



GNU/LINUX ADVANCED ADMINISTRATION

AUTHOR:
A. SUPPI BOLDRITO

COORDINATOR:
J. JORBA ESTEVE



FREE
TECHNOLOGY
ACADEMY



GNU/Linux advanced administration

Josep Jorba Esteve (coordinador)
Remo Suppi Boldrito

XP07/M2103/02279

Josep Jorba Esteve

Senior engineer and PhD in IT of the Universidad Aut3noma 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
© Josep Jorba Esteve, Remo Suppi Boldrito
All rights are reserved
© of this edition, FUOC, 2009
Av. Tibidabo, 39-43, 08035 Barcelona
Design: Manel Andreu
Publishing: Eureka Media, SL

Copyright © 2019, FUOC. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License"

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 1

Introduction to the GNU/Linux operating system

Josep Jorba Esteve

1. Free Software and Open Source
2. UNIX. A bit of history
3. GNU/Linux systems
4. The profile of the systems administrator
5. Tasks of the administrator
6. GNU/Linux distributions
7. What we will look at...

Module 2

Migration and coexistence with non-Linux systems

Josep Jorba Esteve

1. Computer systems: environments
2. GNU/Linux services
3. Types of use
4. Migration or coexistence
5. Migration workshop: case study analysis

Module 3

Basic tools for the administrator

Josep Jorba Esteve

1. Graphics tools and command line
2. Standards
3. System documentation
4. Shell scripting
5. Package management tools
6. Generic administration tools
7. Other tools

Module 4

The kernel

Josep Jorba Esteve

1. The kernel of the GNU/Linux system
2. Configuring or updating the kernel
3. Configuration and compilation process
4. Patching the kernel
5. Kernel modules
6. Future of the kernel and alternatives
7. Tutorial: : configuring de kernel to the requirements of the user

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

Index

Introduction.....	5
1. Free Software and Open Source.....	7
2. UNIX. A bit of history.....	13
3. GNU/Linux systems.....	21
4. The profile of the systems administrator.....	25
5. Tasks of the administrator.....	30
6. GNU/Linux distributions.....	35
6.1. Debian	39
6.2. Fedora Core	42
7. What we will look at.....	47
Activities.....	51
Bibliography.....	52

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 Cathedral 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

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.

Note

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

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).

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.

Note

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

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.

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.

Note

Illegal domestic copies are also sometimes known as pirated copies.

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.

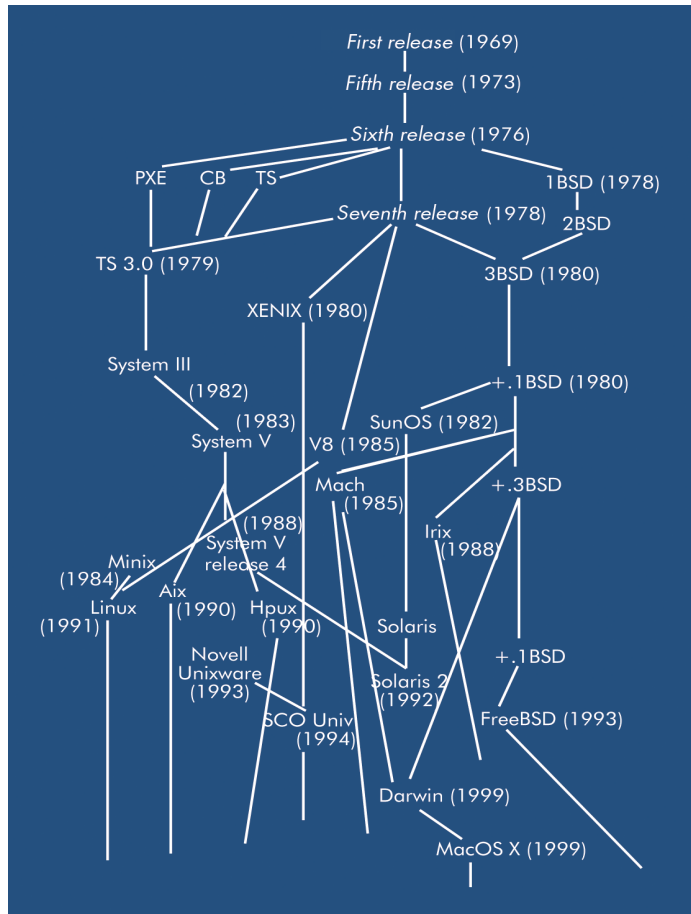


Figure 1. Historical summary of the different versions of UNIX

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:

- **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 HP-UX 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

Note

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

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.

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.

Note

Original Mach project:
[http://www.cs.cmu.edu/afs/
cs/project/mach/public/www/
mach.html](http://www.cs.cmu.edu/afs/cs/project/mach/public/www/mach.html)

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 engineering 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.

