have to evaluate what possibilities it can offer us and which are appropriate for our final objective.

b) A clear example of an open system is GNU/Linux, which will offer us permanent updates, whether to correct system bugs or to incorporate new features. And, obviously, all of this has a considerable direct impact on the maintenance cost of administration tasks.

c) Systems can be used for critical cost tasks, or in critical points of the organisation, where important failures that would slow down or impede the functioning of the organisation cannot be allowed.

d) Networks are currently an important point (if not the most important), but it is also a very critical problems area, due both to its own distributed nature and to the system's complexity for finding, debugging and resolving problems that can arise.

e) In the particular case of UNIX, and our GNU/Linux systems, the abundance of both different versions and distributions, adds more problems to their administration, because it is important to know what problems and differences each version and distribution has.

In particular, system and network administration tasks tend to have different features, and sometimes they are handled separately (or by different administrators). Although we could also look at it as the two sides of the same job, with the system itself (machine and software) on the one hand, and the environment (network environment) where the system coexists, on the other.

Usually, network administration is understood to mean managing the system as part of the network and refers to the nearby services or devices required for the machine to function in a network environment; it does not cover network devices such as switches, bridges or hubs or other network devices, but basic knowledge is essential in order to facilitate administration tasks.

In this course, we will first deal with the local aspects of the system itself and secondly we will look at the tasks of administering a network and its services.

We have already mentioned the problem of determining exactly what a systems administrator is, because in the IT job market it is not very clear. It was common to ask for systems administrators based on categories (established by companies) of programmer or software engineer, which are not entirely appropriate.

A programmer is basically a producer of code; in this case, an administrator would not need to produce much, because it may be necessary for some tasks but not for others. Normally, it is desirable for an administrator to have more or less knowledge depending on the job category:

a) Some qualification or university degree, preferably in IT, or in a field directly related to the company or organisation.
The profile of a systems administrator tends to include computer science or enginnering studies or an education related to the organisation's sphere of activity together with proven experience in the field and broad knowledge of heterogeneous systems and network technologies.

b) It is common to ask for 1 to 3 years of experience as an administrator (unless the job is as an assistant of an already existing administrator). Experience of 3 to 5 years may also be requested.

c) Familiarity with or broad knowledge of network environments and services. TCP/IP protocols, ftp, telnet, ssh, http, nfs, nis, ldap services etc.

d) Knowledge of script languages for prototyping tools or rapid task automation (for example, shell scripts, Perl, tcl, Python etc.) and programming experience in a broad range of languages (C, C++, Java, Assembler etc.).

e) Experience in large applications development in any of these languages may be requested.

f) Extensive knowledge of the IT market, for both hardware and software, in the event of having to evaluate purchases or install new systems or complete installations.

g) Experience with more than one version of UNIX (or GNU/Linux systems), such as Solaris, AIX, AT&T System V, BSD etc.

h) Experience of non-UNIX operating systems, complementary systems that may be found in the organisation: Windows 9x/NT/2000/XP/Vista, Mac OS, VMS, IBM systems etc.

i) Solid knowledge of UNIX design and implementation, paging mechanisms, exchange, interprocess communication, controllers etc., for example, if administration tasks include optimising systems (tuning).

j) Knowledge and experience in IT security: construction of firewalls, authentication systems, cryptography applications, file system security, security monitoring tools etc.

k) Experience with databases, knowledge of SQL etc.

l) Installation and repair of hardware and/or network cabling and devices.

5. Tasks of the administrator

As we have described, we could divide the tasks of a GNU/Linux administrator (or UNIX in general) [Lev02] into two main parts: system administration and network administration. In the following points we will show in summary what these tasks in general consist of for GNU/Linux (or UNIX) systems; most part of the content of this course manual will be treated in a certain amount of detail; most of these administration tasks will be developed in this course manual; for reasons of space or complexity, other parts of the tasks will be explained superficially.

Administration tasks encompass a series of techniques and knowledge, of which this manual only reflects the tip of the iceberg; in any case, the bibliography attached to each unit offers references to expand on those subjects. As we will see, there is an extensive bibliography for almost every point that is treated.

System administration tasks could be summarised, on the one hand, as to administer the local system, and on the other hand, to administer the network.

Local system administration tasks (in no specific order)

- Switching the system on and off: any UNIX-based system has configurable switching on and off systems so that we can configure what services are offered when the machine switches on and when they need to be switched off, so that we can program the system to switch off for maintenance.

- Users and groups management: giving space to users is one of the main tasks of any systems administrator. We will need to decide what users will be able to access the system, how, and with what permissions; and to establish communities through the groups. A special case concerns system users, pseudousers dedicated to system tasks.

- Management of the system's resources: what we offer, how we offer it and to whom we give access.

