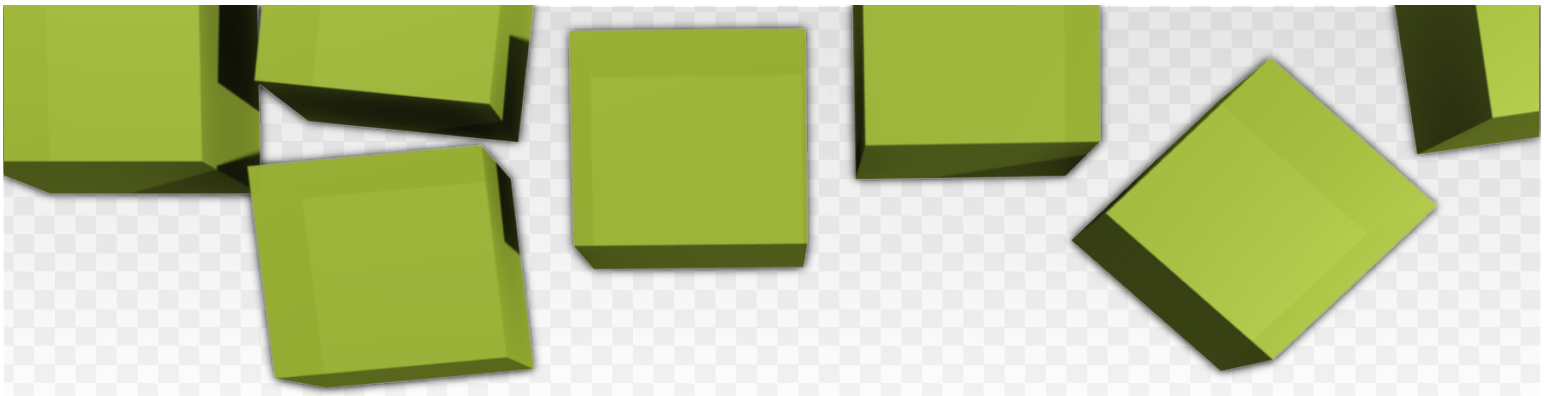

GNU/Linux Advanced Administration

AUTHOR:
A. Suppi Boldrito

COORDINATOR:
J. Jorba Esteve

Free Technology Academy

2

GNU/Linux advanced administration

Josep Jorba Esteve (coordinador)
Remo Suppi Boldrito

XP07/M2103/02279

FREE TECHNOLOGY ACADEMY

Josep Jorba Esteve

Senior engineer and PhD in IT of the Universidad Autónoma de Barcelona (UAB). Professor of IT, Multimedia and Telecommunications Studies of the Open University of Catalonia (UOC).

Remo Suppi Boldrito

Telecommunications Engineer. PhD in IT of the UAB. Professor of the Department of Computer Architecture and Operating Systems of the UAB.

First edition: September 2007

Av. Tibidabo, 39-43, 08035 Barcelona
Design: Manel Andreu
Publishing: Eureca Media, SL

Preface

Software has become a strategic societal resource in the last few decades. The emergence of Free Software, which has entered in major sectors of the ICT market, is drastically changing the economics of software development and usage. Free Software – sometimes also referred to as "Open Source" or "Libre Software" – can be used, studied, copied, modified and distributed freely. It offers the freedom to learn and to teach without engaging in dependencies on any single technology provider. These freedoms are considered a fundamental precondition for sustainable development and an inclusive information society.

Although there is a growing interest in free technologies (Free Software and Open Standards), still a limited number of people have sufficient knowledge and expertise in these fields. The FTA attempts to respond to this demand.

Introduction to the FTA
The Free Technology Academy (FTA) is a joint initiative from several educational institutes in various countries. It aims to contribute to a society that permits all users to study, participate and build upon existing knowledge without restrictions.

What does the FTA offer?
The Academy offers an online master level programme with course modules about Free Technologies. Learners can choose to enrol in an individual course or register for the whole programme. Tuition takes place online in the FTA virtual campus and is performed by teaching staff from the partner universities. Credits obtained in the FTA programme are recognised by these universities.

Who is behind the FTA?
The FTA was initiated in 2008 supported by the Life Long Learning Programme (LLP) of the European Commission, under the coordination of the Free Knowledge Institute and in partnership with three european universities: Open Universiteit Nederland (The Netherlands), Universitat Oberta de Catalunya (Spain) and University of Agder (Norway).

For who is the FTA?
The Free Technology Academy is specially oriented to IT professionals, educators, students and decision makers.

What about the licensing?
All learning materials used in and developed by the FTA are Open Educational Resources, published under copyleft free licenses that allow them to be freely used, modified and redistributed. Similarly, the software used in the FTA virtual campus is Free Software and is built upon an Open Standards framework.

Evolution of this book

The FTA has reused existing course materials from the Universitat Oberta de Catalunya and that had been developed together with LibreSoft staff from the Universidad Rey Juan Carlos. In 2008 this book was translated into English with the help of the SELF (Science, Education and Learning in Freedom) Project, supported by the European Commission's Sixth Framework Programme. In 2009, this material has been improved by the Free Technology Academy. Additionally the FTA has developed a study guide and learning activities which are available for learners enrolled in the FTA Campus.

Participation

Users of FTA learning materials are encouraged to provide feedback and make suggestions for improvement. A specific space for this feedback is set up on the FTA website. These inputs will be taken into account for next versions. Moreover, the FTA welcomes anyone to use and distribute this material as well as to make new versions and translations.

See for specific and updated information about the book, including translations and other formats: *http://ftacademy.org/materials/fsm/2*. For more information and enrolment in the FTA online course programme, please visit the Academy's website: *http://ftacademy.org/*.

I sincerely hope this course book helps you in your personal learning process and helps you to help others in theirs. I look forward to see you in the free knowledge and free technology movements!

Happy learning!

Wouter Tebbens

President of the Free Knowledge Institute

Director of the Free technology Academy

The authors would like to thank the Foundation for the Universitat Oberta de Catalunya for financing the first edition of this work, and a large share of the improvements leading to the the second edition, as part of the Master Programme in Free Software offered by the University in question, where it is used as material for one of the subjects.

The translation of this work into English has been made possible with the support from the SELF Project, the SELF Platform, the European Comission's programme on Information Society Technologies and the Universitat Oberta de Catalunya. We would like to thank the translation of the materials into English carried out by lexia:park.

The current version of these materials in English has been extended with the funding of the Free Technology Academy (FTA) project. The FTA project has been funded with support from the European Commission (reference no. 142706-LLP-1-2008-1-NL-ERASMUS-EVC of the Lifelong Learning Programme). This publication reflects the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

Contents

Module 5

Local administration

Josep Jorba Esteve

1. Distributions: special features
2. Running levels and services
3. Monitoring system state
4. File Systems
5. Users and groups
6. Printing services
7. Disks and file systems management
8. Updating Software
9. Batch jobs
10. Tutorial: combined practices of the different sections

Module 6

Network administration

Remo Suppi Boldrito

1. Introduction to TCP/IP (TCP/IP suite)
2. TCP/IP Concepts
3. How to assign an Internet address
4. How to configure the network
5. DHCP Configuration
6. IP aliasing
7. IP Masquerade
8. NAT with kernel 2.2 or higher
9. How to configure a DialUP and PPP connection
10. Configuring the network through *hotplug*
11. Virtual private network (VPN)
12. Advanced configurations and tools

Module 7

Server administration

Remo Suppi Boldrito

1. Domain name system (DNS)
2. NIS (YP)
3. Remote connection services: telnet and ssh
4. File transfer services: FTP
5. Information exchange services at user level
6. Proxy Service: Squid
7. OpenLdap (Ldap)
8. File services (NFS)

Module 8

Data administration

Remo Suppi Boldrito

1. PostgreSQL
2. Mysql
3. Source Code management systems

4. Subversion

Module 9

Security administration

Josep Jorba Esteve

1. Types and methods of attack
2. System security
3. Local security
4. SELinux
5. Network security
6. Intrusion detection
7. Filter protection through wrappers and firewalls
8. Security tools
9. Logs analysis
10. Tutorial: How to use security analysis tools

Module 10

Configuration, tuning and optimisation

Remo Suppi Boldrito

1. Basic aspects

Module 11

Clustering

Remo Suppi Boldrito

1. Introduction to High Performance Computing (HPC)
2. OpenMosix
3. Metacomputers, grid computing

Introduction to the GNU/Linux operating system

Josep Jorba Esteve

PID_00148470

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.

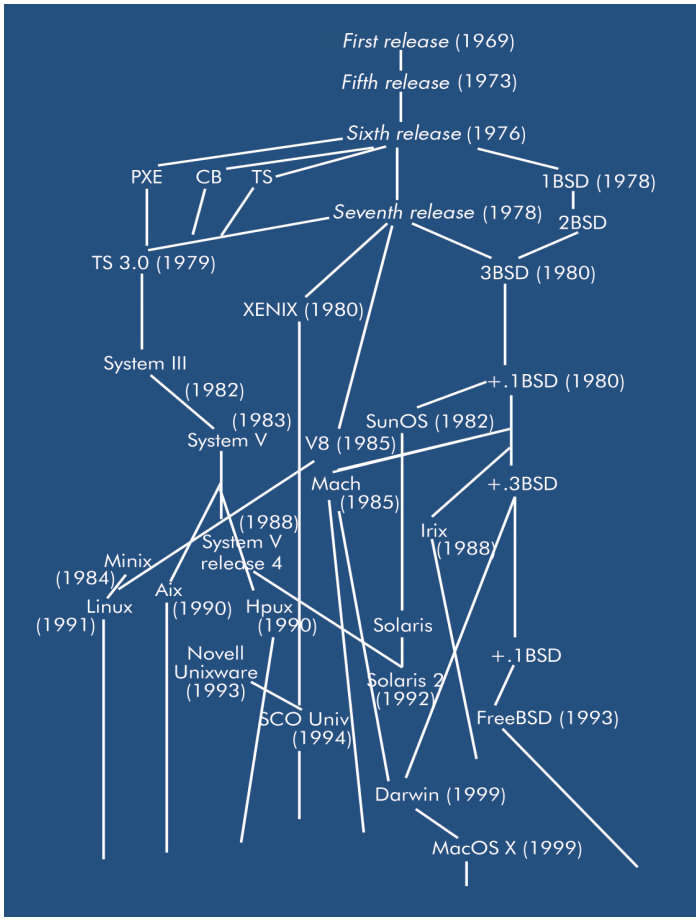

Figure 1. Historical summary of the different versions of UNIX

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.

Migration and coexistence with non-Linux systems

Josep Jorba Esteve

PID_00148467

Universitat Oberta de Catalunya
www.uoc.edu

Index

Introduction

Having had a brief introduction to GNU/Linux systems, the following step is to integrate them in the work environment as production systems. According to the current system in use, we can consider either a full migration to GNU/Linux systems or a coexistence through compatible services.

Migration to the GNU/Linux environment may be done progressively by replacing services partially or by substituting everything in the old system by GNU/Linux equivalents.

In current distributed environments, the most relevant concern is the client/server environments. Any task in the global system is managed by one or more dedicated servers, with the applications or users directly accessing the offered services.

Regarding the work environment, whether in the simplest case of the individual user or the more complex case of a business environment, every environment will require a set of services that we will need to select, later adjusting client and server machines so that they can access them or provide their use.

The services may encompass different aspects and there tend to be various types for sharing resources or information. File servers, print servers, web servers, name servers, e-mail servers etc., are common.

The administrator will normally select a set of services that need to be present in the work environment according to the needs of the end users and/or the organisation; and must configure the right support for the infrastructure, in the form of servers that support the expected workload.

1. Computer systems: environments

During the process of installing some GNU/Linux distributions, we often find that we are asked about the type of environment or tasks our system will be dedicated to, which often allows us to choose a sub-set of software that will be installed for us by default, because it is the most suited to the contemplated job. We will often be asked if the system will be used as a:

a) Workstation: this type of system usually incorporates particular applications that will be used most frequently. The system is basically dedicated to running these applications and a small set of network services.

b) Server: basically it integrates most network services or, in any case, a particular service, which will be the system's main service.

c) Dedicated calculation unit: calculation-intensive applications, renders, scientific applications, CAD graphics etc.

d) Graphics station: desktop with applications that require interaction with the user in graphic form.

We can normally set up our GNU/Linux system with one or more of these possibilities.

More generally, if we had to separate the work environments [Mor03] where a GNU/Linux system can be used, we could identify three main types of environments: workstation, server and desktop .

We could also include another type of systems, which we will generically call embedded devices or small mobile systems like a PDA, mobile telephone, portable video console etc. GNU/Linux also offers support for these devices, with smaller personalised kernels for them.

Note

GNU/Linux systems can be dedicated to server, workstation or desktop functions.

Example

For example, we should mention the initial work done by the Sharp company on its Zaurus models, a PDA with advanced Linux features (there are four or five models on the market). Or also other Linux initiatives of an embedded type such as POS (point of sale) terminals. Or video consoles such as GP2X, and Sony Playstation 3 linux support. Also new smartphone/PDA platforms like Google Android, Nokia Maemo, Intel Moblin.

Regarding the three main environments, let's look at how each one of these computer systems is developed in a GNU/Linux environment:

1) A workstation type system tends to be a high performance machine used for a specific task instead of a general set of tasks. The workstation classically consisted of a high performance machine with specific hardware suited to the task that needed doing; it was usually a Sun's SPARC, IBM's RISC or Silicon Graphics machine (among others) with its variants of proprietary UNIX. These high cost machines were oriented at a clear segment of applications, whether 3D graphic design (in the case of Silicon or Sun) or databases (IBM or Sun). Nowadays, the performance of many current PCs is comparable (although not equal) to these systems and the frontier between one of these systems and a PC is no longer clear, thanks to the existence of GNU/Linux as an alternative to the proprietary UNIX versions.

2) A server type system has a specific purpose, which is to offer services to other machines on the network: it offers a clearly distinct set of characteristics or functionality from other machines. In small computer systems (for example, with less than 10 machines), there is not usually an exclusive server system, and it tends to be shared with other functionalities, for example as a desktop type machine. Medium systems (a few dozen machines) tend to have one or more machines dedicated to acting as a server, whether as an exclusive machine that centralises all services (e-mail, web etc.) or as a pair of machines dedicated to sharing the main services.

In large systems (hundreds or even thousands of machines), the load makes it necessary to have a large group of servers, with each one usually exclusively dedicated to a particular service, or even with a set of machines exclusively dedicated to one service. Moreover, if these services are provided inwards or outwards of the organisation, through access by direct clients or open to the Internet, depending on the workload to be supported, we will have to resort to SMP multicore type solutions (machines with multiple processors/code) or of the cluster type (grouping of machines that distribute a particular service's load).