- 1)** 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 tuning:** 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 files (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, Enlightenment, 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.

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.

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-

Note

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



Figure 2

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. Linux 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. Linux 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.

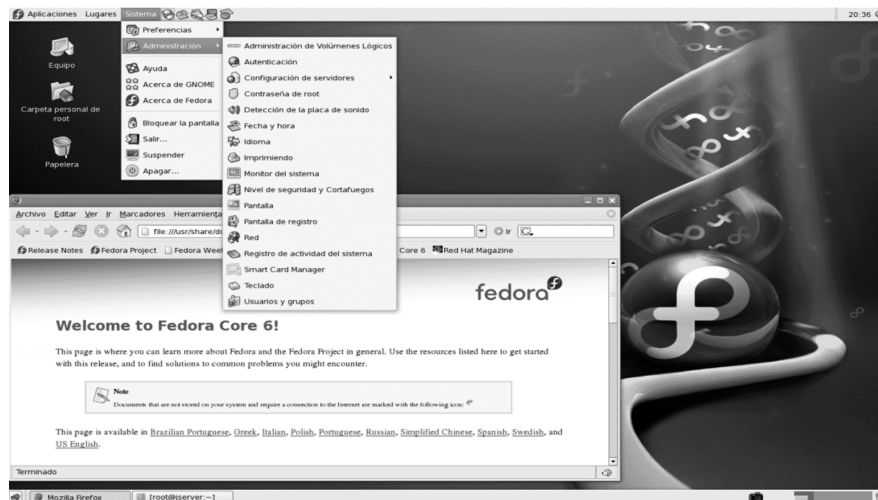


Figure 5. Fedora Core desktop with GNOME

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.
- 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.

Note

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

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 administering 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.....	5
1. Computer systems: environments.....	7
2. GNU/Linux services.....	11
3. Types of use.....	13
4. Migration or coexistence.....	16
4.1. Identify service requirements	17
4.2. Migration process	18
5. Migration workshop: case study analysis.....	24
5.1. Individual migration of a Windows desktop user to a GNU/Linux system	24
5.2. Migration of a small organisation with Windows systems and a few UNIX	27
5.3. Migration of a standalone Windows server to a Samba server running GNU/Linux	29
Activities.....	35
Bibliography.....	36

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 Gnome-Office).
- 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/Gnome-Office>

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, webservice 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/Net-Beui (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).

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,

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.html>

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.

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

Note

Linux Hardware Howto: <http://www.tldp.org/HOWTO/HardwareHOWTO/index.html>

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.

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 Microsystems'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

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.html>

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 `smbxxx` 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



Universitat Oberta
de Catalunya

www.uoc.edu

Index

Introduction.....	5
1. Graphics tools and command line.....	7
2. Standards.....	9
3. System documentation.....	12
4. Shell scripting.....	14
4.1. Interactive <i>shells</i>	15
4.2. Shells	18
4.3. System variables	21
4.4. Programming scripts in Bash	22
4.4.1. Variables in Bash	23
4.4.2. Comparisons	24
4.4.3. Control structures	24
5. Package management tools.....	27
5.1. TGZ package	28
5.2. Fedora/Red Hat: RPM packages	30
5.3. Debian: DEB packages	34
6. Generic administration tools.....	38
7. Other tools.....	40
Activities.....	41
Bibliography.....	42

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/l_s, 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](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 password. 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.