- Management of the file system: the computer may have different resources for storing data and devices (diskettes, hard disks, optical disk drives etc.) with different file access systems. They may be permanent or removable or temporary, which will mean having to model and manage the process

of installing and uninstalling the file systems offered by related disks or devices.

- System quotas: any shared resource will have to be administered, and depending on the number of users, a quota system will need to be established in order to avoid an abuse of the resources on the part of users or to distinguish different classes (or groups) of users according to greater or lesser use of the resources. Quota systems for disk space or printing or CPU use are common (used computing time).

- System security: local security, about protecting resources against undue use or unauthorised access to system data or to other users or groups data.

- System backup and restore: (based on the importance of the data) periodic policies need to be established for making backup copies of the systems. Backup periods need to be established in order to safeguard our data against system failures (or external factors) that could cause data to become lost or corrupted.

- Automation of routine tasks: many routine administration tasks or tasks associated to daily use of the machine can be automated easily, due to their simplicity (and therefore, due to the ease of repeating them) as well as their timing, which means that they need to be repeated at specific intervals. These automations tend to be achieved either through programming in an interpreted language of the script type (shells, Perl etc.), or by inclusion in scheduling systems (crontab, at...).

- Printing and queue management: UNIX systems can be used as printing systems to control one or more printers connected to the system, as well as to manage the work queues that users or applications may send to them.

- Modem and terminals management. These devices are common in environments that are not connected to a local network or to broadband:
  - Modems make it possible to connect to a network through an intermediary (the ISP or access provider) or to our system from outside, by telephone access from any point of the telephone network.

  - In the case of terminals, before the introduction of networks it was common for the UNIX machine to be the central computing element, with a series of dumb terminals that were used merely to visualise information or to allow information to be entered using external keyboards; these tended to be series or parallel type terminals. Nowadays, they are still common in industrial environments and our GNU/Linux desktop system has a special feature: the virtual text terminals accessed using the Alt+Fxx keys.

- System accounting (or log): to check that our system is functioning correctly, we need to enforce log policies to inform us of potential failures of the system or performance of an application, service or hardware resource. Or to summarise spent resources, system uses or productivity in the form of a report.

- System performance tunning: system tuning techniques for an established purpose. Frequently, a system is designed for a specific job and we can verify that it is functioning correctly (using logs, for example), in order to check its parameters and adapt them to the expected service.

- System tailoring: kernel reconfiguration. In GNU/Linux, for example, the kernels are highly configurable, according to the features we wish to include and the type of devices we have or hope to have on our machine, in addition to the parameters that affect the system's performance or are obtained by the applications.

Network administration tasks

- Network interface and connectivity: the type of network interface we use, whether access to a local network, a larger network, or broadband type connection with DSL or ISDN technologies. Also, the type of connectivity we will have, in the form of services or requests.

- Data routing: data that will circulate, where from or where to, depending on the available network devices, and the machine's functions within the network; it may be necessary to redirect traffic from/to one or more places.

- Network security: a network, especially one that is open (like Internet) to any external point, is a possible source of attacks and, therefore, can compromise the security of our systems or our users' data. We need to protect ourselves, detect and prevent potential attacks with a clear and efficient security policy.

- Name services: a network has an infinite number of available resources. Name services allow us to name objects (such as machines and services) in order to be able to locate them. With services such as DNS, DHCP, LDAP etc., we will be able to locate services or equipment later...

- NIS (Network Information Service): large organisations need mechanisms to organise and access resources efficiently. Standard UNIX forms, such as user logins controlled by local passwords, are effective when there are few machines and users, but when we have large organisations, with hierarchical structures, users that can access multiple resources in a unified fashion or separately with different permissions... simple UNIX methods are clearly insufficient or impossible. Then we need more efficient systems

in order to control all of this structure. Services such as NIS, NIS+, LDAP help us to organise this complexity in an effective manner.

- NFS (Network Fylesystems): often, on network system structures information needs to be shared (such as files themselves) by all or some users. Or simply, because of the physical distribution of users, access to the files is required from any point of the network. Network file systems (such as NFS) offer us transparent access to files, irrespective of our location on the network.

- UNIX remote commands: UNIX has transparent network commands, in the sense that irrespective of the physical connection it is possible to run commands that move information along the network or that allow access to some of the machines' services. These commands tend to have an "r" in front of them, meaning "remote", such as: *rcp, rlogin, rsh, rexec* etc., which remotely enable the specified functionalities on the network.

- Network applications: applications for connecting to network services, such as telnet (interactive access), FTP (file transmission), in the form of a client application that connects to a service served from another machine. Or that we can serve ourselves with the right server: telnet server, FTP server, web server etc.

- Remote printing: access to remote printing servers, whether directly to remote printers or to other machines that offer their own local printers. Network printing transparently for the user or application.