The services that we may need internally (or externally) can encompass (among others) the following service categories:

a) Applications: the server can run applications and as clients we just observe their execution and interact with them. For example, it may encompass terminals services and web-run applications.

b) Files: we are offered a shared and accessible space from any point of the network where we can store/recover our files.

c) Database: centralisation of data for consultation or production by the system's applications on the network (or for other services).

d) Printing: there are sets of printers and their queues and jobs sent to them from any point of the network are managed.

e) E-mail: offers services for receiving, sending or resending incoming or outgoing mail.

f) Web: server (or servers) belonging to the organisation for internal or external use by customers.

g) Network information: for large organisations it is vital for finding the services offered or the shared resources; or users themselves, if they need services that make this localisation possible and to consult the properties of each type of object.

h) Names services: services are required to name and translate the different names by which the same resource is known.

i) Remote access services: in the case of not having direct access, we need alternative methods that allow us to interact from the outside, giving us access to the system that we want.

j) Name generation services: in naming machines, for example, there may be a highly variable number of them, or they may not always be the same ones. We need to provide methods for clearly identifying them.

k) Internet access services: many organisations have no reasons for direct access and rather have access through gateways or proxies.

l) Filtering services: security measures for filtering incorrect information or information that affects our security.

3) A desktop type machine would simply be a machine used for routine everyday computer tasks (such as our home or office PC).

Example

For example, we could establish the following as common tasks (included in some of the most used GNU/Linux programs):

- Office tasks: providing the classical software of an office suite: word processor, spreadsheet, presentations, a small database etc. We can find suites like OpenOffice (free), StarOffice (paid for, produced by Sun), KOffice (by KDE), or various programs like Gnumeric, AbiWord which would form part of a suite for Gnome (known as GnomeOffice).

- Web browser: browsers such as Mozilla Firefox, Konqueror, Epiphany etc.

- Hardware support (USB, storage devices...). Supported in GNU/Linux by the appropriate drivers, usually provided in the kernel or by the manufacturers. There are also new hardware analysis tools such as kudzu (Fedora/Red Hat) or discover (Debian). Media and entertainment (graphics, image processing, digital photography, games and more). In GNU/Linux there is an enormous amount of these applications of a very professional quality: Gimp (touching up photographs), Sodipodi, Xine, Mplayer, gphoto etc.

- Connectivity (remote desktop access, access to other systems). In this regard, GNU/Linux has an enormous amount of own tools whether TCP/IP or FTP, telnet, web etc., or X Window, which has remote desktop capabilities for any UNIX machine, rdesktop (for connecting to Windows desktops), or VNC (for connecting to UNIX, Windows, Mac etc.).

Web sites

Open Source office suites:

http://openoffice.org
http://www.koffice.org/
http://live.gnome.org/GnomeOffice

2. GNU/Linux services

GNU/Linux has servers adapted for any work environment.

The service categories we have mentioned have equivalents that we can provide from our GNU/Linux systems to all other machines on the network (and from which they can also act as clients):

a) Applications: GNU/Linux can provide remote terminal services, whether by direct connection through series interfaces of dumb terminals, serving to visualise or interact with the applications. Another possibility is remote connection in text mode, from another machine via TCP/IP services such as rlogin, telnet, or in a secure way with ssh. GNU/Linux provides servers for all these protocols. In the case of running graphics applications, we have remote solutions through X Window, any UNIX, Linux or Windows client (or others) with an X Window client can visualise the running of the environment and its applications. At the same time, there are other solutions such as VNC for the same problem. Regarding the issue of web-run applications, GNU/Linux has the Apache server, and any of the multiple web running systems are available, whether Servlets (with Tomcat), JSP, Perl, PHP, xml, webservices etc., as well as web application servers such as BEA Weblogic, IBM Websphere, JBoss (free) which are also run on GNU/Linux platforms.

b) Files: files can be served in various ways, either through FTP access to the files, or by serving them in a transparent manner to UNIX and Linux machines with NFS, or by acting as client or server towards Windows machines through Samba.

c) Database: it supports a large number of relational client/server type databases such as MySQL, PostgreSQL and several commercial ones such as Oracle or IBM DB2, among others.

d) Printing: it can serve local or remote printers, for both UNIX systems with TCP/IP protocols and Windows through Samba/CIFS.

e) E-mail: it offers services for clients to obtain mail on their machines (POP3 or IMAP servers), as mail transfer agents (MTA) to recover and retransmit mail, such as the Sendmail server (UNIX standard) or others like Exim and, in the case of outward sending, the SMTP service for outgoing mail.

f) Web: we have the http Apache server, whether in its 1.3.x versions or the new 2.0.x. or 2.2.x. versions Also, we can integrate web application servers, such as Tomcat for servlets, JSP...

g) Network information: services such as NIS, NIS+ or LDAP allow us to centralise the information from the machines, users, and various resources on our network, facilitating administration and service to users, in such a way that the latter do not depend on their situation in the network. Or if our organisation has a certain internal structure, these services will allow us to model it allowing access to the resources to whoever needs it.

h) Names services: services such as DNS for machine names and their translation from or to IP, by means of the Bind server for example (the standard UNIX DNS).

i) Remote access services: whether to run applications or to obtain remote information on the machines. The servers could be the ones we have mentioned for the applications: X Window, VNC etc., and also those that allow some remote commands to be run without interactivity such as rexec, rsh, ssh etc.

j) Name generation services: services such as DHCP allow TCP/IP networks, to dynamically (or statically) generate the available IP addresses according to the machines that need it.

k) Internet access services: in certain situations there may be a single output to Internet (or several). These points tend to act as proxy, since they have access and they redirect it to potential Internet accesses on behalf of clients. They also tend to act as content cache. In GNU/Linux we can have Squid for example. In this category, a gateway or router could also come into action in a GNU/Linux system, whether to direct packages to other networks or to find alternative resending routes. Also, in the case of small installations such as domestic ones, we could include the Internet access by modem through the PPP services.

l) Filtering services: one of the most commonly used security measures at present is firewalls. They basically represent filtering techniques for incoming or outgoing packages, for the different protocols we are using, to put up barriers against unwanted ones. In GNU/Linux, we have mechanisms such as ipchains and iptables (more modern) for implementing firewalls.

3. Types of use

GNU/Linux, as a system, offers characteristics that are valid for personal users as well as users of a medium or large-scale infrastructure.

From the perspective of GNU/Linux system users, we could distinguish:

a) The individual or domestic user: normally, this type of user has one or several machines at home that may or may not be shared. In general, in this environment, GNU/Linux is used to develop a desktop system, which means that the graphics part will be important: the GNU/Linux desktop.
For this desktop we have two main options in the form of Gnome and KDE environments, both of which are perfectly valid. Either of the two environments offers applications running and visualisation services, together with a broad range of basic own applications that allow us to develop all sorts of routine tasks. The two environments offer a visual desktop with different menus, icon bars and icons, in addition to navigators for own files and various useful applications. Any environment can run its own applications and the others', although, in the same way as the applications, they run better in their own environment because their visual aspect is more suited to the environment for which they were designed.
Regarding applications for the personal user, we should include the typical ones of the desktop system. If the user has a home network, for example, a small group of computers joined by an Ethernet type network, services for sharing files and printers between machines could also be interesting. Services such as NFS may be necessary if there are other Linux machines; or Samba, if there are machines with Windows.
In the case of having an Internet connection through an ISP (Internet Service Provider) depending on the type of connection used, we would need to control the corresponding devices and protocols:

- Modem connection: telephone modems tend to use the PPP protocol to connect with the provider. We would have to enable this protocol and configure the accounts we have enabled with the provider. An important problem with Linux is the winModems issue, which has caused a lot of trouble. This modem (with some exceptions) is not supported, because it is not a real modem but rather a hardware simplification plus driver software, and most only function with Windows, meaning that we need to avoid them (if not supported) and to buy real (full) modems.

- ADSL modem connection: the functioning would be similar, the PPP protocol could be used or another one called EoPPP. This may depend

on the modem's manufacturer and on the type of modem: Ethernet or USB.

- ADSL connection with a router: the configuration is very simple, because in this situation all we need to do is to configure the Ethernet card and/or wireless card in our system to connect with the ADSL router.

Once the interface to Internet is connected and configured, the last point is to include the type of services that we will need. If we only want to act as clients on Internet, it will be sufficient to use the client tools of the different protocols, whether FTP, telnet, the web navigator, e-mail or news reader etc. If we also wish to offer outgoing services – for example, to publish a website (web server) or to allow our external access to the machine (ssh, telnet, FTP, X Window, VNC, services etc.), in this case, server – then we must remember that this will only be possible if our provider gives us fixed IP addresses for our machine. Otherwise, our IP address will change every time we connect and the possibility of offering a service will become either very difficult or impossible.

Another interesting service would be sharing access to the Internet between our available machines.

b) Mid-scale user: this is the user of a middle scale organisation, whether a small company or group of users. Normally, this type of users will have local network connectivity (through a LAN, for example) with some connected machines and printers. And will have direct access to Internet, either through some proxy (point or machine designed for an external connection), or there will be a few machines physically connected to the Internet. In general, in this environment, work is partly local and partly shared (whether resources, printers or applications). Normally, we will need desktop systems; for example, in an office we can use office suite applications together with Internet clients; and perhaps also workstation type systems; for example, for engineering or scientific jobs, CAD or image processing applications may be used, as well as intensive mathematical calculation systems etc., and almost certainly more powerful machines will be assigned to these tasks.

In this user environment, we will often have to share resources such as files, printers, possibly applications etc. Therefore, in a GNU/Linux system, NFS services will be appropriate, printer services, Samba (if there are Windows machines with which files or printers need to be shared), and we may also need database environments, an internal web server with shared applications etc.

c) Large-scale users: this type of user resembles the preceding one and differs only in the size of the organisation and available resources, which can be plenty, in such a way that some resources of the NIS, NIS+ or LDAP type network system directory may be needed in order to handle the organisation's information and reflect its structure, certainly also to

have large service infrastructures for external clients generally in the form of websites with various applications.

This type of organisation has high levels of heterogeneity in both system hardware and software, and we could find lots of architectures and different operating systems, meaning that the main tasks will consist of easing data compatibility by means of databases and standard document formats and to ease interconnectivity by means of standard protocols, clients and servers (usually with TCP/IP elements).

4. Migration or coexistence

Next, we will consider another important aspect in adopting GNU/Linux systems. Let's suppose that we are amateurs at handling this system; or, the opposite, that we are experienced and wish to adopt one or several GNU/Linux systems as individual users for working in our small organisation; or that we are considering replacing the infrastructure of our large company or organisation in full (or part).

Migrating to a new system is no trivial matter, it needs to be evaluated through a study that analyses both the costs and the beneficial features that we expect to obtain. Also, migration can be done in full or in part, with a certain degree of coexistence with former systems.

We will be dealing with a full or partial migration project of our IT systems to GNU/Linux and, as administrators, we will be responsible for this process.

As in any project, we will have to study the way of responding to questions such as: Does the change make sense in financial terms or in terms of performance benefits? What is the migration's objective? What requirements will we want to or need to fulfil? Can we do a partial migration or do we need to do a full migration? Is coexistence with other systems necessary? Will we need to retrain users? Will we be able to use the same hardware or will we need new hardware? Will there be important added costs? Or simply, will it go okay? These and many others are the questions that we will have to try and answer. In the case of a company, the answers would be provided in a migration project, specifying its objectives, requirements, the implementation process, and including a financial analysis, user training plans etc. We will not go into this in detail, but will consider some of these issues in a simple manner. And in the final workshop we will examine a few small cases of how we would implement the migration.

Also, the moment we start migrating to GNU/Linux, we will start to notice the advantages the system brings to our organisation:

a) Costs: reduction in license costs for the system's software and applications. GNU/Linux has 0 cost for licenses if purchased from the Internet (for example, in the form of images from the distribution's CDs), or a negligible cost if we take into account that the nearest comparison for systems with equivalent features would be Windows Server systems with license costs ranging between € 1,500 and € 3,000, without including a large amount of the additional software that a typical GNU/Linux distribution would include.

But careful, we should not underestimate maintenance and training costs. If our organisation consists solely of users and administrators trained in Windows, we may have high costs for retraining personnel and, possibly, for maintenance. Therefore, many big companies prefer to depend on a commercial distributor of GNU/Linux to implement and maintain the system, such as the business versions offered by Red Hat, SuSe and others. These GNU/Linux versions also have high license costs (comparable to Windows), but at the same time are already adapted to business structures and contain their own software for managing companies' IT infrastructure. Another important aspect, to conclude with cost estimates, is the TCO concept (total cost of ownership), as a global evaluation of the associated costs that we will find when we undertake a technological development; we don't just have to evaluate the costs of licenses and machines, but also the costs of training and support for the people and products involved, which may be as high or more than the implemented solution.

b) Support: GNU/Linux offers the best maintenance support that any operating system has ever had, and it is mostly free. Nevertheless, some companies are reluctant to adopt GNU/Linux on the basis that there is no product support and prefer to buy commercial distributions that come with support and maintenance contracts. GNU/Linux has a well-established support community worldwide, through various organisations that provide free documentation (the famous HOWTOs), specialised user forums, communities of users in practically any region or country in the world etc. Any question or problem we have can be searched on the Internet and we can find answers within minutes. If we don't, if we have found a bug, error, or untested situation, we can report it on various sites (forums, development sites, distribution bug sites etc.), and obtain solutions within hours or, at the most, within days. Whenever we have a question or problem, we should first try a few procedures (this is how we will learn) and if we do not find the solution within a reasonable amount of time, we should consult the GNU/Linux community in case any other user (or group of users) has encountered the same problem and found a solution, and if not, we can always post a report on the problem and see if we are offered solutions.

Note

Linux Howto's: http://www.tldp.org/

4.1. Identify service requirements

Normally, if we have systems that are already functioning we will have to have some services implemented for users or for helping the infrastructure of the IT support. The services will fall within some of the categories seen above, with the GNU/Linux options that we mentioned.

GNU/Linux systems are not at all new, and as we saw in the introduction, stem from a history of more than thirty years of UNIX systems use and development. Therefore, one of the first things that we will find is that we are not lacking support for any type of service we want. If anything, there will

be differences in the way of doing things. Also, many of the services used by IT systems were conceived, researched, developed and implemented in their day for UNIX, and only subsequently adapted to others systems (such as Windows, more or less successfully).

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

Any service available at the time may be adapted to GNU/Linux systems with equivalent (if not the same) services.

Example

A famous case is the one of the Samba servers [Woo00] [Sam]. Windows offers what it calls "sharing files and printers on the network" by means of its own protocols known generically as SMB (server message block) [Smb] (with network support in the NetBios and NetBEUI protocols). The name CIFS (common Internet file system) is also commonly used, which is what the protocol was called in a second revision (which continued to include SMB as a basic protocol). These protocols allowed the sharing of files (or disks) and printers on a network of Windows machines (in a workgroup configuration or in Windows domains). In UNIX this idea was already old when it appeared in Windows and services such as NFS for sharing files or managing printers remotely were already available using TCP/IP protocols.