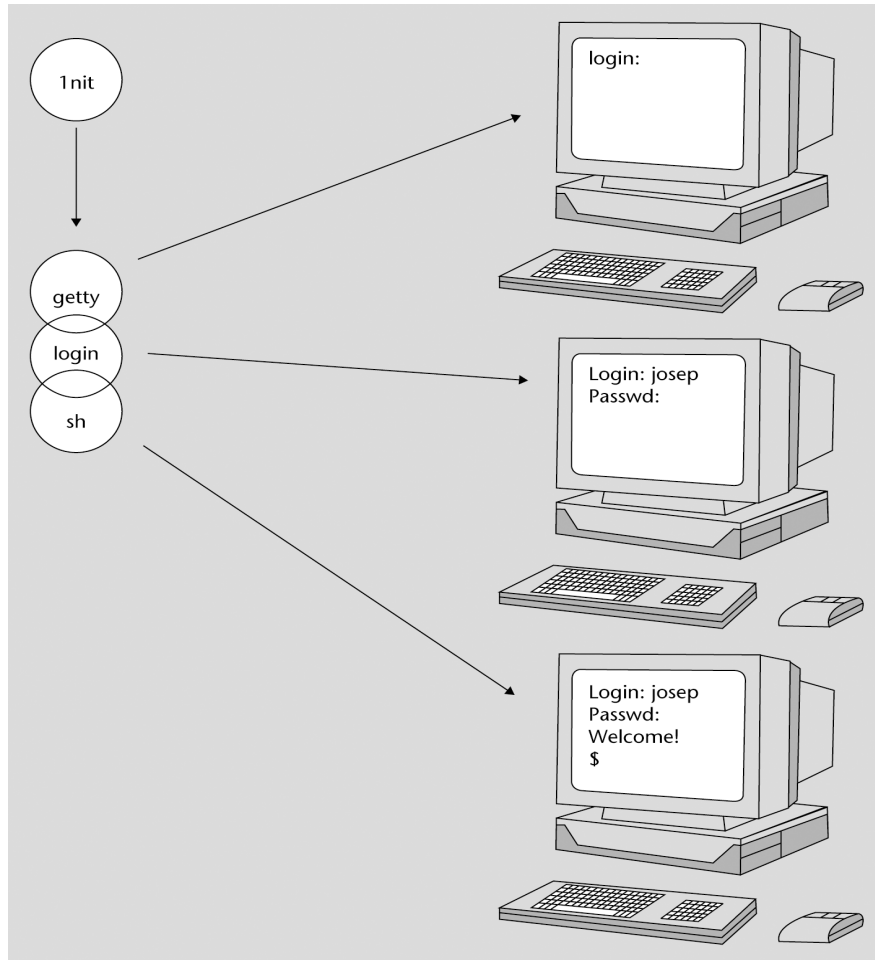


Figure 1. Example of starting up a text shell textual and the system processes involved [Oke]

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.

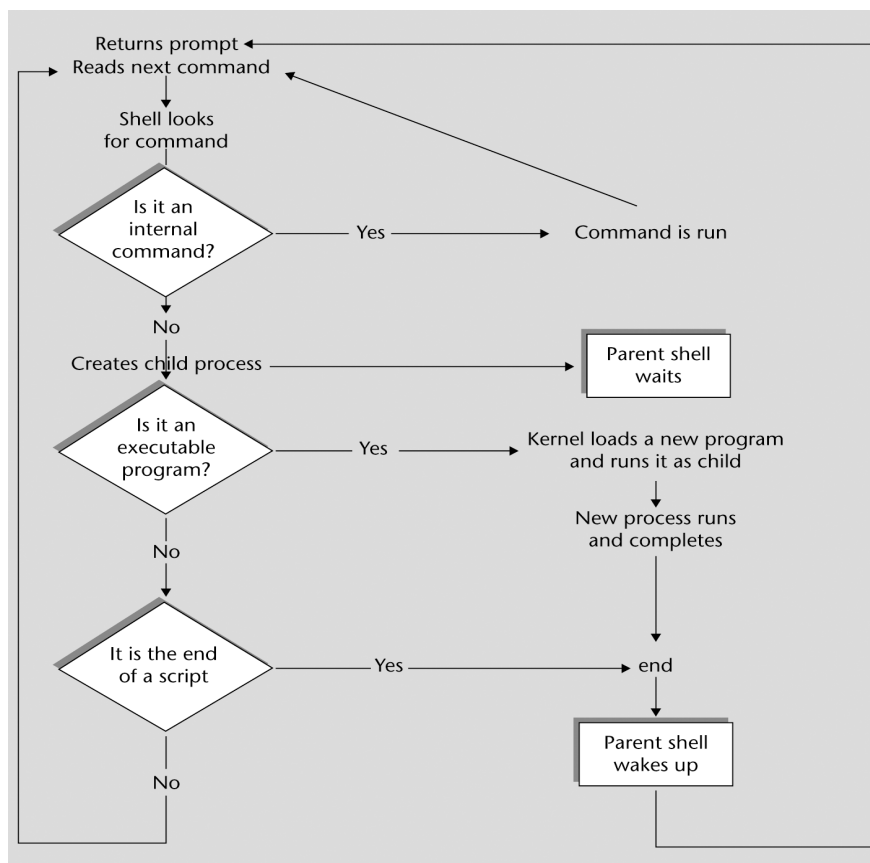


Figure 2. Basic shell flow control

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*, *cs**h* or *ks**h*. 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, *cs**h* and *ks**h*.
- b) Start-up files: *sh*, *ks**h* have *.profile* (in the user account, and is executed in the user's login) and *ks**h* also tends to have a *.kshrc* which is executed next, *cs**h* 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 <i>login</i>
PATH	/bin:/usr/local/bin:/usr/X11/bin	Paths
SHELL	/bin/bash	User <i>shell</i>
PS1	\$	<i>Shell</i> 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 [ expression ]
then
  commands
```