- E-mail: one of the main services offered by UNIX machines is the e-mail server, which can either store mail or redirect it to other servers, if it is not directed at its system's own users. In the case of the web, a UNIX system similarly offers an ideal web platform with the right web server. UNIX has the biggest market share with regards to e-mail and web servers, and this is one of its main markets, where it has a dominating position. GNU/Linux systems offer open source solutions for e-mail and web, representing one of its main uses.

- X Window: a special model of interconnection is the graphics system of the GNU/Linux systems (and most of UNIX), X Window. This system allows total network transparency and operates under client-server models; it allows an application to be totally unlinked from its visualisation and interaction with it by means of input devices, meaning that these can be located anywhere on the network. For example, we may be executing a specific application on one UNIX machine while on another we may visualise the graphic results on screen and we may enter data using the local keyboard and mouse in a remote manner. Moreover, the client, called client X, is just a software component that can be carried onto other operating systems, making it possible to run applications on one UNIX ma-

chine and to visualise them on any other system. So-called X terminals are a special case – they are basically a type of dumb terminal that can only visualise or interact (using a keyboard and mouse) with a remotely run application.

6. GNU/Linux distributions

When speaking about the origins of GNU/Linux, we have seen that there is no clearly defined unique operating system. On the one hand, there are three main software elements that make up a GNU/Linux system:

1) The Linux kernel: as we have seen, the kernel is just the central part of the system. But without the utility applications, shells, compilers, editors etc. we could not have a complete system.

2) GNU applications: Linux's development was complemented by the FSF's existing software under the GNU project, which provided editors (such as *emacs*), a compiler (*gcc*) and various utilities.

3) Third party software: normally open source. Additionally, any GNU/Linux system incorporates third party software which makes it possible to add a number of extensively used applications, whether the graphics system itself X Windows, servers such as Apache for web, navigators etc. At the same time, it may be customary to include some proprietary software, depending on to what extent the distribution's creators want the software to be free.

Because most of the software is open source or free, whether the kernel, GNU or third-party software, normally there is a more or less rapid evolution of versions, either through the correction of bugs or new features. This means that in the event of wanting to create a GNU/Linux system, we will have to choose which software we wish to install on the system, and which specific versions of that software.

The world of GNU/Linux is not limited to a particular company or community, which means that it offers everyone the possibility of creating their own system adapted to their own requirements.

Normally, among these versions there are always some that are stable and others that are under development in phase alpha or beta, which may contain errors or be unstable, which means that when it comes to creating a GNU/Linux system, we will have to be careful with our choice of versions. Another additional problem is the choice of alternatives, the world of GNU/Linux is sufficiently rich for there to be more than one alternative for the same software product. We need to choose among the available alternatives, incorporating some or all of them, if we wish to offer the user freedom of choice to select their software.

Example

We find a practical example with the X Window desktop managers, which, for example, offer us (mainly) two different desktop environments such as Gnome and KDE; both have similar characteristics and similar or complementary applications.

In the case of a distributor of GNU/Linux systems, whether commercial or non-profit, the distributor's responsibility is to generate a system that works, by selecting the best software products and versions available.

In this case, a GNU/Linux distribution [Dis] is a collection of software that makes up an operating system based on the Linux kernel.

An important fact that needs to be taken into account, and that causes more than a little confusion, is that because each of the distribution's software packages will have its own version (irrespective of the distribution it is located on) the allocated distribution number does not correspond to the software packages versions.

Example

Let's look at a few versions as an example (the versions that appear refer to the end of 2003):

a) Linux kernel: we can currently find distributions that offer one or more kernels, such as those of the old series 2.4.x or generally, the latest 2.6.x in revisions of varying recentness (the number x).

b) The X Window graphics option, in open source version, which we can find on practically all GNU/Linux systems, whether as some residual versions of Xfree86 such as the ones handled by 4.x.y versions or as the new Xorg project (a fork of the previous one in 2003), which is more popular in various versions 6.x or 7.x.

c) Desktop or windows manager: we can have Gnome or KDE, or both; Gnome with versions 2.x or KDE 3.x.y.

For example, we could obtain a distribution that included kernel 2.4, with XFree 4.4 and Gnome 2.14; or another, for example, kernel 2.6, Xorg 6.8, KDE 3.1. Which is better? It is difficult to compare them because they combine a mixture of elements and depending on how the mixture is made, the product will come out better or worse, and more or less adapted to the user's requirements. Normally, the distributor will maintain a balance between the system's stability and the novelty of included versions. As well as provide attractive application software for the distribution's users, whether it is of a general nature or specialized in any specific field.