One of the problems with replacing the Windows sharing services based on NetBios/NetBeui (and ultimately with NetBios over TCP/IP) was how to support these protocols, since if we wanted to keep the client machines with Windows, we could not use the UNIX services. For this purpose, Samba was developed as a UNIX server that supported Windows protocols and that could replace a Windows server/client machine transparently, with client users with Windows not having to notice anything at all. Moreover, the result in most cases was that the performance was comparable if not better than in the original machine with Windows services.

Currently, Samba [Sam] is constantly evolving to maintain compatibility with Windows file and printer sharing services; because of the general changes that Microsoft subjects SMB/CIFS [Smb] protocols to (the base implemented by Samba) with each new Windows version, in particular the evolution of workgroup schemes in the operating systems' client versions, to centralised server (or group of servers) schemes, with specific user authentication services (NTLM, NTLMv2, Kerberos), and centralised storage of the system's management such as Active Directory. In addition to this, the configuration of existing domain servers (whether with primary controller, backup or Active Directory).

Currently, in migration processes with Samba, we will need to observe what configurations of Windows clients/servers (and its versions) exist on the system, as well as what user authentication and/or information management systems are used. Also, we will need to know how the system is structured into domains (and its controller servers, members or isolated servers), in order to make a complete and correct mapping towards Samba-based solutions, and into complementary user authentication (winbind, kerberos, nss_ldap) and management services (for example openLDAP) [Sama] [Samb] .

4.2. Migration process

In the migration process, we need to consider how we want to migrate and if we want to migrate totally or partially, coexisting with other services or equipment that has a different operating system .

In the environments of large organisations, where we find a large number of heterogeneous systems, we will need to take into account that we will almost certainly not migrate every one of them, especially workstation type systems

that are dedicated to running a basic application for a specific task; it could be that there is no equivalent application or simply that we wish to keep these systems for financial reasons or in order to maximise an investment.

We can migrate various elements, whether the services we offer, the machines that offer the services or the clients who access the services.

Elements that can be migrated include:

a) Services or machines dedicated to one or more services. In migrating, we will replace the service with another equivalent one, normally with minimum possible impact unless we also wish to replace the clients. In the case of Windows clients, we can use the Samba server to replace the file and printer services offered by the Windows machines. For other services, we can replace them with GNU/Linux equivalents. In the case of replacing just one service, normally we will disable the service on the machine that offered it and enable it on the new system. Client changes may be necessary (for example, new machine addresses or parameters related to the service).

If a server machine was responsible for an entire function, we will need to analyse whether the machine was dedicated to one or more services and whether they can all be replaced. If so, we will just have to replace the old machine with the new one (or maintain the old one) with the services under GNU/Linux and in any case, modify a client parameter if necessary. Normally, before making a change, it is advisable to test the machine separately with a few clients in order to make sure that it performs the function correctly and then to replace the machines during a period when the system is inactive.

In any case, we will certainly have to back up data existing prior to the new system, for example, file systems or the applications available in the original server. Another point to consider in advance is data portability; a problem we often find is compatibility when the organisation used data or applications that depended on a platform.

Example

To mention a few practical cases that some companies find nowadays:

- Web applications with ASP: these applications can only be executed on web platforms with Windows and Microsoft's IIS web server. We should avoid them if we intend to migrate platforms at any time and don't wish to rewrite them or pay another company to do so. GNU/ Linux platforms have the Apache web server (the most commonly used on the Internet), which can also be used with Windows, this server supports ASP in Perl (in Windows it generally uses visual basic, C# and Javascript), there are third party solutions to migrate ASP or to more or less convert them. But if our company depended on this, it would be very costly in terms of time and money. A practical solution would have been to make the web developments in Java (which is portable between platforms) or other solutions such as PHP. On this point, we should highlight the Mono project [Mon] (sponsored by Novell) for portability of part of Microsoft's .NET environment to GNU/Linux, in particular a large amount of the.NET API's, C# language, and the ASP.NET specification. Allowing a flexible

migration of .NET applications based on .NET APIs that are supported by the Mono platform. At the same time, we should mention the FSF's DotGnu [Dgn] project, as a GPL alternative to Mono.

- Databases: using a Microsoft SQL Server for example, makes us totally dependant on its Windows platform, plus, if we use proprietary solutions in a specific environment for database applications, they will be difficult to transfer. Other databases such as Oracle and DB2 (IBM) are more portable because they have a version in the different platforms or because they use more portable programming languages. We could also work with PostgreSQL or MySQL database systems (it also has a version for Windows) available in GNU/Linux, and that allow an easier transition. At the same time, if we combine it with a web development we have a lot of possibilities; in this sense, nowadays we use systems such as: web applications with Java, whether servlets, applets, or EJB; or solutions such as the famous LAMP, the combination of GNU/Linux, Apache, Mysql and Php.

b) Workstation: in these migrations, the biggest problem stems from the applications, whether for CAD, animation, engineering or scientific programs, which are the workstation's main reason for being. Here it will be important to be able to replace them with equal or at least compatible applications with the same expected features or functionality. Normally, most of these applications stem from a UNIX world, given that most of these workstations were conceived as UNIX machines. Meaning that a compilation or minimum adaptation to the new GNU/Linux may be enough, if we have source code (as tends to be the case with many scientific applications). If we are dealing with commercial applications, the manufacturers (of engineering and scientific software) are starting to adapt them to GNU/Linux, although in these cases the applications are usually very expensive (easily hundreds to thousands of euros).

Note

For examples of GNU/Linux equivalent applications, see:
http://www.linuxalt.com/
http://wiki.linuxquestions.org/wiki/Linux_software_equivalent_to_Windows_software
http://www.linuxrsp.ru/win-lin-soft/table-eng.htmlç

c) Desktop client machines. Desktop machines continue to be a headache for the world of GNU/Linux, because they involve a number of additional problems. In servers, the machines are assigned clear functionalities, as a rule they do not require complex graphic interfaces (often text communication is sufficient), and the normally specific high performance hardware is purchased for a specific set of functions and the applications tend to be the servers themselves included in the operating system or some third party applications. Also, these machines are often managed by administrators with extensive knowledge of what they are dealing with. However, in the case of desktops, we are dealing with a problem factor (in itself and more so for administrators): the system's end users. The users of desktop systems expect to have powerful graphic interfaces that are more or less intuitive and applications that allow them to run routine – usually office – tasks. This type of user (with a few exceptions) has no reason to have advanced knowledge of computers; in general, they are familiar with office suites and use a couple of applications with varying degrees of skill. Here GNU/Linux has a clear problem, because UNIX as such was never conceived as a purely desktop system and was only later adapted with graphic interfaces such as X Window and the different desktops,

such as the current GNU/Linux ones: Gnome and KDE. Furthermore, the end user tends to be familiar with Windows systems (which have almost a 95% share of the desktop market).

In the case of desktops, GNU/Linux has a number of obstacles to overcome. One of the most critical ones is that it does not come preinstalled on machines, which obliges the user to have a certain amount of knowledge in order to be able to install it. Other reasons could be:

Note

The desktop environment is a battle yet to be waged by GNU/Linux systems; which need to defeat users' reluctance to switch systems and generate awareness of their ability to offer simple alternatives and applications that can handle the tasks demanded by users.

• User reluctance: a question a user may ask is: Why should I switch system? Will the new environment offer me the same thing? One of the basic reasons for changing will be quality software and its cost, since a large proportion will be free. On this point, we should consider the issue of illegal software. Users seem to consider that their software is free, when really they are in an illegal situation. GNU/Linux software offers good quality at a low cost (or at no cost in many cases), with several alternatives for the same job.

• Simplicity: users are normally lost if the system does not have similar reference points to those the user is already familiar with, such as interface behaviour or tools with similar functionality. Users generally expect not to have to spend too much extra time on learning how to handle the new system. GNU/Linux still has a few problems with more or less automatic installations, which means that a certain amount of knowledge is still required in order to install it correctly. On this point, we should mention the ease of installing it in different environments provided by recent desktop oriented distributions like Ubuntu [Ubu]. Another common problem concerns support for the PC hardware; even though it is improving all the time, manufacturers still don't pay enough attention to it (partly for reasons of market share). Until there is a clear intention in this regard, we will not be able to have the same support as other proprietary systems (like Windows). However, we should emphasise the work of the Linux kernel community to offer the right support for new technologies, in some cases by supporting the manufacturer or by preparing primary support (if not supported by the manufacturer) or alternative support to that offered by the manufacturer.

• Transparency: GNU/Linux environments have many complex mechanisms, such as daemons, services, difficult to configure ASCII files etc. For end users, it should be necessary to hide all of these

complexities by means of graphics programs, configuration wizards etc. This is the path taken by some distributions such as Red Hat, Mandriva, Ubuntu or SuSe.

• Support for known applications: a standard office suite user will face the problem of data portability or handling data formats. What to do with existing data? This problem is being solved daily, thanks to the office suites that are starting to have the functionalities a desktop user needs. For example, if we consider a migration from using a Windows Office suite, we can find suites such as OpenOffice (free software) that can read (and create) the formats of Office files (with some restrictions). Format compatibility is not difficult when it is open, but in the case of Windows, Microsoft continues to maintain a policy of closed formats; and a serious amount of work is needed in order to be able to use these formats, by means of reverse engineering (a fairly costly process). Also, in the Internet age, when information is supposed to move about freely, undocumented closed formats are more an obstacle than anything else. The best thing is to use open formats such as RTF (although these also have some problems because of the many versions of it that there are), or XML based formats (OpenOffice generates its own documents in XML), or PDF for read-only documents. We should also highlight recent efforts by the OpenOffice community to create the *standard open document* (used by the suite from versions 2.x), which have made it possible to have a free format as an ISO standard for document creation. This fact has obliged Microsoft to (partially) open its format in versions starting from Office 2007, to incorporate OpenXML formats.

• To provide valid alternatives: the software we stop using has to have alternatives that do the same job as the previous system. Most applications have one or several alternatives with similar, if not better, functionalities. On the Internet you can find different lists of (more or less complete) applications for GNU/Linux that match the functionality of Windows applications.

• Support for running applications for other systems: under some conditions it is possible to run applications for other UNIX systems (with the same architecture, for example, Intel x86), or for MS-DOS or Windows, through compatibility packages or some type of emulator.

Most of the problems that affect desktop migrations are being overcome slowly but surely and will allow us in future to have a larger number of GNU/Linux desktop users, who, as they increase, will have access to better applications encouraging software companies to start implementing versions for GNU/Linux.

In the case of companies, it can be overcome with a gentle migration, starting with servers and workstations, and then desktops after following an extensive training program for users in the new systems and applications.

A process that will help to a large extent is to introduce open code software in education and in public administrations, as in the case of Extremadura region in Spain with its GNU/Linux distribution called Linex; or recent measures for taking this software to primary education, or the measures taken by universities by running courses and subjects using these systems.

5. Migration workshop: case study analysis

In this workshop we will try to apply what we have learned in this unit to analyse some simple migration processes, and some detail of the required techniques (in the case of network techniques, we will look at these in the units on network administration).

We will consider the following case studies:

- Individual migration of a Windows desktop user to a GNU/Linux system.
- Migration of a small organisation with Windows systems and a few UNIX.
- Migration of a standalone Windows server to a Samba server running GNU/ Linux.

5.1. Individual migration of a Windows desktop user to a GNU/Linux system

A user considers migrating to GNU/Linux [Ray02b]. Normally, there will first be a period of cohabitation, so that the user can have both systems and use each one for a series of tasks: tasks will continue to be executed in Windows while the user learns about the new system and finds equivalent software or new software that does tasks for which no software was previously available.

Migration for a private user is perhaps one of the most complex processes; we need to offer users alternatives to what they commonly use, so that adaptation is as simple as possible and the user can adapt gradually and with ease to the new system.

A first possibility would be a dual installation [Ban01] [Sko03b] of the original system (Windows) together with the GNU/Linux system.

Note

Linux Hardware Howto: http://www.tldp.org/HOWTO/Hardware-HOWTO/index.html

A first step for a determined machine configuration will consist of checking that our hardware is compatible with Linux [Pri02], either from a list of hardware compatibility or by checking with the manufacturer if new components need to be purchased or the existing ones require a particular configuration. If we are unfamiliar with our hardware, we can check it through the Windows "device administrator" (in the control panel) or using some type of hardware recognition software. At the same time, an advisable method is to use LiveCD-type GNU/Linux distributions, which will allow us to check the functioning of GNU/Linux on our hardware without requiring a physical installation, since the only requirement is the possibility of booting the system from a CD/DVD (in some cases the BIOS configuration may have to be changed for this). There are Live CDs such as Knoppix [Knp] with great support for hardware checks and most GNU/Linux distributions tend to offer a Live CD in order to initially

check its functioning (in some cases, Ubuntu [Ubn] for example, the full installation can be done using the same Live CD). In any case, we should mention that checking with a specific Live CD does not mean that there will not be any problems with the final installation, either because the Live CD is not of the same GNU/Linux distribution that we eventually install or because the versions of the system and/or applications will not be the same.

Regarding the physical installation on disk, we will either need to have unpartitioned free disk space or, if we have FAT/32-type partitions, we can liberate space using programs that make it possible to adjust the size of partitions, reducing an existing partition (a previous data backup here is obviously advisable). Currently, most distributions support various disk partitioning and partition reduction schemes, although problems may arise depending on the distribution. If there is not enough space or there are partitions with file systems that present problems (like NTFS with some distributions), we may have to consider buying a new additional hard disk, to use totally or partially for GNU/Linux.

After checking the hardware, we will have to decide on the distribution of the GNU/Linux system that we will use (a possibility we mentioned before is to choose a Live CD that has been satisfactory and to install that distribution). If the user is inexperienced in GNU/Linux or only has basic computer knowledge, it is preferable to choose one of the more user-friendly distributions such as Fedora, Mandriva, SuSe, or similar (we would highlight the ease of Ubuntu in this regard). If we are more knowledgeable or tempted to experiment, we could try a Debian distribution. In the case of commercial distributions, on most occasions the distributions with compatible hardware (business versions like Red Hat and SuSe certify the hardware that they support), are installed perfectly without any problem and basic configurations are made that allow the operating system to be used immediately. During the process, we will have to install the software, which will normally be defined by sets of oriented software: for servers, specific applications or desktop applications, such as office suites, development applications (if we are interested in programming) etc.