```
fi
```

b) Structure *if...then...else*, evaluates the expression and if a certain value is obtained then the commands1 are executed, otherwise commands2 are executed:

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

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

```
if [ expression ]
then
    commands
elif [ expression2 ]
then
    commands
else
    commands
fi
```

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

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

Note

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

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 [ expression ]
```

```
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

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.

Note

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

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.arch.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	<code>rpm -ql</code>	List of the files it contains
Information	<code>rpm -qi</code>	Package description
Requirements	<code>rpm -qR</code>	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 GPG-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
<code>yum install <name></code>	Install the package with the name
<code>yum update <name></code>	Update an existing package
<code>yum remove <name></code>	Eliminate package
<code>yum list <name></code>	Search package by name (name only)
<code>yum search <name></code>	More extensive search
<code>yum provides <file></code>	Search for packages that provide the file
<code>yum update</code>	Update the entire system
<code>yum upgrade</code>	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:

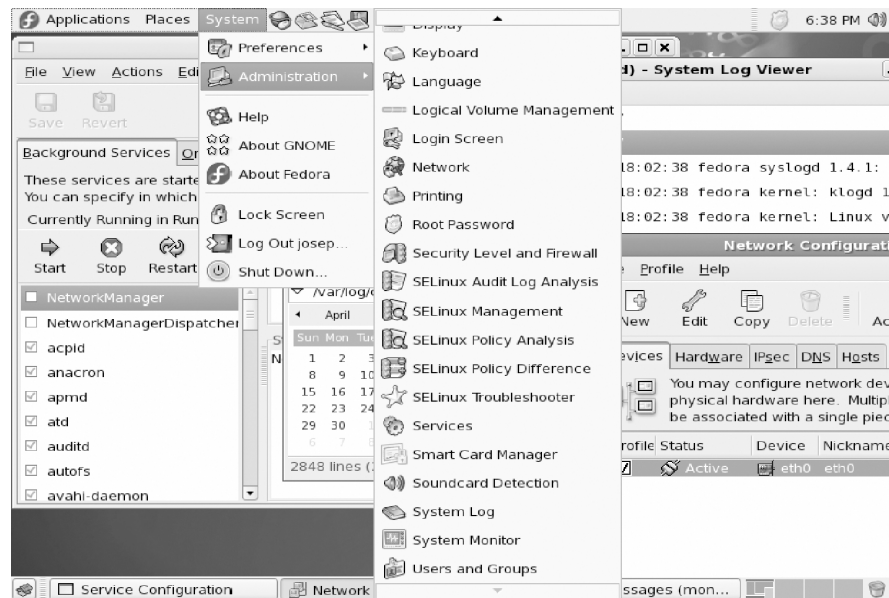


Figure 3. A few of the administration graphics utilities in Fedora

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.

Linux kernel

Josep Jorba Esteve

PID_00148468



Universitat Oberta
de Catalunya

www.uoc.edu

Index

Introduction.....	5
1. The kernel of the GNU/Linux system.....	7
2. Configuring or updating the kernel.....	15
3. Configuration and compilation process.....	18
3.1. Kernel compilation versions 2.4.x	19
3.2. Migration to kernel 2.6.x	24
3.3. Compilation of the kernel versions 2.6.x	26
3.4. Compilation of the kernel in Debian (Debian way)	27
4. Patching the kernel.....	30
5. Kernel modules.....	32
6. Future of the kernel and alternatives.....	34
7. Tutorial: : configuring de kernel to the requirements of the user.....	38
7.1. Configuring the kernel in Debian	38
7.2. Configuring the kernel in Fedora/Red Hat	40
7.3. Configuring a generic kernel	42
Activities.....	45
Bibliography.....	46

Introduction

The kernel of the GNU/Linux system (which is normally called Linux) [Vasb] is the heart of the system: it is responsible for booting the system, for managing the machine's resources by managing the memory, file system, input/output, processes and intercommunication of processes.

Its 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 core that worked together with the GNU project software and that he was offering it to the community of developers for testing and suggesting improvements for making it more usable. This was the origin of the operating system's kernel that would later come to be known as Linux.

One of the particular features of Linux is that following the Free Software philosophy, it offers the source code of the operating system itself (of the kernel), in a way that makes it a perfect tool for teaching about operating systems.

Another main advantage, is that by having the source code, we can compile it to adapt it better to our system and we can also configure its parameters to improve the system's performance.

In this unit, we will look at how to handle this process of preparing a kernel for our system. How, starting with the source code, we can obtain a new version of the kernel adapted to our system. Similarly, we will discuss how to develop the configuration and subsequent compilation and how to test the new kernel we have obtained.

Note

The Linux kernel dates back to 1991, when Linus Torvalds made it available to the community. It is one of the few operating systems that while extensively used, also makes its source code available.

1. The kernel of the GNU/Linux system

The core or kernel is the basic part of any operating system [Tan87], where the code of the fundamental services for controlling the entire system lie. Basically, its structure can be divided into a series of management components designed to:

- Manage processes: what tasks will be run, in what order and with what priority. An important aspect is the scheduling of the CPU: how do we optimise the CPU's time to run the tasks with the best possible performance or interactivity with users?
- Intercommunication of processes and synchronisation: how do tasks communicate with each other, with what different mechanisms and how can groups of tasks be synchronised?
- Input/output management (I/O): control of peripherals and management of associated resources.
- Memory management: optimising use of the memory, paginating service, and virtual memory.
- File management: how the system controls and organises the files present in the system and access to them.

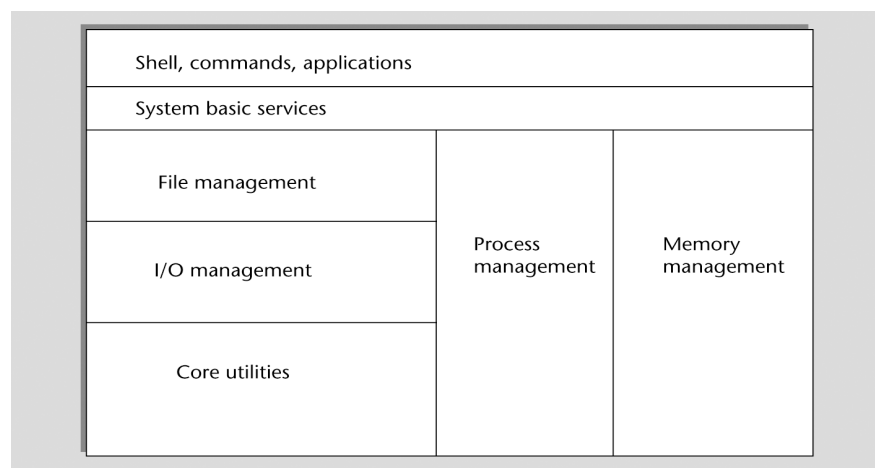


Figure 1. Basic functions of a kernel with regards to executed applications and commands