In general, we could analyse the distributions better on the basis of the following headings, which would each have to be checked:

a) Version of the Linux kernel: the version is indicated by numbers *X.Y.Z,* where normally *X* is the main version, which represents important changes to the kernel; *Y* is the secondary version and usually implies improvements in the kernel's performance: *Y* is even for stable kernels and uneven for developments or tests. And *Z* is the build version, which indicates the revision number of *X.Y,* in terms of patches or corrections made. Distributors tend not to include the kernel's latest version, but rather the version that they have tested most frequently and have checked is stable for the software and components that they include. This classical numbering scheme (which was observed for branches 2.4.x, until the first ones of 2.6), was slightly modified to adapt to the fact that the kernel (branch 2.6.x) becomes more stable and that there are fewer revisions all the time

(meaning a leap in the first numbers), but development is continuous and frenetic. Under the latest schemes, fourth numbers are introduced to specify in Z minor changes or the revision's different possibilities (with different added patches). The version thus defined with four numbers is the one considered to be stable. Other schemes are also used for the various test versions (normally not advisable for production environments), using suffixes such as -rc (*release candidate*), -mm, experimental kernels testing different techniques, or -git, a sort of daily snapshot of the kernel's development. These numbering schemes are constantly changing in order to adapt to the kernel community's way of working, and its needs in order to speed up the kernel's development.

b) Packaging format: this is the mechanism used for installing and administering the distribution's software. It tends to be known for the format of the software packages it supports. In this case we normally find RPM, DEB, tar.gz, mdk formats, and although every distribution usually offers the possibility of using different formats, it tends to have a default format. The software normally comes with its files in a package that includes information on installing it and possible dependencies on other software packages. The packaging is important if third party software that does not come with the distribution is used, since the software may only be found in some package systems, or even in just one.

c) File system structure: the main file system structure (/) tells us where we can find our fils (or the system's files) in the file system. GNU/Linux and UNIX have some file location standards (as we will see in the tools unit), such as FHS (*filesystem hierarchy standard*) [Lin03b]. Therefore, if we have an idea of the standard, we will know where to find most of the files; then it depends whether the distribution follows it more or less and tells us of any changes that have been made.

d) System boot scripts: UNIX and GNU/Linux systems incorporate boot scripts (or shell scripts) that indicate how the machine should start up, what will be the process (or phases) followed, and what has to be done at each step. There are two models for this start up, those of SysV or BSD (this is a difference between the two main UNIX branches); and every distribution may choose one or the other. Although both systems have the same functionality, they differ in the details, and this will be important for administration issues (we will look at this under local administration). In our case, the analysed systems, both Fedora and Debian, use the SysV system (which we will look at under the unit on local administration), but there are other distributions such as Slackware that use the other BSD system. And there are some proposals (like Ubuntu's Upstart) of new options for this start up aspect.

e) Versions of the system library: all the programs (or applications) that we have on the system will depend on a (bigger or smaller) number of system

libraries for running. These libraries, normally of two types, whether static joined to the program (*libxxx.a* files) or dynamic runtime loaded (*libxxx.so* files), provide a large amount of utility or system code that the applications will use. Running an application may depend on the existence of corresponding libraries and the specific version of these libraries (it is not advisable, but can happen). A fairly common case affects the GNU C library, the standard C library, also known as *glibc*. An application may ask us for a specific version of *glibc* in order to be run or compiled. It is a fairly problematic issue and therefore, one of the parameters valued by the distribution is knowing what version of the *glibc* it carries and possible additional versions that are compatible with old versions. The problem appears when trying to run or compile an old software product on a recent distribution, or a very new software product on an old distribution.
The biggest change occurred in moving to a *glibc* 2.0, in which all the programs had to be recompiled in order to run correctly, and in the different revisions numbered 2.x there have been a few minor modifications that could affect an application. In many cases, the software packages check whether the correct version of *glibc* is available or the name itself mentions the version that needs to be used (example: *package-xxx-glibc2.rpm*).

f) X Window desktop: the X Window system is the graphics standard for desktop visualisation in GNU/Linux. It was developed by MIT in 1984 and practically all UNIX systems have a version of it. GNU/Linux distributions have different versions such as Xfree86 or Xorg. Usually, X Window is an intermediary graphic layer that entrusts another layer known as the windows manager to visualise its elements. Also, we can combine the windows manager with a variety of application programs and utilities to create what is known as a desktop environment.
Linux mainly has two desktop environments: Gnome and KDE. Each one is special in that it is based on a library of its own components (the different elements of the environment such as windows, buttons, lists etc.): *gtk+* (in Gnome) and *Qt* (in KDE), which are the main graphics libraries used to program applications in these environments. But in addition to these environments, there are many more windows or desktop managers: XCFE, Motif, Enlightement, BlackIce, FVWM etc., meaning that there is a broad range of choice. In addition, each one makes it possible to change the appearance (look & feel) of the windows and components as users' desire, or even to create their own.

g) User software: software added by the distributor, mostly Open Source, for common tasks (or not so common, for highly specialised fields).
Common distributions are so large that we can find hundreds to thousands of these extra applications (many distributions have 1 to 4 CDs – approximately 1 DVD of extra applications). These applications cover practically all fields, whether domestic, administrative or scientific. And some distributions add third party proprietary software (for example, in the case

of an Office-type suite), server software prepared by the distributor, for example an e-mail server, secure web server etc.
This is how each distributor tends to release different versions of their distribution, for example, sometimes there are distinctions between a personal, professional or server version.
Often, this financial cost does not make sense, because the standard software is sufficient (with a bit of extra administration work); but it can be interesting for companies because it reduces server installation times and maintenance and also optimises certain critical servers and applications for the company's IT management.