Once the system is installed, we have to tackle the issue of sharing data [Gon00] [Kat01], how will we share the data between the two systems? or is it possible to share certain applications? There are various solutions for this:

- Indirect method: this consists of sharing data using a diskette for example. For this, the best thing are the utilities known as mtools, which allow transparent access to diskettes in MS-DOS format, and there are several commands that function in a very similar way to MS-DOS or Windows. These commands have exactly the same names as the original MS-DOS commands, except that they have an "m" in front, for example: mcd, mcopy, mdir, mdel, mformat, mtype etc.

- Direct method: this consists of using the file system in Windows directly. As we will see in the unit on local administration, GNU/Linux can read and write a large number of file systems, including FAT, FAT32, and NTFS (read only in some cases, although most distributions already include the ntfs-3g [Nt3] driver that allows writing). Mounting the Windows disk is required first and that makes it possible to incorporate the Windows file system into a point of the Linux file tree; for example, we could mount our Windows disk in /mnt/Windows and from this point access its folders and files for reading and writing. With ASCII text files, conversions need to be considered, since UNIX and Windows treat them differently: in UNIX, the end of a line has only one character, the line feed, ASCII 10, whereas Windows has two, the return and the line feed, characters ASCII 13 and 10 (as a curious note, in Mac it is ASCII 13). Which means that usually when we read a DOS/Windows ASCII file, it contains strange characters at the end of a line. There are editors such as emacs that handle them transparently and, in any case, there are GNU/Linux utilities that make it possible to convert them into another format (utilities such as duconv, recode, dos2UNIX, UNIX2dos).

- Use of applications: there are a few alternatives for running the applications (not all of them) for MS-DOS and Windows. For GNU/Linux there are MS-DOS emulators such as Dosemu [Sun02] or DOsBox, and for Windows there is the Wine [Win] software. It can run various Windows applications (for example, it can run some version of Office and Internet Explorer), but it is constantly being improved. If it is vital to run Windows applications, some commercial software can help us; these applications give extra support to Wine, for example, Win4Lin, CrossOver and in some cases special support for games like Cedega. Another potential solution is to use virtual machines; an example of extensively used software is VMware and VirtualBox, which creates a full PC as a virtual machine, simulated by the software, where a large number of different operating systems can be installed. VMware and VirtualBox are available in versions for Windows and for GNU/Linux, which makes it possible to have a GNU/Linux installed with a Windows running on it virtually, or a Windows installed with a virtual GNU/Linux. There are also other solutions of free virtual machines like QEmu, KVM, Bochs. In another segment, virtual machines or generically virtualisation is used oriented at the creation of virtual servers, with solutions such as VMware server or the open projects Xen, OpenVZ, Vserver; where it is possible to make several virtual machines running on an operating system coexist (normally through modifications to the kernel that support this virtualisation), or even directly on the hardware, with small layers of software.
  Aside from sharing the information (applications and/or data) you can search for GNU/Linux applications that replace the original Windows ones as the user gradually learns to use them and sees that they offer the expected functionalities.

Example

A typical case would be the office suite that can be migrated to OpenOffice, which has a high degree of compatibility with Office files and functions fairly similarly, or KOffice (for the KDE desktop), or GNumeric and AbiWord (for Gnome). Or, in the case of image processing, we can take Gimp, with similar functionalities to Photoshop. And numerous multimedia players: Xine, Mplayer (or also a version of RealPlayer). On the Internet we can find numerous lists of equivalent programs between Windows and GNU/Linux.

5.2. Migration of a small organisation with Windows systems and a few UNIX

Migration within an organisation (even a small one) has several difficulties: we will have different work environments and heterogeneous software, and, once more, users who are resistant to change.

Note

For examples of GNU/Linux equivalent applications, see:
http://www.linuxalt.com/
http://wiki.linuxquestions.org/wiki/Linux_software_equivalent_to_Windows_software
http://www.linuxrsp.ru/win-lin-soft/table-eng.htmlç

Now, let's consider an organisation with Windows machines and some UNIX machines as servers or workstations and somewhat "anarchic" users. For example, let's study the following situation: the organisation has a small local network of Windows machines shared by users as equal machines in a Windows workgroup (there are no Windows server domains).

The group is diverse: we have machines with Windows 98, ME, NT, XP, but configured for each user with the software needed for their daily jobs: whether Office, a browser, e-mail reader, or development environments for different language programmers (for example, C, C++, Java).

There are some extra hardware resources available, such as various printers connected to the local network (they accept TCP/IP jobs), which can be used from any point within the organisation. At the same time, there is a shared machine, with a few special resources, such as a scanner, CD recorder and directories shared by the network, where users can leave their own directories with their files for backup processes or to recover scanned images, for example.

We also have several workstations, in this case Sun Microsystem's SPARC, which are running Solaris (commercial UNIX of Sun). These stations are dedicated to development and to some scientific and graphics applications. These machines have NFS services for file sharing and NIS+ for handling the information of users who connect to them and who can do so from any machine in a transparent manner. Some of the machines include specific services; one is the company's web server and another is used as an e-mail server.

We are considering the possibility of migrating to GNU/Linux because of an interest in software development and the particular interest from some users to use this system.

Also, the migration will be made the most of in order to resolve certain problems related to security – some old Windows systems are not the best way of sharing files; we want to restrict use of the printer (the cost in paper and associated costs are high) to more reasonable quotas. At the same time we would

like users to have a certain amount of freedom, they will not be obliged to change system, although the suggestion will be made to them. And we will also take advantage in order to purchase new hardware to complement existing hardware, for example if the workstations require additional disk space, which imposes limits on e-mail and user accounts.

Following this small description of our organisation (in other more complex cases it could fill several pages or be a full document analysing the present situation and making future proposals), we can start to consider the possibilities for solving all this:

- What do we do with the current workstations? The cost of maintenance and software licenses is high. We need to cover the maintenance of faults in the stations, expensive hardware (in this case, SCSI disks) and also expensive memory extensions. The cost of the operating system and its updates is also expensive. In this case, we have two possibilities (depending on the budget that we have to make the change):
  - We can cut costs by converting the machines to GNU/Linux systems. These systems have a SPARC architecture and there are distributions that support this architecture. We could replace the services for their GNU/Linux equivalents; replacement would be virtually direct, since we already use a UNIX system.

  - Another possibility would be to eliminate Sun's proprietary hardware and to convert the stations into powerful PCs with GNU/Linux; this would make subsequent maintenance simpler, although the initial cost would be high.

- And what about the workstations software? If the applications have been developed in-house, it may be enough to compile them again or to make a simple adjustment to the new environment. If they are commercial, we will have to see whether the company can provide them in GNU/Linux environments, or if we can find replacements with a similar functionality. In the case of the developers, their environments of C, C++ and Java languages are easily portable; in the case of C and C++, gcc, the GNU compiler, can be used and there are numerous IDEs for development (KDevelop, Anjuta,...); or in the case of Java, the Sun kit can be used in GNU/Linux and in various open code environments (IBM's Eclipse or Netbeans).

- And what about users? For those who are interested in GNU/Linux systems, we can install dual equipment with Windows and GNU/Linux so that they can start to test the system and if they are interested, we can finally transfer to just the one GNU/Linux system. We can find two types of users: purely office suite users, who will basically need the suite, navigator and e-mail; all of which can be offered with a GNU/Linux desktop such as Gnome or KDE and software such as OpenOffice, Mozilla/Firefox navigator, and Mozilla Mail or Thunderbird e-mail (or any other Kmail,

Evolution...). They are more or less directly equivalent, it all depends on users' desire to test and use the new software. For developers, the change can be more direct, since they are offered many more environments and flexible tools; they could pass completely over to the GNU/Linux systems or work directly with the workstations.

- And the printers? We could establish a workstation as a printer server (whether through TCP/IP queues or Samba server), and control printing by means of quotas.

- The shared machine? The shared hardware can be left on the same machine or can be controlled from a GNU/Linux system. Regarding the shared disk space, it can be moved to a Samba server that will replace the current one.

- Do we expand the disk space? This will depend on our budget. We can improve control by means of a quota system that distributes space equitably and imposes limits on saturation.

5.3. Migration of a standalone Windows server to a Samba server running GNU/Linux

The basic required process tends to me much more extensive, consult the bibliography for the full steps to be taken.

In this case, the basic required process for a migration from a Windows server that shares files and a printer to a Samba server in a GNU/Linux system.

Thanks to software such as Samba, migration from Windows environments is very flexible and fast and even improves the machine's performance.

Let's suppose a machine belonging to a workgroup GROUP, sharing a printer called PRINTER and with a shared file called DATA, which is no more than the machine's D drive. Several Windows clients access the folder for reading/writing, within a local network with IP 192.168.1.x addresses, where x will be 1 for our Windows server, and the clients will have other values (192.168.x.x networks are often used as addresses to install private internal networks).

As part of our process we will build a Samba server, which is what, as we saw, will allow us to run the SMB/CIFS (server message block / common Internet file system) protocol in GNU/Linux. This protocol allows the file system and the printers to interact through networks on different operating systems. We can mount folders belonging to Windows on the GNU/Linux machines, or

part of the GNU/Linux folders on Windows and similarly with each other's printers. The server consists of two daemons (system processes) called smbd and nmbd.

The smbd process manages clients' requests from shared files or printers. The nmbd process manages the machines' names system and resources under the NetBIOS protocol (created by IBM). This protocol is independent from the network used (currently, in NT/2000/XP Microsoft generally uses Netbios over TCP/IP). The nmbd also offers WINS services, which is the name assignment service that is normally run on Windows NT/Server if we have a collection of machines; it is a sort of combination of DNS and DHCP for Windows environments. The process is somewhat complex, but to summarise: when a Windows machine starts up or has a static IP address or dynamic address through a DHCP server and additionally possibly a NetBIOS name (that the user assigns to the machine: in network identification), then the WINS client contacts the server to report its IP; if a network machine subsequently requests the NetBios name, the WINS server is contacted to obtain its IP address and communication is established. The nmbd runs this process on GNU/Linux.

Like any other network service, it should not be run without considering the risk activating it could entail, and how we can minimise this risk. Regarding Samba, we need to be aware of security issues, because we are opening part of our local or network files and printers. We will also have to check the communication restrictions properly in order to prevent access to unwanted users or machines. In this basic example, we will not comment on these issues; in a real case scenario, we would have to examine the security options and only allow access for those we want.

In the migration process, we will first have to configure the GNU/Linux system to support Samba [Woo00], we will need the Samba file systems support in the kernel (*smbfs*), which is normally already activated. We should add that currently there is additional support in the kernel through the cifs module [Ste07], which as of kernel version 2.6.20 is considered the default method, leaving smbfs as a secondary option. The cifs module offers support for new features related to the CIFS protocol (as an extension of SMB). Through "smbfs" and "cifs" file system names these modules allow us to conduct operations for mounting Windows file systems onto the Windows directory tree (mount -t smbfs or mount -t cifs). Apart from the fact that the kernel support is inclined towards the cifs module, there are some characteristics that may need smbfs support, which means that usually both modules are activated in the kernel. We should also mention the configuration issue, whereas smbfs bases its functioning on the Samba configuration (as we will see in the smb.conf file), the cifs module is given its configuration through the operations (for example, in the mounting process through mount).

In the case of using a Samba server, in addition to the kernel support, we will need to install the associated software packages: we will have to examine what packages related to Samba the distribution includes and install those associated to the functioning of the server. And also, if wanted, those related to Samba as a client, in the event we wish to be clients of Windows machines or to test resources shared with the Windows machines from our GNU/Linux system. In a Debian distribution, these packages are: samba, samba-common, smbclient, smbfs. It may also be interesting to install swat, which is a web-based graphics tool for Samba services administration. For our GNU/Linux Samba server [Woo00] [War03], for the proposed example, we will have to transfer the contents of the previous D disk (where we had our shared file system) from the original machine to the new machine and place its content in a path, like, /home/DATA, whether through a backup copy, FTP transfer, or using Samba as a client to transfer the files.

Regarding the use of GNU/Linux as a Samba client, it is fairly simple. Through the use of client commands for occasional use of the file system:

a) We mount a Windows shared directory (for instance, host being the name of the Windows server), on an existing predefined mounting point:

```
smbmount //host/carpeta /mnt/windows
```

b) We will place the access to the Windows folder of the host machine in our local directory, accessing in the directory tree:

```
/mnt/windows
```

c) Next, when it is no longer in use we can dismount the resource with:

```
smbumount /mnt/windows
```

If we are not aware of the shared resources, we can obtain a list with:

```
smbclient -L host
```

And we can also use smbclient //host/folder, which is a similar program to an FTP client.

In the event of wanting to make the file systems available permanently, or to provide certain special configurations, we can study the use of *mount* directly (the smbxxxx utilities use it), whether with the smbfs or cifs file systems (supported in the kernel), taking the parameters into account (Windows users/groups authentication or other service parameters) that we will have to provide depending on the case, and of the pre-existing Samba configuration [Ste07].

Note

Always consult the **man** pages, or manuals, that come with the software package.

In the case of the Samba server, once we have installed all the Samba software, we will have to configure the server through its configuration file. Depending on the version (or distribution), this file may be in /etc/smb.conf or in /etc/samba/smb.conf. The options shown here belong to a Samba 3.x.x installed on a Debian distribution system. Other versions may have a few minor modifications.

During the installation of the software packages we will normally be asked for data regarding its configuration. In the case of Samba, we will be asked for the workgroup to be served; we will have to place the same group name as in Windows. We will also be asked if we want encrypted passwords (advisable for security reasons, in Windows 9x they were sent in raw text, in a clear case of scarce security and high system vulnerability).

Next we will look at the process of configuring the file smb.conf. This file has three main sections:

1) *Global* (basic functioning characteristics).

2) *Browser* (controls what other machines see of our resources).

3) *Share* (controls what we share).

In this file's extensive manual we can see the available options (man smb.conf). We will edit the file with an editor and see some of the file's lines (characters '#' or ';' at the beginning of a line are comments: If the line contains ';' it is a comment; to enable a line, if it is an optional configuration line we must edit it and remove the ';'):

```
workgroup = GROUP
```

This shows the Windows workgroup that the Windows client machines will be members of.

```
server string = %h server (Samba %v)
```

We can place a text description of our server. The *h* and the *v* that appear are variables of Samba that refer to the host name and version of Samba. For security reasons, it is a good idea to remove the *v,* since this will inform the exterior what version of Samba we have; if there are known security bugs, this can be used.

```
hosts allow = 192.168.1
```

This line may or may not be present, and we can include it to enable what hosts will be served; in this case, all of those in the 192.168.1.x range.

```
printcap name = /etc/printcap
```

The printcap file is where GNU/Linux stores the printers' definition, and this is where Samba will look for information about them.

```
guest account = nobody
```

This is the guest account. We can create a different account, or just enable access to Samba for the users registered on the GNU/Linux system.

```
log file = /var/log/samba/log.%m
```

This line tells us where the Samba log files will be stored. One is stored per client (variable *m* is the name of the connected client).

```
encrypt passwords = true
```

For security reasons it is advisable to use encrypted passwords if we have client machines with Windows 98, NT or above. These passwords are saved in a /etc/samba/smbpasswd file, which is normally generated for users of the Samba installation. Passwords can be changed with the *smbpasswd* command. There is also an option called *UNIX password sync*, which allows the change to be simultaneous for both passwords (Samba user and Linux user).