In proprietary systems, the kernel is perfectly "hidden" below the layers of the operating system's software; the end user does not have a clear perspective of what the kernel is and has no possibility of changing it or optimising it, other than through the use of esoteric editors of internal "registers" or specialised

third party programs, which are normally very expensive. Besides, the kernel is normally unique, it is the one the manufacturer provides and the manufacturer reserves the right to introduce any changes it wants whenever it wants and to handle the errors that appear in non-stipulated periods through updates offered to us in the form of error "patches" (or service packs).

One of the main problems of this approach is precisely the availability of these patches, having the error updates on time is crucial and if they are security-related, even more so, because until they are corrected we cannot guarantee the system's security for known problems. Many organisations, large companies, governments, scientific and military institutions cannot depend on the whims of a manufacturer to solve the problems with their critical applications.

The Linux kernel offers an open source solution with the ensuing permissions for modifying, correcting, generating new versions and updates very quickly by anyone anywhere with the required knowledge for doing so.

This allows critical users to control their applications and the system itself better, and offers the possibility of mounting systems with a "tailor-made" operating system adjusted to each individual's taste and in turn to have an open source operating system developed by a community of programmers who coordinate via the Internet, accessible for educational purposes because it has open source code and abundant documentation, for the final production of GNU/Linux systems adapted to individual needs or to the needs of a specific group.

Because the source code is open, improvements and solutions can be found immediately, unlike proprietary software, where we have to wait for the manufacturer's updates. Also, we can personalise the kernel as much as we wish, an essential requirement, for example, in high performance applications, applications that are critical in time or solutions with embedded systems (such as mobile devices).

Following a bit of (quick) history of the kernel [Kera] [Kerb]: it was initially developed by a Finnish student called Linus Torvalds, in 1991, with the intention of creating a similar version to Minix [Tan87] (version for PC of UNIX [Bac86]) for the Intel 386 processor. The first officially published version was Linux 1.0 in March 1994, which only included the execution for the i386 architecture and supported single-processor machines. Linux 1.2 was published in March 1995, and was the first version to cover different architectures such as Alpha, SPARC and Mips. Linux 2.0, in June 1996, added more architectures and was the first version to include multiprocessor support (SMP) [Tum]. In Linux 2.2, January 1999, SMP benefits were significantly increased, and controllers were added for a large amount of hardware. In 2.4, released in January 2001, SMP support was improved, new supported architectures were incorporated and controllers for USB, PC card devices were included (PCMCIA for laptops) part of PnP (plug and play), RAID and volumes support etc. Branch 2.6

of the kernel (December 2003), considerably improved SMP support, offered a better response of the CPU scheduling system, use of threads in the kernel, better support for 64-bit architectures, virtualisation support and improved adaptation to mobile devices.

Where the development is concerned, since the kernel was created by Linus Torvalds in 1991 (version 0.01), he has continued to maintain it, but as his work allowed it and as the kernel matured (and grew) he was helped to maintain the different stable versions of the kernel by different collaborators, while Linus continued (insofar as possible) developing and compiling contributions for the latest version of the kernel's development. The main collaborators of these versions have been [lkm]:

- 2.0 David Weinehall.
- 2.2 Alan Cox (who also develops and publishes patches for most versions).
- 2.4 Marcelo Tosatti.
- 2.6 Andrew Morton / Linus Torvalds.

In order to understand a bit about the complexity of the Linux kernel, let's look at a table with a bit of a summarised history of its different versions and its size in relation to the source code. The table only shows the production versions; the (approximate) size is specified in thousands of lines (K) of source code:

Version	Publication date	Code lines (thousands)
0.01	09-1991	10
1.0	03-1994	176
1.20	03-1995	311
2.0	06-1996	649
2.2	01-1999	1800
2.4	01-2001	3378
2.6	12-2003	5930

As we can see, we have moved from about ten thousand lines to six million.

Now, development of branch 2.6.x of the kernel continues, the latest stable version, which most distributions include as the default version (although some still include 2.4.x, but 2.6.x is an option during the installation); although a certain amount of knowledge about the preceding versions is essential, because we can easily find machines with old distributions that have not been updated, which we may have to maintained or migrated to more modern versions.

Note

The kernel has its origins in the MINIX system, a development by Andrew Tanenbaum, as a UNIX clone for PC.

Note

Today's kernel has reached a significant degree of complexity and maturity.