6.1. Debian

The case of Debian [Debb] is special, in the sense that it is a distribution delivered by a community with no commercial objectives other than to maintain its distribution and promote the use of free and open source software.

Debian is a distribution supported by an enthusiastic community of its own users and developers, based on the commitment to use free software.

The Debian project was founded in 1993 to create the Debian GNU/Linux distribution. Since then it has become fairly popular and even rivals other commercial distributions in terms of use, such as Red Hat or Mandrake. Because it is a community project, the development of this distribution is governed by a series of policies or rules; there are documents known as the Debian Social Contract, which mention the project's overall philosophy and Debian's policies, specifying in detail how to implement its distribution.

Note

We can see the Debian Social Contract documents at: debian.org.

The Debian distribution is closely related to the objectives of the FSF and its GNU Free Software project; for this reason, they always include "Debian GNU/Linux" in their name; also, the text of their social contract has served as the basis for open source definitions. Where their policies are concerned, anyone who wishes to participate in the distribution project, must abide by them. Although not a collaborator, these policies can be interesting because they explain how the Debian distribution operates.

Figure 2

We should also mention a practical aspect where end users are concerned: Debian has always been a difficult distribution. It tends to be the distribution used by Linux hackers, meaning those that gut the kernel and make changes, low level programmers, who wish to be on the leading edge to test new software, and to test unpublished kernel developments... in other words, all manner of folk who are mad about GNU/Linux.

Earlier versions of Debian became famous for the difficulty of installing them. The truth is that not enough effort had been made to make it easy for non-experts. But with time things have improved. Now, the installation still re-

quires a certain amount of knowledge, but can be done following menus (text menus, unlike other commercial versions that are totally graphic), and there are programs to facilitate package installations. But even so, the first attempts can be somewhat traumatic.

Normally, they tend to be variants (called flavours) of the Debian distribution. Currently, there are three branches of the distribution: *stable, testing* and *unstable.* And, as their names indicate, *stable* is the one used for production environments (or users who want stability), *testing* offers newer software that has been tested minimally (we could say it is a sort of beta version of Debian) that will soon be included in the *stable* branch. And the *unstable* branch offers the latest novelties in software, and its packages change over a short time period; within a week, or even every day, several packages can change. All distributions are updatable from various sources (CD, FTP, web) or by a system known as APT which manages Debian DEB software packages. The three distributions have more common names assigned to them e.g. (in a Debian specific line of time):

- Etch (*stable*)
- Lenny (*testing*)
- Sid (*unstable*)

The previous *stable* version was called Sarge (3.1r6), formerly Woody (that was 3.0). The most current one (in 2007), is the Debian GNU/Linux Etch (4.0). The most extended versions are Etch and Sid, which are the two extremes. At this time, Sid is not recommended for daily working environments (production), because it may have features that are halfway through testing and can fail (although this is uncommon); it is the distribution that GNU/Linux hackers tend to use. Also, this version changes almost daily; it is normal, if a daily update is wanted, for there to be between 10 and 20 new software packages per day (or even more at certain points in the development).

Etch is perhaps the best choice for daily working environments, it is updated periodically in order to cover new software or updates (such as security updates). Normally, it does not have the latest software which is not included until the community has tested it with an extensive range of tests.

We will comment briefly on some of this distribution's characteristics (current default versions of Etch and Sid):

a) The current (stable) version consists of between 1 and 21 CDs (or 3 DVDs) of the latest available version of Etch. Normally there are different possibilities depending on the set of software that we find on physical support (CD or DVD) or what we can subsequently download from the Internet, for which we only need a basic CD (netinstall CD), plus the internet access to download the rest upon demand. This distribution can be bought (at a

symbolic cost for the physical support, thus contributing to maintain the distribution) or can be downloaded from debian.org or its mirrors.

b) The testing and unstable versions tend not to have official CDs, but rather a *stable* Debian can be converted into a *testing* or *unstable* version by changing the configuration of the APT packages system.

c) Linux kernel: the default kernels were 2.4.x series and included an optional 2.6.x, which is now the default in the latest versions. The focus of the *stable* Debian is to promote stability and to leave users the option of another more updated software product if they need it (in *unstable* or *testing*).