Next, we will jump to the Share Definitions section:

```
[homes]
```

These lines allow access to the users' accounts from the Windows machines. If we don't want this, we will add some ';' to the start of these lines, and when the machines connect they will see the name comment. In principle, writing is disabled, to enable it, you just have to set "yes" as the writable option.

Any sharing of a specific directory (Samba tends to call a group of shared data a partition), we will proceed as shown in the examples that appear (see, for example the definition of sharing the CD-ROM in the lines that start with [cdrom]). In path we will place the access route.

Example

In our case, for example, we would give the name DATA to the partition on the route /home/DATA, where we had copied the D disk from the original Windows machine and the path where it can be found, in addition to a large group of options that can be modified, users authorised to access them and the way of doing so.

Note

See: man smb.conf

There is also a profiles definition, that makes it possible to control the profiles of Windows users, in other words, the directory where their Windows desktop configuration is saved, the start up menu etc.

The method is similar for the printers: a partition is made with the printer name (the same one given in GNU/Linux), and in the path we place the queue address associated to the printer (in GNU/Linux we will find it in: /var/spool/samba/PRINTER). And the option printable = yes, if we want jobs to be sent with Samba. And we can also restrict user access (valid users).

Once we have made these changes we will just have to save them and reinitiate Samba so that it can read the new configuration. In Debian:

```
/etc/init.d/samba restart
```

Now, our shared directory and the printer through Samba will be available to serve users without them noticing any difference in relation to the previous connections with the Windows server.

Activities

1) In the GNU/Linux services description, do we find we are missing any functionality? What other type of services would we add?

2) In the second case study of the tutorial (the one of the organisation), how would you change the IT infrastructure if you had zero budget, an average budget, or a high budget? Present some alternative solutions to the ones shown.

3) Virtualisation technologies like VMware Workstation or VirtualBox, virtual machine through software, which can install operating systems on a virtual PC. You can obtain the software from www.vmware.com or www.virtualbox.org. Test (in the case of having a Windows license) installing it on Windows, and then on GNU/Linux on the virtual PC (or the other way around). What advantages does this method for sharing operating systems offer? What problems does it cause?

4) If we have two machines for installing a Samba server, we can test the server installation or configuration in configurations of Samba UNIX client-Windows server, or Windows client-Samba server in GNU/Linux. You can test it on a single machine using the same machine as a Samba server and client.

Bibliography

Other sources of reference and information

[LPD] Linux Documentation Project offers Howtos regarding different aspects of a GNU/Linux system and a set of more detailed manuals.

[Mor03] Good reference for the configuration of Linux systems, with some case studies in different environments; comments on different distributions of Debian and Red Hat.

Basic tools for the administrator

Josep Jorba Esteve

PID_00148464

UOC
Universitat Oberta de Catalunya
www.uoc.edu

Index

Introduction

On a daily basis, an administrator of GNU/Linux systems has to tackle a large number of tasks. In general, the UNIX philosophy does not have just one tool for every task or just one way of doing things. What is common is for UNIX systems to offer a large number of more or less simple tools to handle the different tasks.

It will be the combination of the basic tools, each with a well-defined task that will allow us to resolve a problem or administration task.

Note

GNU/Linux has a very broad range of tools with basic functionalities, whose strength lies in their combination.

In this unit we will look at different groups of tools, identify some of their basic functions and look at a few examples of their uses. We will start by examining some of the standards of the world of GNU/Linux, which will help us to find some of the basic characteristics that we expect of any GNU/Linux distribution. These standards, such as LSB (or Linux standard base) [Linc] and FHS (filesystem hierarchy standard) [Linb], tell us about the tools we can expect to find available, a common structure for the file system, and the various norms that need to be fulfilled for a distribution to be considered a GNU/Linux system and to maintain shared rules for compatibility between them.

For automating administration tasks we tend to use commands grouped into shell scripts (also known as command scripts), through language interpreted by the system's shell (command interpreter). In programming these shell scripts we are allowed to join the system's commands with flow control structures, and thus to have a fast prototype environment of tools for automating tasks.

Another common scheme is to use tools of compiling and debugging high level languages (for example C). In general, the administrator will use them to generate new developments of applications or tools, or to incorporate applications that come as source code and that need to be adapted and compiled.

We will also analyse the use of some graphics tools with regards to the usual command lines. These tools tend to facilitate the administrator's tasks but their use is limited because they are heavily dependent on the GNU/Linux distribution and version. Even so, there are some useful exportable tools between distributions.

Finally, we will analyse a set of essential tools for maintaining the system updated, the package management tools. The software served with the GNU/Linux distribution or subsequently incorporated is normally offered in units known as packages, which include the files of specific software, plus the vari-

ous steps required in order to prepare the installation and then to configure it or, where applicable, to update or uninstall specific software. And every distribution tends to carry management software for maintaining lists of installed or installable packages, as well as for controlling existing versions or various possibilities of updating them through different original sources.

1. Graphics tools and command line

There are a large number of tools, of which we will examine a small share in this and subsequent modules, which are provided as administration tools by third parties, independent from the distribution, or by the distributor of the GNU/Linux system itself.

These tools may cover more or fewer aspects of the administration of a specific task and can appear with various different interfaces: whether command line tools with various associated configuration options and/or files or text tools with some form of menus; or graphics tools, with more suitable interfaces for handling information, wizards to automate the tasks or web administration interfaces.

All of this offers us a broad range of possibilities where administration is concerned, but we will always have to evaluate the ease of using them with the benefits of using them, and the knowledge of the administrator responsible for these tasks.

The common tasks of a GNU/Linux administrator can include working with different distributions (for example, the ones we will discuss Fedora [Fed] or Debian [Debb] or any other) or even working with commercial variants of other UNIX systems. This entails having to establish a certain way of working that allows us to perform the tasks in the different systems in a uniform manner.

For this reason, throughout the different modules we will try to highlight the most common aspects and the administration techniques will be mostly performed at a low level through a command line and/or the editing of associated configuration files.

Any of the GNU/Linux distributions tends to include command line, text, or especially, graphics tools to complement the above and to a greater or lesser degree simplify task administration [Sm02]. But we need to take several things into account:

a) These tools are a more or less elaborate interface of the basic command line tools and corresponding configuration files.

b) Normally they do not offer all the features or configurations that can be carried out at a low level.

c) Errors may not be well managed or may simply provide messages of the type "this task could not be performed".

d) The use of these tools hides, sometimes completely, the internal functioning of the service or task. Having a good understanding of the internal functioning is basic for the administrator, especially if the administrator is responsible for correcting errors or optimising services.

e) These tools are useful for improving production once the administrator has the required knowledge to handle routine tasks more efficiently and to automate them.

f) Or, in the opposite case, the task may be so complex, require so many parameters or generate so much data, that it may become impossible to control it manually. In these cases, the high level tools can be very useful and make practicable tasks that are otherwise difficult to control. For example, this category would include visualisation tools, monitorisation tools, and summaries of tasks or complex services.

g) For automating tasks, these tools (of a higher level) may not be suitable: they may not have been designed for the steps that need taking or may perform them inefficiently. For example, a specific case would be creating users, where a visual tool can be very attractive because of the way of entering the data; but what if instead of entering one or a few users we want to enter a list of tens or hundreds of them? if not prepared for this, the tool will become totally inefficient.

h) Finally, administrators normally wish to personalise their tasks using the tools they find most convenient and easy to adapt. In this aspect, it is common to use basic low-level tools, and shell scripts (we will study the basics in this unit) combining them in order to form a task.

We may use these tools occasionally (or daily), if we have the required knowledge for dealing with errors that can arise or to facilitate a process that the tool was conceived for, but always controlling the tasks we implement and the underlying technical knowledge.

2. Standards

Standards, whether generic of UNIX or particular to GNU/Linux, allow us to follow a few basic criteria that guide us in learning how to execute a task and that offer us basic information for starting our job.

In GNU/Linux we can find standards, such as the FHS (*filesystem hierarchy standard*) [Linb], which tells us what we can find in the our system's file system structure (or where to look for it), or the LSB (*Linux standard base),* which discusses the different components that we tend to find in the systems [Linc].

Note

See FHS in: www.pathname.com/fhs

The **FHS** *filesystem hierchachy standard* describes the main file system tree structure (/), which specifies the structure of the directories and the main files that they will contain. This standard is also used to a greater or lesser extent for commercial UNIX, where originally there were many differences that made each manufacturer change the structure as they wished. The standard originally conceived for GNU/Linux was made to normalise this situation and avoid drastic changes. Even so, the standard is observed to varying degrees, most distributions follow a high percentage of the FHS, making minor changes or adding files or directories that did not exist in the standard.

Note

The FHS standard is a basic tool that allows us to understand the structure and functionality of the system's main file system.

A basic directories scheme could be:

- /bin: basic system utilities, normally programs used by users, whether from the system's basic commands (such as /bin/ls, list directory), shells (/bin/bash) etc.

- /boot: files needed for booting the system, such as the image of the Linux kernel, in /boot/vmlinuz.

- /dev: here we will find special files that represent the different possible devices in the system, access to peripherals in UNIX systems is made as if they were files. We can find files such as /dev/console, /dev/modem, /dev/mouse, /dev/cdrom, /dev/floppy... which tend to be links to more specific devices of the driver or interface type used by the devices: /dev/mouse, linked to /dev/psaux, representing a PS2 type mouse; or /dev/cdrom to /dev/hdc, a CD-ROM that is a device of the second IDE connector and master. Here we find IDE devices such as /dev/hdx, scsi /dev/sdx... with x varying according to the number of the device. Here we should mention that initially this directory was static, with the files predefined, and/or configured at specific moments, nowadays we use dynamic technology

techniques (such as hotplug or udev), that can detect devices and create /dev files dynamically when the system boots or while running, with the insertion of removable devices.

- /etc: configuration files. Most administration tasks will need to examine or modify the files contained in this directory. For example: /etc/passwd contains part of the information on the system's user accounts.

- /home: it contains user accounts, meaning the personal directories of each user.

- /lib: the system's libraries, shared by user programs, whether static (.a extension) or dynamic (.so extension). For example, the standard C library, in libc.so files or libc.a. Also in particular, we can usually find the dynamic modules of the Linux kernel, in /lib/modules.

- /mnt: point for mounting (mount command) file systems temporarily; for example: /mnt/cdrom, for mounting a disk in the CD-ROM reader temporarily.

- /media: for common mounting point of removable devices.

- /opt: the software added to the system after the installation is normally placed here; another valid installation is in /usr/local.

- /sbin: basic system utilities. They tend to be command reserved for the administrator (root). For example: /sbin/fsck to verify the status of the file systems.

- /tmp: temporary files of the applications or of the system itself. Although they are for temporary running, between two executions the application/service cannot assume that it will find the previous files.

- /usr: different elements installed on the system. Some more complete system software is installed here, in addition to multimedia accessories (icons, images, sounds, for example in: /usr/share) and the system documentation (/usr/share/doc). It also tends to be used in /usr/local for installing software.

- /var: log or status type files and/or error files of the system itself and of various both local and network services. For example, log files in /var/log, e-mail content in /var/spool/mail, or printing jobs in /var/spool/lpd.

These are some of the directories defined in the FHS for the root system, then for example it specifies some subdivisions, such as the content of /usr and /var, and the typical data and/or executable files expected to be found at minimum in the directories (see references to FHS documents).

Regarding the distributions, Fedora/Red Hat follows the FHS standard very closely. It only presents a few changes in the files present in /usr, /var. In /etc there tends to be a directory per configurable component and in /opt, /usr/local there is usually no software installed unless the user installs it. Debian follows the standard, although it adds some special configuration directories in /etc.

Another standard in progress is the LSB (*Linux standard base*) [Linc]. Its idea is to define compatibility levels between the applications, libraries and utilities, so that portability of applications is possible between distributions without too many problems. In addition to the standard, they offer test sets to check the compatibility level. LSB in itself is a collection of various standards applied to GNU/Linux.

Note

See standard specifications: http://www.linuxfoundation.org/en/Specifications

3. System documentation

One of the most important aspects of our administration tasks will be to have the right documentation for our system and installed software. There are numerous sources of information, but we should highlight the following:

a) man is by far the best choice of help. It allows us to consult the GNU/Linux manual, which is grouped into various sections corresponding to administration commands, file formats, user commands, C language calls etc. Normally, to obtain the associated help, we will have enough with:

```
man command
```

Every page usually describes the command together with its options and, normally, several examples of use. Sometimes, there may be more than one entry in the manual. For example, there may be a C call with the same name as a command; in this case, we would have to specify what section we want to look at:

```
man n command
```

with *n* being the section number.

There are also several tools for exploring the manuals, for example *xman* and *tkman*, which through a graphic interface help to examine the different sections and command indexes. Another interesting command is *apropos* word, which will allow us to locate man pages that discuss a specific topic (associated with the word).

b) info is another common help system. This program was developed by GNU to document many of its tools. It is basically a text tool where the chapters and pages can be looked up using a simple keyboard-based navigation system.

c) Applications documentation: in addition to certain man pages, it is common to include extra documentation in the applications, in the form of manuals, tutorials or simple user guides. Normally, these documentation components are installed in the directory /usr/share/doc (or /usr/doc depending on the distribution), where normally a directory is created for each application package (normally the application can have a separate documentation package).

d) Distributions' own systems. Red Hat tends to come with several CDs of consultation manuals that can be installed on the system and that come in HTML or PDF formats. Fedora has a documentation project on its webpage. Debian offers its manuals in the form of one more software package that is usually installed in /usr/doc. At the same time, it has tools that classify the documentation in the system, organising it by means of menus for visualisation, such as *dwww* or *dhelp*, which offer web interfaces for examining the system's documentation.

e) Finally, X desktops, such as Gnome and KDE, usually also carry their own documentation systems and manuals, in addition to information for developers, whether in the form of graphic help files in their applications or own applications that compile all the help files (for example devhelp in Gnome).

4. Shell scripting

The generic term shell is used to refer to a program that serves as an interface between the user and the GNU/Linux system's kernel. In this section, we will focus on the interactive text shells, which are what we will find as users once we have logged in the system.

> The shell is a system utility that allows users to interact with the kernel through the interpretation of commands that the user enters in the command line or files of the shell script type.

The shell is what the users see of the system. The rest of the operating system remains mostly hidden from them. The shell is written in the same way as a user process (program); it does not form part of the kernel, but rather is run like just another user program.