During the development of branch 2.6, the works on the kernel accelerated considerably, because both Linus Torvalds, and Andrew Morton (who maintain Linux 2.6) joined (in 2003) OSDL (Open Source Development Laboratory) [OSDa], a consortium of companies dedicated to promoting the use of Open Source and GNU/Linux by companies (the consortium includes among many other companies with interests in GNU/Linux: HP, IBM, Sun, Intel, Fujitsu, Hitachi, Toshiba, Red Hat, Suse, Transmeta...). Now we are coming across an interesting situation, since the OSDL consortium sponsored the works of both the stable version of the kernel's maintainer (Andrew) and developer (Linus), working full time on the versions and on related issues. Linus remains independent, working on the kernel, while Andrew went to work for Google, where he continued his developments full time, making patches with different contributions to the kernel. After some time, OSDL became The Linux Foundation.

Note

The Linux Foundation:
www.linuxfoundation.org

We need to bear in mind that with current versions of the kernel, a high degree of development and maturity has been achieved, which means that the time between the publication of versions is longer (this is not the case with partial revisions).

Another factor to consider is the number of people that are currently working on its development. Initially, there were just a handful of people with complete knowledge of the entire kernel, whereas nowadays many people are involved in its development. Estimates are almost two thousand with different levels of contribution, although the number of developers working on the hard core is estimated at several dozen.

We should also take into consideration that most only have partial knowledge of the kernel and neither do they all work simultaneously nor is their contribution equally relevant (some just correct simple errors); it is just a few people (such as the maintainers who have full knowledge of the kernel. This means that developments can take a while to occur, contributions need to be debugged to make sure that they do not come into conflict with each other and choices need to be made between alternative features.

Regarding the numbering of the Linux kernel's versions ([lkm][DBo]), we should bear in mind the following:

a) Until kernel branch 2.6.x, the versions of the Linux kernel were governed by a division into two series: one was known as the "experimental" version (with the second number being an odd number, such as 1.3.xx, 2.1.x or 2.5.x) and the other was the "production" version (even series, such as 1.2.xx, 2.0.xx, 2.2.x, 2.4.x and more). The experimental series were versions that moved rapidly and that were used for testing new features, algorithms, device drivers etc. Because of the nature of the exper-

imental kernels, they could behave unpredictably, losing data, blocking the machine etc. Therefore, they were not suited to production environments, unless for testing a specific feature (with the associated dangers).

Production or stable kernels (even series) were kernels with a well defined set of features, a low number of known errors and with tried and tested device controllers. They were published less frequently than the experimental versions and there were a variety of versions, some better than others. GNU/Linux distributions are usually based on a specifically chosen stable kernel, not necessarily the latest published production kernel.

b) The current Linux kernel numbering (used in branch 2.6.x), continues to maintain some basic aspects: 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 odd 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 do not tend to include the latest version of the kernel, but rather the one they have tested most frequently and can verify is stable for the software and components it includes. On the basis of this classical numbering scheme (followed during versions 2.4.x, until the early versions of branch 2.6), modifications were made to adapt to the fact that the kernel (branch 2.6.x) is becoming more stable (fixing $X.Y$ to 2.6), and that there are fewer and fewer revisions (thus the leap in version of the first numbers), but development remains continuous and frenetic.

Under the latest schemes, four 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), such as *-rc* suffixes (*release candidate*), *-mm* which refers to experimental kernels with tests for different innovative techniques, or *-git* which are a sort of daily snapshot of the kernel's development. These numbering schemes are constantly changing to adapt to the way of working of the kernel community and its needs to accelerate the development.

c) To obtain the latest published kernel, you need to visit the Linux kernels file (at <http://www.kernel.org>) or its local mirror in Spain (<http://www.es.kernel.org>). It will also be possible to find some patches for the original kernel, which correct errors detected after the kernel's publication.

Note

Kernel repository:
www.kernel.org

Some of the technical characteristics ([DBo][Arc]) of the Linux kernel that we should highlight are:

- Kernel of the monolithic type: basically it is a program created as a unit, but conceptually divided into several logical components.
- It has support for loading/downloading portions of the kernel, these portions are known as modules, and tend to be characteristics of the kernel or device drivers.
- Kernel threading: for internal functioning, several execution threads are used internal to the kernel, which may be associated to a user program or to an internal functionality of the kernel. In Linux, this concept was not used intensively. The revisions of branch 2.6.x offered better support and a large proportion of the kernel is run using these various execution threads.