d) Packaging format: Debian supports one of the formats that offers most facilities, APT. The software packages have a format known as DEB. APT is a high level tool for managing them and maintaining a database of instantly installable or available ones. Also, the APT system can obtain software from various sources, CD, FTP, or web.

e) The APT system is updatable at any time, from a list of Debian software sources (APT sources), which may be default Debian (debian.org) or third party sites. This way we are not linked to a single company or to a single subscription payment system.

f) Some of the versions used are, for example: Xfree86(4.x), *glibc* (2.3.x)... Debian Sid has Xorg (7.1), *glibc* (2.3.x)...

g) For the desktop, it accepts Gnome 2.16.x (default) or KDE 3.3.x (K Desktop Environment). Unstable with Gnome 2.18.x and KDE 3.5.x.

h) In terms of interesting applications, it includes the majority of those we tend to find in GNU/Linux distributions; in Sid: editors such as *emacs* (and *xemacs*), *gcc* compiler and tools, Apache web server, Mozilla (or Firefox) web browser, Samba software for sharing files with Windows etc.

i) It also includes office suites such as OpenOffice and KOffice.

j) Debian includes many personalised configuration files for distribution in /etc directories.

k) Debian uses the *lilo,* boot manager by default, although it can also use *Grub.*

l) The configuration for listening to TCP/IP network services, which is done, as on most UNIX systems, with the inetd server (/etc/inetd.conf). Although it also has an optional xinetd, which is becoming the preferred choice.

m) There are many more GNU/Linux distributions based on Debian, since the system can be easily adapted to make bigger or smaller distributions with more or less software adapted to a particular segment. One of the most famous ones is Knoppix, a single CD distribution, of the Live CD type (run on CD), which is commonly used for GNU/Linux demos, or to test it on a machine without previously installing it, since it runs from the CD, although it can also be installed on the hard disk and become a standard Debian. Linex is another distribution that has become quite famous because of its development supported by the local authority of the autonomous community of Extremadura. At the same time, we find Ubuntu, one of the distributions to have achieved the greatest impact (even exceeding Debian in several aspects), because of its ease for building an alternative desktop.

Note

Debian can be used as a base for other distributions; for example, Knoppix is a distribution based on Debian that can be run from CD without having to install it on the hard drive. Linex is a Debian distribution adapted to the autonomous community of Extremadura as part of its project to adopt open source software. And Ubuntu is a distribution optimised for desktop environments.

Figure 3. Debian Sid environment with Gnome 2.14

6.2. Fedora Core

Red Hat Inc. [Redh] is one of the main commercial companies in the world of GNU/Linux, with one of the most successful distributions. Bob Young and Marc Ewing created Red Hat Inc. in 1994. They were interested in open source software models and thought it would be a good way of doing business. Their main product is their Red Hat Linux distribution (which we will abbreviate

to Red Hat), which is available to different segments of the market, individual users (personal and professional versions), or medium or large companies (with their Enterprise version and its different sub-versions).

Red Hat Linux is the main commercial distribution of Linux, oriented at both the personal desktop and high range server markets. Additionally, Red Hat Inc. is one of the companies that collaborates the most in the development of Linux, since various important members of the community work for it.

Figure 4

Although they work with an open source model, it is a company with commercial objectives, which is why they tend to add value to their basic distribution through support contracts, update subscriptions and other means. For businesses, they add tailor-made software (or own software), to adapt it to the company's needs, either through optimised servers or utility software owned by Red Hat.

As of a certain point (towards the end of 2003), Red Hat Linux (version 9.x), decided to discontinue its desktop version of GNU/Linux, and advised its clients to migrate towards the company's business versions, which will continue to be the only officially supported versions.

Note

See: http://fedoraproject.org

At that moment, Red Hat decided to initiate the project open to the community known as Fedora [Fed], with a view to producing a distribution guided by the community (Debian-style, although for different purposes), to be called Fedora Core. In fact, the goal is to create a development laboratory open to the community that makes it possible to test the distribution and at the same time to guide the company's commercial developments in its business distributions.

To some extent, critics have pointed out that the community is being used as betatesters for technologies that will subsequently be included in commercial products. Also, this model is subsequently used by other companies to create

in turn dual models of community and commercial distributions. Examples such as OpenSuse appear (based on the commercial SuSe), or Freespire (based on Linspire).

Normally, the duo of Red Hat and the Fedora community present a certain conservative vision (less accentuated at Fedora) of the software elements it adds to the distribution, since its main market is businesses, and it tries to make its distribution as stable as possible, even if it means not having the latest versions. What it does do as an added value is to extensively debug the Linux kernel with its distribution and to generate corrections and patches to improve its stability. Sometimes, it can even disable a functionality (or driver) of the kernel, if it considers that it is not stable enough. It also offers many utilities in the graphics environment and its own graphics programs, including a couple of administration tools; in terms of graphics environments, it uses both Gnome (by default) and KDE, but through its own modified environment called BlueCurve, which makes the two desktops practically identical (windows, menus etc.).