When our GNU/Linux system starts up, it tends to offer users an interface with a determined appearance; the interface may be a text or graphic interface. Depending on the modes (or levels) of booting the system, whether with the different text console modes or modes that give us a direct graphic start up in X Window.

In graphic start up modes, the interface consists of an access administrator to manage the user login procedure using a graphic cover page that asks for the corresponding information to be entered: user identification and passsword. Access managers are common in GNU/Linux: xdm (belonging to X Window), gdm (Gnome) and kdm (KDE), as well as a few others associated to different window managers. Once we have logged in, we will find ourselves in the X Window graphic interface with a windows manager such as Gnome or KDE. To interact through an interactive shell, all we will need to do is to open one of the available terminal emulation programs.

If our access is in console mode (text), once logged in, we will obtain direct access to the interactive shell.

Another case of obtaining an interactive shell is by remote access to the machine, whether through any of the text possibilities such as telnet, rlogin, ssh, or graphic possibilities such as the X Window emulators.

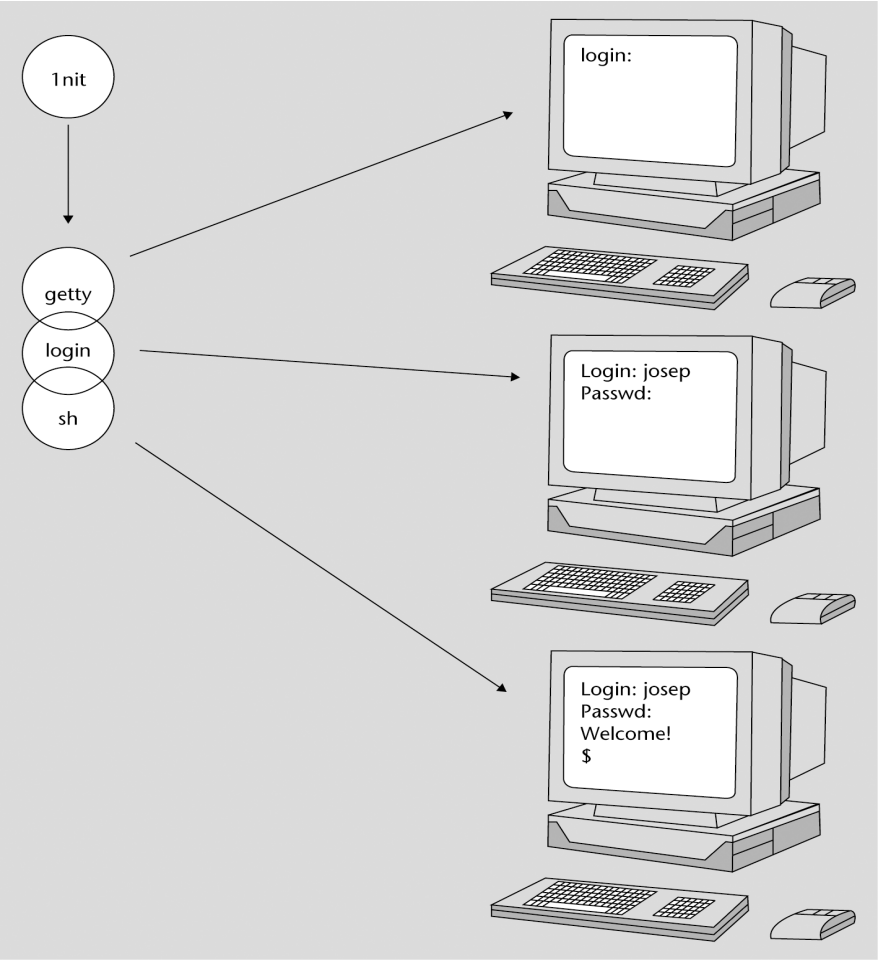

Figure 1. Example of starting up a text shell textual and the system processes involved [Oke]

4.1. Interactive *shells*

Having initiated the interactive shell [Qui01], the user is shown a prompt, indicating that a command line may be entered. After entering it, the shell becomes responsible for validating it and starting to run the required processes, in a number of phases:

- Reading and interpreting the command line.
- Evaluating wildcard characters such as $ * ? and others.
- Managing the required I/O redirections, pipes and background processes (&).
- Handling signals.
- Preparing to run programs.

Normally, command lines will be ways of running the system's commands, interactive shell commands, starting up applications or shell scripts.

Shell scripts are text files that contain command sequences of the system, plus a series of internal commands of the interactive shell, plus the necessary control structures for processing the program flow (of the type *while, for* etc.).

The system can run script files directly under the name given to the file. To run them, we invoke the shell together with the file name or we give the shell script execution permissions.

To some extent, we can see shell script as the code of an interpreted language that is executed on the corresponding interactive shell. For the administrator, shell scripts are very important, basically for two reasons:

1) The system's configuration and most of the services are provided through tools in the form of shell scripts.

2) The main way of automating administration processes is creating shell scripts.

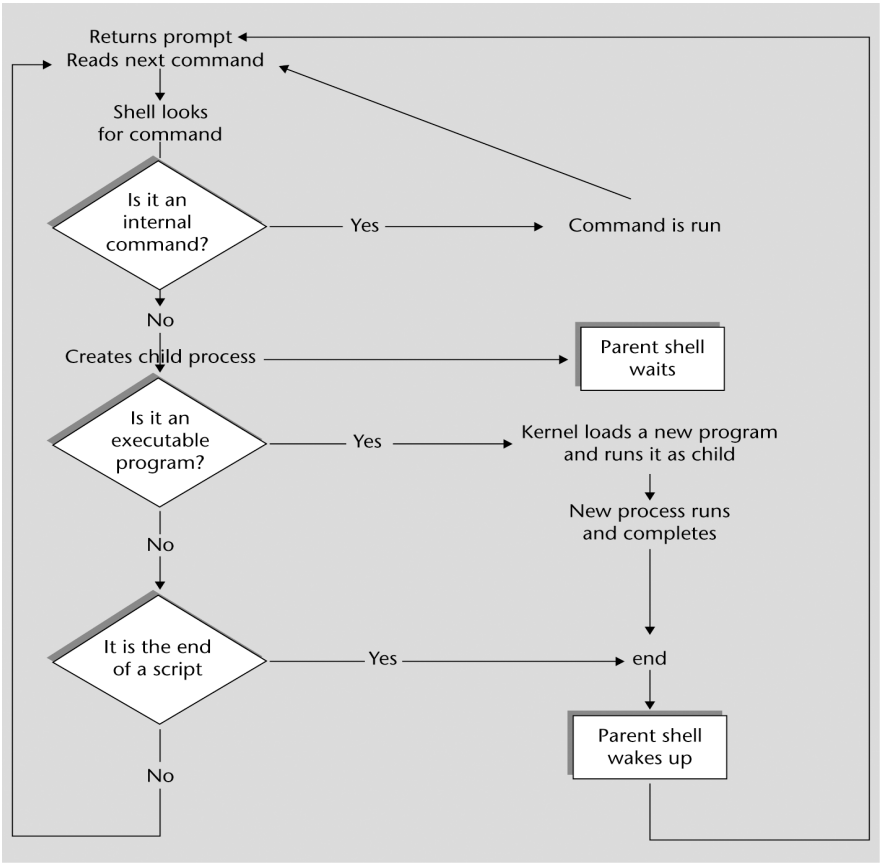

Figure 2. Basic shell flow control

All the programs that are invoked by a shell possess three predefined files, specified by the corresponding file handles. By default, these files are:

1) *standard input:* normally assigned to the terminal's keyboard (console); uses file handle number 0 (in UNIX the files use whole number file handles).

2) *standard output:* normally assigned to the terminal's screen; uses file handle 1.

3) *standard error:* normally assigned to the terminal's screen; uses file handle 2.

This tells us that any program run from the shell by default will have the input associated to the terminal's keyboard, the output associated to the screen, and that it will also send errors to the screen.

Also, the shells tend to provide the three following mechanisms:

1) Redirection: given that I/O devices and files are treated the same way in UNIX, the shell simply handles them all as files. From the user's point of view, the file handles can be reassigned so that the data flow of one file handle goes to any other file handle; this is called redirection. For example, we refer to redirecting file handles 0 or 1 as redirecting standard I/O.

2) Pipes: a program's standard output can be used as another's standard input by means of pipes. Various programs can be connected to each other using pipes to create what is called a pipeline.

3) Concurrence of user programs: users can run several programs simultaneously, indicating that they will be run in the background, or in the foreground, with exclusive control of the screen. Another way consists of allowing long jobs in the background while interacting with the shell and with other programs in the foreground.

In practice, in UNIX/Linux these shells entail:

- **Redirection**: a command will be able to receive input or output from other files or devices.

Example

let's see

```
command op file
```

where op may be:

- < : receive input from file.
- > : send output to file.
- >> : it indicates to add the output (by default, with > the file is created again).

- Pipes: chaining several commands, with transmission of their data:
  ```
  command1 | command2 | command3
  ```

- This instruction tells us that command1 will receive input possibly from the keyboard, send its output to command2, which will receive it as input and produce output towards command3, which will receive it and send its output to the standard output (by default, the screen).

- Background concurrence: any command executed with the '&' at the end of the line will be run in the background and the prompt of the shell will be returned immediately while it continues to be executed. We can follow the execution of commands with the *ps* command and its options, which allows us to observe the status of the system's processes. And we also have the *kill* order, which allows us to eliminate processes that are still being run or that have entered an error condition: *kill -9 PID* allows us to kill the process with PID identification number. PID is the identifier associated to the process, a whole number assigned to it by the system and that can be obtained using the *ps* command.

4.2. Shells

The shell's independence in relation to the operating system's kernel (the shell is just an interface layer), allows us to have several of them on the system [Qui01]. Although some of the more frequent ones are:

a) The Bash (initialism for *Bourne-again shell*). The default GNU/Linux shell.

b) The Bourne shell (sh). This has always been the standard UNIX shell, and the one that all UNIX systems have in some version. Normally, it is the administrator's default shell (root). In GNU/Linux it tends to be Bash, an improved version of the Bourne shell, which was created by Stephen Bourne at AT&T at the end of the seventies. The default prompt tends to be a '$' (in root a '#').

c) The Korn shell (ksh). It is a supergroup of Bourne (some compatibility is maintained), written at AT&T by David Korn (in the mid eighties), which some functionalities of Bourne and C, with some additions. The default prompt is the $.

d) The C shell (csh). It was developed at the University of Berkeley by Bill Joy towards the end of the seventies and has a few interesting additions to Bourne, like a command log, alias, arithmetic from the command line, it completes file names and controls jobs in the background. The default prompt for users is '%'. UNIX users tend to prefer this shell for interaction, but UNIX administrators prefer to use Bourne, because the scripts tend to be more compact and to execute faster. At the same time, an advantage of the scripts in C shell is that, as the name indicates, the syntax is based on C language (although it is not the same).

e) Others, such as restricted or specialised versions of the above.

The Bash (*Bourne again shell*) [Bas] [Coo] has grown in importance since it was included in GNU/Linux systems as the default shell. This shell forms part of the GNU software project. It is an attempt to combine the three preceding shells (Bourne, C and Korn), maintaining the syntax of the original Bourne shell. This is the one we will focus on in our subsequent examples.

A rapid way of knowing what shell we are in as users is by using the variable $*SHELL*, from a command line with the instruction:

```
echo $ SHELL
```

We will find that some aspects are common to all shells:

- They all allow shell scripts to be written, which are then interpreted executing them either by the name (if the file has an execution permission) or by passing it as a parameter to the command of the shell.

- System users have a default shell associated to them. This information is provided upon creating the users' accounts. The administrator will assign a shell to each user, or otherwise the default shell will be assigned (*bash* in GNU/Linux). This information is saved in the passwords file in /etc/passwd and can be changed with the *chsh* command, this same command with the option -l will list the system's available shells (see also */etc/shells*).

- Every shell is actually an executable command, normally present in the /bin directories in GNU/Linux (or /usr/bin).

• Shell scripts can be written in any of them, but adjusting to each one's syntax, which is normally different (sometimes the differences are minor). The construction syntax, as well as the internal commands, are documented in every shell's man page (*man bash* for example).

• Every shell has some associated start up files (initialisation files), and every user can adjust them to their needs, including code, variables, paths...

• The capacity in the programming lies in the combination of each shell's syntax (of its constructions), with the internal commands of each shell, and a series of UNIX commands that are commonly used in the scripts, like for example *cut*, *sort*, *cat*, *more*, *echo*, *grep*, *wc*, *awk*, *sed*, *mv*, *ls*, *cp*...

Note

To program a shell it is advisable to have a good knowledge of these UNIX commands and of their different options.

• If as users we are using a specific shell, nothing prevents us from starting up a new copy of the shell (we call it a subshell), whether it is the same one or a different one. We simply invoke it through the name of the executable, whether *sh*, *bash*, *csh* or *ksh*. Also when we run a shell script a subshell is launched with the corresponding shell for executing the requested script.

Some basic differences between them [Qui01]:

a) Bash is the default shell in GNU/Linux (unless otherwise specified in creating the user account). In other UNIX systems it tends to be the Bourne shell (sh). Bash is compatible with sh and also incorporates some features of the other shells, csh and ksh.

b) Start-up files: sh, ksh have *.profile* (in the user account, and is executed in the user's login) and ksh also tends to have a *.kshrc* which is executed next, csh uses *.login* (it is run when the user login initiates one time only), *.logout* (before leaving the user's session) and *.cshrc* (similar to the *.profile,* in each initiated C subshell). And Bash uses the *.bashrc* and the *.bash_profile*. Also, the administrator can place common variables and paths in the */etc/profile* file that will be executed before the files that each

user has. The shell start-up files are placed in the user's account when it is created (normally they are copied from the */etc/skel* directory), where the administrator can leave some skeletons of the prepared files.

c) The system or service configuration scripts are usually written in Bourne shell (sh), since most UNIX systems used them this way. In GNU/Linux we can also find some in Bash and also in other script languages not associated to the shell such as Perl or Python.

d) We can identify what shell the script is run on using the *file* command, for example *file <scriptname>*. Or by examining the first line of the script, which tends to be: *#!/bin/name*, where the name is *bash, sh, csh, ksh*... This line tells us, at the moment of running the script, what shell needs to be used to interpret it (in other words, what subshell needs to be launched in order to run it). It is important for all scripts to contain it, since otherwise they will try to run the default shell (Bash in our case) and the syntax may not be the right one, causing many syntax errors in the execution.