- Multithreaded applications support: user applications support of the multithread, since many computing paradigms of the client/server type, need servers capable of attending to numerous simultaneous requests, dedicating an execution thread to each request or group of requests. Linux has its own library of threads that can be used for multithread applications, with the improvements made to the kernel, they have also allowed a better use for implementing thread libraries for developing applications.
- The kernel is of a nonpreemptive type: this means that within the kernel, system calls (in supervisory mode) cannot be interrupted while the system task is being resolved and, when the latter finishes, the execution of the previous task is resumed. Therefore, the kernel within a call cannot be interrupted to attend to another task. Normally, preemptive kernels are associated to systems that operate in real time, where the above needs to be allowed in order to handle critical events. There are some special versions of the Linux kernel for real time, that allow this by introducing some fixed points where they can be exchanged. This concept has also been especially improved in branch 2.6.x of the kernel, in some cases allowing some resumable kernel tasks to be interrupted in order to deal with others and resuming them later. This concept of a preemptive kernel can also be useful for improving interactive tasks, since if costly calls are made to the system, they can cause delays in interactive applications.
- Multiprocessor support, known as symmetrical multiprocessing (SMP). This concept tends to encompass machines that incorporate the simple case of 2 up to 64 CPUs. This issue has become particularly relevant with multicore architectures, that allow from 2 or 4 to more CPU cores in machines accessible to domestic users. Linux can use multiple processors, where each processor can handle one or more tasks. But some parts of the kernel decreased performance, since they were designed for a single CPU and forced the entire system to stop

under certain cases of blockage. SMP is one of the most studied techniques in the Linux kernel community and important improvements have been achieved in branch 2.6. Since SMP performance is a determining factor when it comes to companies adopting Linux as an operating system for servers.

- File systems: the kernel has a good file system architecture, internal work is based on an abstraction of a virtual system (VFS, *virtual file system*), which can be easily adapted to any real system. As a result, Linux is perhaps the operating system that supports the largest number of file systems, from ext2, to MSDOS, VFAT, NTFS, journaled systems, such as ext3, ReiserFS, JFS(IBM), XFS(Silicon), NTFS, ISO9660 (CD), UDF and more added in the different revisions.

Other less technical characteristics (a bit of marketing):

- a) Linux is free: together with the GNU software and included in any distribution, we can have a full UNIX-like system practically for the cost of the hardware, regarding GNU/Linux distribution costs, we can have it practically free. Although it makes sense to pay a bit extra for a complete distribution, with the full set of manuals and technical support, at a lower cost than would be paid for some proprietary systems or to contribute with the purchase to the development of distributions that we prefer or that we find more practical.
- b) Linux can be modified: the GPL license allows us to read and to modify the source code of the kernel (on condition that we have the required know-how).
- c) Linux can run on fairly limited old hardware; for example, it is possible to create a network server on a 386 with 4 MB of RAM (there are distributions specialised for limited resources).
- d) Linux is a powerful system: the main objective of Linux is efficiency, it aims to make the most of the available hardware.
- e) High quality: GNU/Linux systems are very stable, have a low fault ratio and reduce the time needed for maintaining the systems.
- f) The kernel is fairly small and compact: it is possible to place it, together with some basic programs, on a disk of just 1.44 MB (there are several distributions on just one diskette with basic programs).

g) Linux is compatible with a large number of operating systems, it can read the files of practically any file system and can communicate by network to offer/receive services from any of these systems. Also, with certain libraries it can also run the programs of other systems (such as MSDOS, Windows, BSD, Xenix etc.) on the x86 architecture.

h) Linux has extensive support: there is no other system that has the same speed and number of patches and updates as Linux, not even any proprietary system. For a specific problem, there is an infinite number of mail lists and forums that can help to solve any problem within just a few hours. The only problem affects recent hardware controllers, which many manufacturers are still reluctant to provide if they are not for proprietary systems. But this is gradually changing and many of the most important manufacturers in sectors such as video cards (NVIDIA, ATI) and printers (Epson, HP,) are already starting to provide the controllers for their devices.

2. Configuring or updating the kernel