The version that we will use will be the latest available Fedora Core, which we will simply call Fedora. In general, the developments and features that are maintained tend to be fairly similar in the versions released later, meaning that most comments will be applicable to the different versions over time. We should take into account that the Fedora [Fed] community tries to meet a calendar of approximately 6 months for each new version. And there is a certain consensus over what new features to include.

Red Hat, on the other hand, leaves its desktop versions in the hands of the community and focuses its activity on the business versions (Red Hat Linux Enterprise WS, ES, and AS).

Let's look briefly at a few characteristics of this Fedora Core distribution:

a) The current distribution consists of 5 CDs, the first one being the bootable one, which serves for the installation. There are also extra CDs containing documentation and the source code of most of the software installed with the distribution. The distribution is also provided on 1 DVD.

b) Linux kernel: it uses kernels of the 2.6.x series, which can be updated with the rpm packages system (see unit on the kernel) (through the yum utility for example). Red Hat, for its part, subjects the kernel to many tests and creates patches for solving problems, which are normally also incorporated into the version of the Linux community, since many important Linux collaborators also work for Red Hat.

c) Packaging format: Red Hat distributes its software through the RPM packages system (*red hat package manager*), which are managed by the *rpm* command or the yum utilities (we will comment on this in the unit on local

administration). RPM is one of the best available packaging systems (similar to Debian's deb), and some proprietary UNIX systems are including it. Basically, the RPM system maintains a small database with the installed packages and verifies that the package to be installed with the *rpm* command is not already installed or does not enter into conflict with any other software package, or on the other hand that a software package or the version required by the installation is not missing. The RPM package is basically a set of compressed files containing information on dependencies or on the software that it requires.

d) Regarding start up, it uses scripts of the *System V* type (which we will look at in the unit on local administration).

e) Some of the versions used are: Xorg (7.x), glibc (2.5.x) etc.

f) The desktop accepts Gnome (default desktop) and KDE as an option.

g) Where interesting applications are concerned, it includes most of the ones we tend to find with almost all GNU/Linux distributions: editors such as *emacs* (and *xemacs*), *gcc* compiler and tools, Apache web server, Firefox/Mozilla web browser, Samba software for sharing files with Windows etc.

h) It also includes office suites such as OpenOffice and KOffice.

i) Additional software can be obtained through the yum update services (among others) in a similar way to the Debian APT system or using different update tools, or from the Internet using RPM packages designed for the distribution.

j) Fedora uses the Grub boot loader by default to start up the machine.

k) Red Hat has replaced the configuration for listening to the TCP/IP network services, which for most UNIX systems uses the inetd server (/etc/inetd.conf), with xinetd, which has a more modular configuration (directory/etc/xinetd.d).

l) Upon start up it has a program called Kudzu which verifies any changes in hardware and detects newly installed hardware. We expect that it will be left out of following versions, because there is now a new API called HAL, which performs this function.

m) There are several more distributions based on the original Red Hat, which retain many of its characteristics, in particular Mandriva (formerly Mandrake): a French distribution, that was originally based on Red Hat and that together with Red Hat remains among the leaders in terms of user preferences (especially for desktop work). Mandriva develops its own software and lots of wizards to help with the installation and administration

of the most common tasks, separating itself from its origin based on Red Hat. At the same time, Red Hat business versions have also given rise to a series of very popular free distributions in server environments, such as CentOS [Cen] (which tries to maintain 100% compatibility with the business Red Hat), and Scientific Linux [Sci] (specialised in scientific computing for scientific research projects). As for the packaging system, it is worth noting that the rpm system is used for a large number of distributions, including SuSe.

Figure 5. Fedora Core desktop with Gnome

Regarding the community distribution Fedora Core, and its commercial origins in Red Hat:

a) It is a distribution created by a community of programmers and users based on development; it does not have any support for updates or maintenance on the part of the manufacturer. This aspect comes to depend on the community, as in the case of the Debian GNU/Linux distribution.

Note

See Fedora Release Notes at: http://docs.fedoraproject.org/

b) These versions are produced fairly rapidly, and new versions of the distribution are expected approximately every six months.

c) It also uses the RPM package management system. In terms of the process of updating the distribution's packages or installing other new ones, it can be achieved by means of different tools, via update, through the Fedora update channels or the new Yum update systems and in some cases Apt (inherited from Debian, but that works with RPM files).

d) Other more technical aspects (some of which we will look at in later chapters) can be found in the Fedora Core version notes.

7. What we will look at...

Having studied this "philosophical" introduction to the world of open source and the history of UNIX and GNU/Linux systems, as well as defining the tasks of a system administrator, we will look at how to handle the typical tasks involved in administrating GNU /Linux systems.