4.3. System variables

Some useful system variables (we can see them using the *echo* command for example), which can be consulted in the command line or within the programming of the shell scripts are:

Variable	Value Example	Description
HOME	/home/juan	Root directory of the user
LOGNAME	juan	User ID at *login*
PATH	/bin:/usr/local/bin:/usr/X11/bin	Paths
SHELL	/bin/bash	User *shell*
PS1	$	*Shell* prompt, the user can change it
MAIL	/var/mail/juan	E-mail directory
TERM	xterm	Type of terminal used by the user
PWD	/home/juan	Current user directory

The different variables of the environment can be seen using the *env* command. For example:

```
$ env
SSH_AGENT_PID = 598
MM_CHARSET = ISO-8859-15
TERM = xterm
DESKTOP_STARTUP_ID =
SHELL = /bin/bash
```

```
WINDOWID = 20975847
LC_ALL = es_ES@euro
USER = juan
LS_COLORS = no = 00:fi = 00:di = 01;34:ln = 01;
SSH_AUTH_SOCK = /tmp/ssh-wJzVY570/agent.570
SESSION_MANAGER = local/aopcjj:/tmp/.ICE-unix/570
USERNAME = juan
PATH=/soft/jdk/bin:/usr/local/bin:/usr/bin:/bin:/usr/bin/
X11:/usr/games
MAIL = /var/mail/juan
PWD = /etc/skel
JAVA_HOME = /soft/jdk
LANG = es_ES@euro
GDMSESSION = Gnome
JDK_HOME = /soft/jdk
SHLVL = 1
HOME = /home/juan
GNOME_DESKTOP_SESSION_ID = Default
LOGNAME = juan
DISPLAY = :0.0
COLORTERM = gnome-terminal
XAUTHORITY = /home/juan/.Xauthority
_ = /usr/bin/env
OLDPWD = /etc
```

4.4. Programming scripts in Bash

Here we will look at some basic concepts of the shell scripts in Bash, we advise further reading in [Bas] [Coo].

All Bash scripts have to start with the line:

```
#!/bin/bash
```

This line indicates the shell used by the user, the one active at the time, what shell is needed for running the script that appears next.

The script can be run in two different ways:

1) By running directly from the command line, on condition it has an execution permission. If this is not the case, we can establish the permission with: *chmod +x script*.

2) By running through the shell, we call on the shell explicitly: */bin/bash script*.

We should take into account that, irrespective of the method of execution, we are always creating a subshell where our script will be run.

4.4.1. Variables in Bash

The assignment of variables is done by:

```
variable = value
```

The value of the variable can be seen with:

```
echo $variable
```

where '$' refers us to the variable's value.

The default variable is only visible in the script (or in the shell). If the variable needs to be visible outside the script, at the level of the shell or any subshell that is generated a posteriori, we will need to "export" it as well as assign it. We can do two things:

- Assign first and export after:

```
var=value
export var
```

- Export during assignment:

```
export var=value
```

In Bash scripts we have some accessible predetermined variables:

- $1-$N: It saves past arguments as parameters to the script from the command line.
- $0: It saves the script name, it would be parameter 0 of the command line.
- $*: It saves all parameters from 1 to N of this variable.
- $: It saves both parameters, but with double inverted commas (" ") for each of them.
- $?: "Status": it saves the value returned by the most recent executed command. Useful for checking error conditions, since UNIX tends to return 0 if the execution was correct, and a different value as an error code.

Another important issue regarding assignments is the use of inverted commas:

- Double inverted commas allow everything to be considered as a unit.

- Single inverted commas are similar, but ignore the special characters inside them.

- Those pointed to the left (`command`) are used for evaluating the inside, if there is an execution or replacement to be made. First the content is executed, and then what there was is replaced by the result of the execution. For example: var = 'ls' saves the list of the directory in $var.

4.4.2. Comparisons

For conditions the order *test expression* tends to be used or directly *[expression]*. We can group available conditions in:

- Numerical comparison: -eq, -ge, -gt, -le, -lt, -ne, corresponding to: equal to, greater than or equal to (ge), greater than, less than or equal to (le), less than, not equal to.

- Chain comparison: :=, !=, -n, -z, corresponding to chains of characters: equal, different, with a greater length than 0, length equal to zero or empty.

- File comparison: -d, -f -r, -s, -w, -x. The file is: a directory, an ordinary file, is readable, is not empty, is writable, is runnable.

- Booleans between expressions: !, -a, -o, conditions of not, and, and or.

4.4.3. Control structures

Regarding the script's internal programming, we need to think that we are basically going to find:

- Commands of the operating system itself.
- Internal commands of the Bash (see: man bash).
- Programming control structures (for, while...), with the syntax of Bash.

The basic syntax of control structures is as follows:

a) Structure *if...then,* evaluates the expression and if a certain value is obtained, then the commands are executed.

```
if [ expresion ]
 then
  commands
```

```
fi
```

b) Structure if...then...else, evaluates the expression and if a certain value is obtained then the commands1 are executed, otherwise comands2 are executed:

```
if [ expresion ]
 then
  commands1
 else
  commands2
fi
```

c) Structure if..then...else if...else, same as above, with additional if structures.

```
if [ expresion ]
 then
  commands
 elif [ expresion2 ]
 then
  commands
 else
  commands
fi
```

d) Structure *case select*, multiple selection structure according to the selection value (in *case*)

Note

Shells such as Bash offer a wide set of control structures that make them comparable to any other language.

```
case string1 in
 str1)
  commands;;
 str2)
  commands;;
 *)
  commands;;
esac
```

e) Loop *for*, replacement of the variable for each element of the list:

```
for var1 in list
do
  commands
done
```

f) Loop *while*, while the expression is fulfilled:

```
while [ expresion ]
```

```
do
  commands
done
```

g) Loop *until*, until the expression is fulfilled:

```
until [ expression ]
do
  commands
done
```

h) Declaration of functions:

```
fname() {
  commands
}
```

or with a call accompanied by parameters:

```
fname2(arg1,arg2...argN) {
  commands
}
```

and function calls with fname or fname2 p1 p2 p3 ... pN.

5. Package management tools

In any distribution, the packages are the basic item for handling the tasks of installing new software, updating existing software or eliminating unused software.

Basically, a package is a set of files that form an application or the combination of several related applications, normally forming a single file (known as a package), with its own format, normally compressed, which is distributed via CD/DVD or downloading service (ftp or http repositories).

The use of packages is helpful for adding or removing software, because it considers it as a unit instead of having to work with the individual files.

In the distribution's content (its CD/DVDs) the packages tend to be grouped into categories such as: a) base: essential packages for the system's functioning (tools, start-up programs, system libraries); b) system: administration tools, utility commands; c) development: programming tools: editors, compilers, debuggers... d) graphics: graphics controllers and interfaces, desktops, windows managers... e) other categories.

Normally, to install a package we will need to follow a series of steps:

1) Preliminary steps (pre-installation): check that the required software exists (and with the correct versions) for its functioning (dependencies), whether system libraries or other applications used by the software.

2) Decompress the package content, copying the files to their definitive locations, whether absolute (with a fixed position) or can be relocated to other directories.

3) Post-installation: retouching the necessary files, configuring possible software parameters, adjusting it to the system...

Depending on the types of packages, these steps may be mostly automatic (this is the case in RPM [Bai03] and DEB [Deb02]) or they may all be needed to be done by hand (.tgz case) depending on the tools provided by the distribution.

Next, let's see perhaps the three most classical packages of most distributions. Each distribution has one as standard and supports one of the others.

5.1. TGZ package

Note

TGZ packages are a basic tool when it comes to installing unorganised software. Besides, they are a useful tool for back-up processes and restoring files.

TGZ packages are perhaps those that have been used for longest. The first GNU/Linux distributions used them for installing the software, and several distributions still use it (for example, Slackware) and some commercial UNIX. They are a combination of files joined by the tar command in a single .tar file that has then been compressed using the gzip utility, and that tends to appear with the .tgz or .tar.gz extension. At the same time, nowadays it is common to find tar.bz2 which instead of gzip use another utility called bzip2, which in some cases obtains greater file compression.

Contrary to what it may seem, it is a commonly used format especially by the creators or distributors of software external to the distribution. Many software creators that work for various platforms, such as various commercial UNIX and different distributions of GNU/Linux prefer it as a simpler and more portable system.

An example of this case is the GNU project, which distributes its software in this format (in the form of source code), since it can be used in any UNIX, whether a proprietary system, a BSD variant or a GNU/Linux distribution.

If in binary format, we will have to bear in mind that it is suitable for our system, for example a denomination such as the following one is common (in this case, version 1.4 of the Mozilla web navigator):

```
mozilla-i686-pc-linux-gnu-1.4-installer.tar.gz
```

where we have the package name, as Mozilla, designed for i686 architecture (Pentium II or above or compatible), it could be i386, i586, i686, k6 (amd k6), k7 (amd athlon), amd64 u x86_64 (for AMD64 and some 64bit intels with em64t), o ia64 (intel Itaniums) others for the architectures of other machines such as sparc, powerpc, mips, hppa, alpha... then it tells us that it is for Linux, on a PC machine, software version 1.4.

If it were in source format, it could appear as:

```
mozilla-source-1.4.tar.gz
```

where we are shown the word *source*; in this case it does not mention the machine's architecture version, this tells us that it is ready for compiling on different architectures.

Otherwise, there would be different codes for every operating system or source: GNU/Linux, Solaris, Irix, bsd...

The basic process with these packages consists of:

1) Decompressing the package (they do not tend to use absolute path, meaning that they can be decompressed anywhere):

```
tar –zxvf file.tar.gz (or .tgz file)
```

With the *tar* command we use z options: decompress, x: extract files, v: view process, f: name the file to be treated.

It can also be done separately (without the tar's z):

```
gunzip file.tar.gz
```

(leaves us with a tar file)

```
tar –xvf file.tar
```

2) Once we have decompressed the tgz, we will have the files it contained, normally the software should include some file of the *readme* or *install* type, which specifies the installation options step by step, and also possible software dependencies.

In the first place, we should check the dependencies to see if we have the right software, and if not, look for it and install it.

If it is a binary package, the installation is usually quite easy, since it will be either directly executable from wherever we have left it or it will carry its own installer. Another possibility is that we may have to do it manually, meaning that it will be enough to copy it (cp -r, recursive copy) or to move it (*mv* command) to the desired position.

Another case is the source code format. Then, before installing the software we will first have to do a compilation. For this we will need to read the instruction that the program carries in some detail. But most developers use a GNU system called autoconf (from autoconfiguration), which normally uses the following steps (if no errors appear):

- ./*configure*: it is a script that configures the code so that it can be compiled on our machine and that verifies that the right tools exist. The --prefix = directory option makes it possible to specify where the software will be installed.

- *make*: compilation itself.

- *make install*: installing the software in the right place, normally previously specified as an option to configure or assumed by default.

This is a general process, but it depends on the software whether it follows it or not, there are fairly worse cases where the entire process needs to be carried out by hand, retouching configuration files or the *makefile*, and/or compiling the files one by one, but luckily this is becoming less and less common.

In the case of wanting to delete all of the installed software, we will have to use the uninstaller if provided or, otherwise, directly delete the directory or installed files, looking out for potential dependencies.

The tgz packages are fairly common as a backup mechanism for administration tasks, for example, for saving copies of important data, making backups of user accounts or saving old copies of data that we do not know if we will need again. The following process tends to be used: let's suppose that we want to save a copy of the directory "dir", we can type: `tar -cvf dir.tar dir` (c: compact dir in the file dir.tar) `gzip dir.tar` (compress) or in a single instruction like:

```
tar -cvzf dir.tgz dir
```

The result will be a dir.tgz file. We need to be careful if we are interested in conserving the file attributes and user permissions, as well as possibly links that may exist (we must examine the tar options so that it adjusts to the required backup options).

5.2. Fedora/Red Hat: RPM packages

The RPM packages system [Bai03] created by Red Hat represents a step forward, since it includes the management of software configuration tasks and dependencies. Also, the system stores a small database with the already installed packages, which can be consulted and updated with new installations.

Conventionally, RPM packages use a name such as:

```
package-version-rev.arq.rpm
```

where *package* is the name of the software, *version* is the numbering of the software version, *rev* normally indicates the revision of the RPM package, which indicates the number of times it has been built and *arq* refers to the architecture that it is designed for, whether Intel/AMD (i386, i586, i686, x86_64, em64t, ia64) or others such as alpha, sparc, PPC... The noarch "architecture"is normally used when it is independent, for example, a Set of scripts and src in the case of dealing with source code packages. A typical execution will include running rpm, the options of the operation to be performed, together with one or more names of packages to be processed together.

Note

The package: apache-1.3.19-23.i686.rpm would indicate that it is Apache software (the web server), in its version 1.3.19, package revision RPM 23, for Pentium II architectures or above.

Typical operations with RPM packages include:

- **Package information**: specific information about the package is consulted using the option -q together with the package name (with -p if on an rpm file). If the package has not yet been installed, the option would be -q accompanied by the information option to be requested, and if the request is to be made to all the installed packages at the same time, the option would be -qa. For example, requests from an installed package:

Request	RPM options	Results
Files	`rpm -ql`	List of the files it contains
Information	`rpm -qi`	Package description
Requirements	`rpm -qR`	Prior requirements, libraries or software

- **Installation**: simply `rpm -i package.rpm`, or with the URL where the package can be found, for downloading from FTP or web servers, we just need to use the syntax ftp:// or http:// to obtain the package's location.
  The installation can be completed on condition that the package dependencies are met, whether in the form of prior software or the libraries that should be installed. In the case of not fulfilling this requirement, we will be told what software is missing, and the name of the package that provides it. We can force the installation (although the installed software may not work) with the options `--force` or `--nodeps`, or simply by ignoring the information on the dependencies.
  The task of installing a package (done by `rpm`) entails various sub-tasks: a) checking for potential dependencies; b) examining for conflicts with other previously installed packages; c) performing pre-installation tasks; c) deciding what to do with the configuration files associated to the package if they existed previously; d) unpackaging the files and placing them in the right place; e) performing other post-installation tasks; finally, f) storing the log of tasks done in the RPM database.

- **Updating**: equivalent to the installation but first checking that the software already exists `rpm –U package.rpm`. It will take care of deleting the previous installation.

- **Verification**: during the system's normal functioning many of the installed files will change. In this regard, RPM allows us to check files in order to detect any changes from a normal process or from a potential error that could indicate corrupt data. Through `rpm –V package` we verify a specific package and through `rpm –Va` we will verify all of them.

- **Deletion**: erasing the package from the RPM system (-e or --erase); if there are dependencies, we may need to eliminate others previously.

Example

For a remote case:

rpm -i ftp://site/directory/package.rpm

would allow us to download the package from the provided FTP or web site, with its directory location, and proceed in this case to install the package.

We need to control where the packages come from and only use known and reliable package sources, such as the distribution's own manufacturer or trustworthy sites. Normally, together with the packages, we are offered a digital signature for them, so that we can check their authenticity. The sums md5 are normally used for checking that the package has not been altered and other systems, such as GPG (GNU version of PGP), for checking the authenticity of the package issuer. Similarly, we can find different RPM package stores on Internet, where they are available for different distributions that use or allow the RPM format.

Note

View the site:
www.rpmfind.net.

For a secure use of the packages, official and some third party repositories currently sign the packages electronically, for example, using the abovementioned GPG; this helps us to make sure (if we have the signatures) that the packages come from a reliable source. Normally, every provider (the repository) will include some PGP signature files with the key for its site. From official repositories they are normally already installed, if they come from third parties we will need to obtain the key file and include it in RPM, typically:

```
$ rpm –import GPG–KEY–FILE
```

With GPP-KEY-FILE being the GPG key file or URL of the file, normally this file will also have sum md5 to check its integrity. And we can find the keys in the system with:

```
$ rpm –qa | grep ^gpg–pubkey
```

we can observe more details on the basis of the obtained key:

```
$ rpm -qi gpg-key-xxxxx-yyyyy
```

For a specific RPM package we will be able to check whether it has a signature and with which one it has been used:

```
$ rpm -checksig -v <package>.rpm
```

And to check that a package is correct based on the available signatures, we can use:

```
$ rpm -K <package.rpm>
```

We need to be careful to import just the keys from the sites that we trust. When RPM finds packages with a signature that we do not have on our system or when the package is not signed, it will tell us and, then, we will have to decide on what we do.

Regarding RPM support in the distributions, in Fedora (Red Hat and also in its derivatives), RPM is the default package format and the one used extensively by the distribution for updates and software installation. Debian uses the format called DEB (as we will see), there is support for RPM (the rpm command exists), but only for consulting or package information. If it is essential to install an rpm package in Debian, we advise using the *alien* utility, which can convert package formats, in this case from RPM to DEB, and proceed to install with the converted package.

In addition to the distribution's basic packaging system, nowadays each one tends to support an intermediate higher level software management system, which adds an upper layer to the basic system, helping with software management tasks, and adding a number of utilities to improve control of the process.

In the case of Fedora (Red Hat and derivatives) it uses the YUM system, which allows as a higher level tool to install and manage packages in rpm systems, as well as automatic management of dependencies between packages. It allows access to various different repositories, centralises their configuration in a file (/etc/yum.conf normally), and has a simple commands interface.

Note

YUM in: http://yum.baseurl.org

The yum configuration is based on:

/etc/yum.config	(options file)
/etc/yum	(directory for some associated utilities)
/etc/yum.repos.d	(directory for specifying repositories, a file for each one, including access information and location of the gpg signatures).

A summary of the typical yum operations would be:

Order	Description
yum install <name>	Install the package with the name
yum update <name>	Update an existing package
yum remove <name>	Eliminate package
yum list <name>	Search package by name (name only)
yum search <name>	More extensive search
yum provices <file>	Search for packages that provide the file
yum update	Update the entire system
yum upgrade	As above, including additional packages

Finally, Fedora also offers a couple of graphics utilities for YUM, pup for controlling recently available updates, and pirutas a software management package. There are also others like *yumex*, with greater control of yum's internal configuration.

5.3. Debian: DEB packages

Debian has interactive tools such as *tasksel*, which makes it possible to select sub-sets of packages grouped into types of tasks: packages for X, for development, for documentation etc., or such as *dselect*, which allows us to navigate the entire list of available packages (there are thousands) and select those we wish to install or uninstall. In fact, these are only a front-end of the APT mid-level software manager.

At the command line level it has *dpkg*, which is the lowest level command (would be the equivalent to rpm), for managing the DEB software packages directly [Deb02], typically *dpkg -i package.deb* to perform the installation. All sorts of tasks related to information, installation, removal or making internal changes to the software packages can be performed.

The intermediary level (as in the case of Yum in Fedora) is presented by the APT tools (most are *apt-xxx* commands). APT allows us to manage the packages from a list of current and available packages based on various software sources, whether the installation's own CDs, FTP or web (HTTP) sites. This management is conducted transparently, in such a way that the system is independent from the software sources.

The APT system is configured from the files available in `/etc/apt`, where `/etc/apt/sources.list` is the list of available sources; an example could be:

```
deb http://http.us.debian.org/debian stable main contrib non-
free
```

```
debsrc http://http.us.debian.org/debian stable main contrib
non-free
deb http://security.debian.org stable/updates main contrib
non-free
```

Where various of the "official" sources for a Debian are compiled (*etch* in this case, which is assumed to be stable), from which we can obtain the software packages in addition to their available updates. Basically, we specify the type of source (web/FTP in this case), the site, the version of the distribution (stable in this example) and categories of software to be searched for (free, third party contributions, non-free or commercial licenses).

The software packages are available for the different versions of the Debian distribution, there are packages for the stable, testing, and unstable versions. The use of one or the others determines the type of distribution (after changing the repository sources in sources.list). It is possible to have mixed package sources, but it is not advisable, because conflicts could arise between the versions of the different distributions.

Note

Debian's DEB packages are perhaps the most powerful installation system existing in GNU/Linux. A significant benefit is the system's independence from the sources of the packages (through APT).

Once we have configured the software sources, the main tool for handling them in our system is apt-get, which allows us to install, update or remove from the individual package, until the entire distribution is updated. There is also a front-end to apt-get, called *aptitude*, whose options interface is practically identical (in fact it could be described as an apt-get emulator, since the interface is equivalent); as benefits it manages package dependencies better and allows an interactive interface. In fact it is hoped that aptitude will become the default interface in the command line for package management in Debian.

Some basic functions of apt-get:

- Installation of a particular package:
  ```
  apt-get install package
  ```
- Removing a package:
  ```
  apt-get remove package
  ```
- Updating the list of available packages:
  ```
  apt-get update
  ```
- Updating the distribution, we could carry out the combined steps:
  ```
  apt-get update
  apt-get upgrade
  ```

```
apt-get dist-upgrade
```

Through this last process, we can keep our distribution permanently updated, updating installed packages and verifying dependencies with the new ones. Some useful tools for building this list are `apt-spy`, which tries to search for the fastest official sites, or netselect, which allows us to test a list of sites. On a separate note, we can search the official sites (we can configure these with apt-setup) or copy an available source file. Additional (third party) software may need to add more other sources (to *etc/sources.list*); lists of available source sites can be obtained (for example: http://www.apt-get.org).

Updating a system in particular generates a download of a large number of packages (especially in `unstable`), which makes it advisable to empty the cache, the local repository, with the downloaded packages (they are kept in /var/cache/apt/archive) that will no longer be used, either with `apt-get clean` to eliminate them all or with `apt-get autoclean` to eliminate the packages that are not required because there are already new versions and, in principle, they will no longer be needed. We need to consider whether we may need these packages again for the purposes of reinstalling them, since, if so, we will have to download them again.

The APT system also allows what is known as SecureAPT, which is the secure management of packages through verifying sums (md5) and the signatures of package sources (of the GPG type). If the signatures are not available during the download, apt-get reports this and generates a list of unsigned packages, asking whether they will stop being installed or not, leaving the decision to the administrator. The list of current reliable sources is obtained using:

```
# apt-key list
```

The GPG keys of the official Debian sites are distributed through a package, and we can install them as follows:

```
# apt-get install debian-archive-keyring
```

Obviously, considering that we have the sources.list with the official sites. It is hoped that by default (depending on the version of Debian) these keys will already be installed when the system initiates. For other unofficial sites that do not provide the key in a package, but that we consider trustworthy, we can import their key, obtaining it from the repository (we will have to consult where the key is available, there is no defined standard, although it is usually on the repository's home page). Using apt-key add with the file, to add the key or also:

```
# gpg -import file.key
# gpg -export -armor XXXXXXXX | apt-key add -
```

With X being a hexadecimal related to the key (see repository instructions for the recommended way of importing the key and the necessary data).

Another important functionality of the APT system is for consulting package information, using the apt-cache tool, which allows us to interact with the lists of Debian software packages.

Example

The apt-cache tool has commands that allow us to search for information about the packages, for example:

- Search packages based on an incomplete name:
  `apt-cache search name`
- Show package description:
  `apt-cache show package`
- What packages it depends on:
  `apt-cache depends package`

Other interesting apt tools or functionalities:

- apt-show-versions: tells us what packages may be updated (and for what versions, see option -u).

Other more specific tasks will need to be done with the lowest level tool, such as dpkg. For example, obtaining the list of files of a specific installed package:

```
dpkg -L package
```

The full list of packages with

```
dpkg -l
```

Or searching for what package an element comes from (file for example):

```
dpkg -S file
```

This functions for installed packages; apt-file can also search for packages that are not yet installed.

Finally, some graphic tools for APT, such as synaptic, gnome-apt for gnome, and kpackage or adept for KDE are also worth mentioning, as well as the already mentioned text ones such as aptitude or dselect.

Conclusion: we should highlight that the APT management system (in combination with the dpkg base) is very flexible and powerful when it comes to managing updates and is the package management system used by Debian and its derived distributions such as Ubuntu, Kubuntu, Knoppix, Linex etc.

6. Generic administration tools

In the field of administration, we could also consider some tools, such as those designed generically for administration purposes. Although it is difficult to keep up to date with these tools because of the current plans of distribution with different versions, which evolve very quickly. We will mention a few examples (although at a certain time they may not be completely functional):

a) Linuxconf: this is a generic administration tool that groups together different aspects in a kind of text menu interface, which in the latest versions evolved to web support; it can be used with practically any GNU/Linux distribution and supports various details inherent to each one (unfortunately, it has not been updated for a while).

Note

We can find them in: Linuxconf http://www.solucorp.qc.ca/linuxconf

b) Webmin: this is another administration tool conceived from a web interface; it functions with a series of plug-ins that can be added for each service that needs to be administered; normally it has forms that specify the service configuration parameters; it also offers the possibility (if activated) of allowing remote administration from any machine with a navigator.

c) Others under development like cPanel, ISPConfig.

At the same time, the Gnome and KDE desktop environments tend to include the "Control Panel" concept, which allows management of the graphical interfaces' visual aspect as well as the parameters of some system devices.

With regards to the individual graphics tools for administration, the GNU/Linux distribution offers some directly (tools that accompany both Gnome and KDE), tools dedicated to managing a device (printers, sound, network card etc.), and others for the execution of specific tasks (Internet connection, configuring the start up of system services, configuring X Window, visualising logs...). Many of them are simple front-ends for the system's basic tools, or are adapted to special features of the distribution.

In this section, we should particularly highlight the Fedora distribution (Red Hat and derivatives), which tries to offer several (rather minimalist) utilities for different administration functions, we can find them on the desktop (in the administration menu), or in commands like system-config-xxxxx for different management functionalities for: screen, printer, network, security, users, packages etc. We can see some of them in the figure:

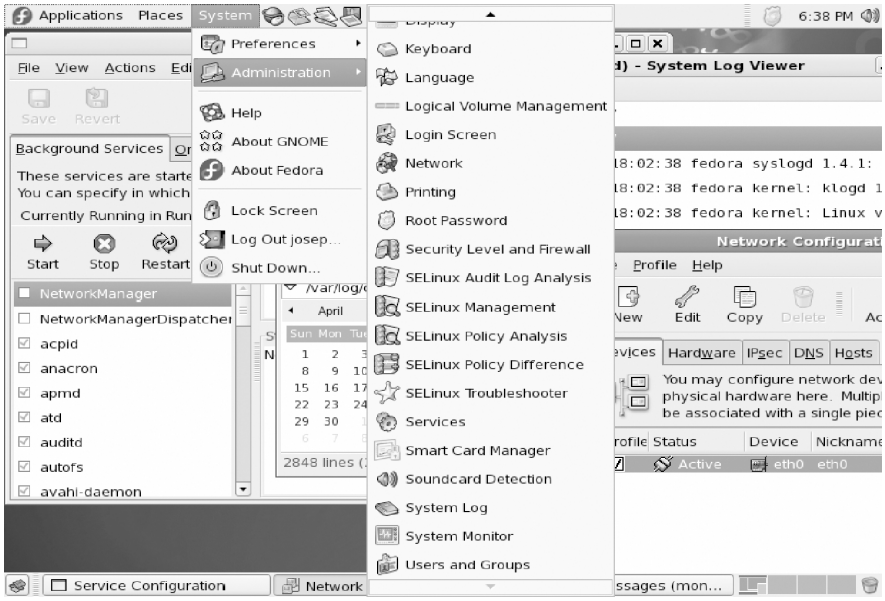

Figure 3. A few of the administration graphics utilities in Fedora

7. Other tools

In this unit's limited space we cannot comment on all the tools that can offer us benefits for administration. We will cite some of the tools that we could consider basic:

- The various basic UNIX commands: grep, awk, sed, find, diff, gzip, bzip2, cut, sort, df, du, cat, more, file, which...

- The editors, essential for any editing task, like: vi, very much used for administration tasks because of the speed of making small changes to the files. Vim is the vi compatible editor, which GNU/Linux tends to carry; it allows a syntax coloured in various languages. Emacs, a very complete editor, adapted to different programming languages (syntax and editing modes); it has a very complete environment and an X version called Xemacs. Joe, editor compatible with Wordstar. And many more...

- Scripting languages, tools for administration, like: Perl, very useful for handling regular expressions and analysing files (filtering, ordering etc.). PHP, language that is very often used in web environments. Python, another language that can make fast prototypes of applications...

- Tool for compiling and debugging high level languages: GNU `gcc` (compiler of C and C++), `gdb` (debugger), `xxgdb` (X interface for gdb), `ddd` (debugger for various languages).

Note

See material associated to the introduction course to GNU/Linux, the man pages of the commands or a tools reference such as [Stu01].

Activities

1) For a fast reading, see the FHS standard, which will help us to have a good guide for searching for files in our distribution.

2) To revise and broaden concepts and programming of shell scripts in bash, see: [Bas] [Coo].

3) For RPM packages, how would we do some of the following tasks?:
- Find out what package installed a specific command.
- Obtain a description of the package that installed a command.
- Erase a package whose full name we don't know.

Show all the files that were in the same package as a specific file.

4) Perform the same tasks as above, but for Debian packages, using APT tools.

5) Update a Debian (or Fedora) distribution.

6) Install a generic administration tool, such as Linuxconf or Webadmin for example, on our distribution. What do they offer us? Do we understand the executed tasks, and the effects they cause?

Bibliography

Other sources of reference and information

[Bas][Coo] offer a broad introduction (and advanced concepts) of programming shell scripts in bash, as well as several examples. [Qui01] discusses the different programming shells in GNU/Linux, as well as their similarities and differences.

[Deb02][Bai03] offer a broad vision of the software package systems of the Debian and Fedora/Red Hat distributions.

[Stu] is a wide introduction to the tools available in GNU/Linux.