Next, we will look at the different areas involved in administering GNU/Linux systems. For each area, we will try to examine a few basic theoretical foundations that will help us to explain the tasks that need to be done and to understand how the tools that we will use work. Each subject will be accompanied by a type of tutorial where we will look at a small work session or how some tools are used. We will simply remember that, as mentioned in the introduction, the field of administration is very broad and any attempt at covering it completely (like this one) is destined to fail because of its limited size; therefore, you will find an abundant bibliography for each subject (in the form of books, web pages, web sites, howtos etc.), where you can broaden your knowledge from the brief introduction we have made on the subject.

The subjects we will look at are as follows:

- Under the section on migration, we will gain a perspective of the type of computer systems that are being used and in what work environments; we will also look at how GNU/Linux systems adapt better or worse to each one of them and will consider a first dilemma when it comes to introducing a GNU/Linux system: do we change the system we had or do we do it in stages with both coexisting?

- Under the section on tools we will study (basically) the set of tools that the administrator will have to live with (and/or suffer with) on a daily basis, and that could comprise the administrator's toolbox. We will talk about the GNU/Linux standards, which will allow us to learn about common aspects of all GNU/Linux distributions, in other words, what we can expect to find in any system. Other basic tools will be: simple (or not so simple) editors; some basic commands for learning about the system's status or for obtaining filtered information depending on what we are interested in; programming command scripts (or shell scripts) that will allow us to automate tasks; characteristics of the languages we may find in the administration tools or applications; basic program compilation processes based on source codes; tools for managing the installed software, as well as commenting on the dilemma over using graphics tools or command lines.

- Under the section concerning the kernel, we will observe the Linux kernel and how, by tailoring it, we can adjust it better to the hardware or to the services that we wish to provide from our system.

- Under the local administration heading, we will deal with those aspects of the administration that we could consider "local" to our system. These aspects may comprise most of the administrator's typical tasks when it comes to handling elements such as users, printers, disks, software, processes etc.

- In the section on the network, we will examine all the administration tasks that concern our system and its neighbourhood in the network, irrespective of its type, and we will look at the different types of connectivity that we can have with neighbouring systems or the services that we can offer or receive from them.

- In the section on servers, we will look at a few typical configurations of servers that we can commonly find on a GNU/Linux system.

- In the section on data, we will look at one of today's most relevant themes, the data storage and consultation mechanisms that GNU/Linux systems can offer us, in particular, database systems and version control mechanisms.

- In the section on security, we will handle one of today's most relevant and important issues regarding the whole GNU/Linux system. The existence of a world interconnected by the Internet entails a series of important dangers for our systems' correct functioning and gives rise to the issue of reliability, both of these systems and of the data that we may receive or offer through the net. Therefore, our systems need to provide minimum levels of security and to prevent unauthorised access to or handling of our data. We will look at the most frequent types of attacks, security policies that can be enforced and the tools that can help us to control our security level.

- In the section on optimisation, we will see how, because of the large number of servers and services on offer, as well as the large number of environments for which the system is designed, GNU/Linux systems tend to have many functioning parameters that influence the performance of the applications or services on offer. We can (or should) try to extract maximum performance by analysing the system's own configurations to adjust them to the quality of service that we wish to offer clients.

- In the section on clustering, we will look at some of the techniques for providing high performance computing on GNU/Linux systems, extensively used in the fields of scientific computing and becoming more frequently used by a large number of industries (pharmaceuticals, chemistry,

materials etc.), for researching and developing new products. In addition to the organisation of various GNU/Linux systems into clusters, to amplify the performance of individual systems, by creating groups of systems that make it possible to scale the services offered to an increased client demand.

Activities

1) Read the Debian manifesto at:

http://www.debian.org/social_contract

2) Read up on the different distributions based on Debian: Knoppix, Linex, Ubuntu variants. Apart from each distribution's website, the address www.distrowatch.com offers a good guide to the distributions and their status, as well as the software that they include. Through this webpage or by accessing the different communities or manufacturers we can obtain the ISO images of the different distributions.

Bibliography

Other sources of reference and information (see references under Bibliography)

[LPD] The Linux Documentation Project (LDP), collection of Howtos, manuals and guides covering any aspect of GNU/Linux.

[OSDb] Community with various websites, news, developments, projects etc.

[Sla] Open Source community news site and general sites on IT and the Internet.

[New] [Bar] Open Source News.

[Fre] [Sou] List of Open Source projects.

[Dis] Monitoring of GNU/Linux distributions and new features of the software packages. And links to the sites for downloading the ISO images of the GNU/Linux distribution CDs/DVDs.

[His] [Bul] [LPD] General documentation and communities of users.

[Mag03] [Jou03] GNU/Linux magazines.