As GNU/Linux users or system administrators, we need to bear in mind the possibilities the kernel offers us for adapting our requirements and equipment.

At installation time, GNU/Linux distributions provide a series of preconfigured and compiled binary Linux kernels and we will usually have to choose which kernel from the available set best adapts to our hardware. There are generic kernels, oriented at IDE devices, others at SCSI, others that offer a mix of device controllers [AR01] etc.

Another option during the installation is the kernel version. Distributions normally use an installation that they consider sufficiently tested and stable so that it does not cause any problems for its users. For example, nowadays many distributions come with versions 2.6.x of the kernel by default, since it is considered the most stable version (at the time the distribution was released). In certain cases, as an alternative, more modern versions may be offered during the installation, with improved support for more modern (latest generation) devices that perhaps had not been so extensively tested at the time when the distribution was published.

Distributors tend to modify the kernel to improve their distribution's behaviour or to correct errors detected in the kernel during tests. Another fairly common technique with commercial distributions is to disable problematic features that can cause errors for users or that require a specific machine configuration or when a specific feature is not considered sufficiently stable to be included enabled by default.

Note

The possibility of updating and adapting the kernel offers a good adjustment to any system through tuning and optimisation.

This leads us to consider that no matter how well a distributor does the job of adapting the kernel to its distribution, we can always encounter a number of problems:

- The kernel is not updated to the latest available stable version; some modern devices are not supported.
- The standard kernel does not support the devices we have because they have not been enabled.
- The controllers a manufacturer offers us require a new version of the kernel or modifications.
- The opposite, the kernel is too modern, and we have old hardware that is no longer supported by the modern kernels.
- The kernel, as it stands, does not obtain the best performance from our devices.

- Some of the applications that we want to use require the support of a new kernel or one of its features.
- We want to be on the leading edge, we risk installing the latest versions of the Linux kernel.
- We like to investigate or to test the new advances in the kernel or would like to touch or modify the kernel.
- We want to program a driver for an unsupported device.
- ...

For these and other reasons we may not be happy with the kernel we have; in which case we have two possibilities: updating the distribution's binary kernel or tailoring it using the source.

Let's look at a few issues related to the different options and what they entail:

1) Updating the distribution's kernel: the distributor normally also publishes kernel updates as they are released. When the Linux community creates a new version of the kernel, every distributor joins it to its distribution and conducts the relevant tests. Following the test period, potential errors are identified, corrected and the relevant update of the kernel is made in relation to the one offered on the distribution's CDs. Users can download the new revision of the distribution from the website, or update it via some other automatic package system through a package repository. Normally, the system's version is verified, the new kernel is downloaded and the required changes are made so that the following time the system functions with the new kernel, maintaining the old version in case there are any problems.

This type of update simplifies the process for us a lot, but may not solve our problems, since our hardware may not yet be supported or the feature of the kernel to be tested is still not in the version that we have of the distribution; we need to remember that there is no reason for distributor to use the latest available version (for example in kernel.org) but rather the one it considers stable for its distribution.

If our hardware is not enabled by default in the new version either, we will find ourselves in the same situation. Or simply, if we want the latest version, this process is no use.

2) Tailoring the kernel (this process is described in detail in the following sections). In this case, we will go to the sources of the kernel and "manually" adjust the hardware or required characteristics. We will pass through a process of configuring and compiling the source code of the kernel so as to create a binary kernel that we will install on the system and thus have it available the following time the system is booted.

Here we may also encounter two more options, either by default we will obtain the "official" version of the kernel (kernel.org), or we can go to the sources provided by the distribution itself. We need to bear in mind that distributions like Debian and Fedora do a lot of work on adapting the kernel and correcting kernel errors that affect their distribution, which means that in some cases we may have additional corrections to the kernel's original code. Once again, the sources offered by the distribution do not necessarily have to correspond to the latest published version.

This system allows us maximum reliability and control, but at a high administration cost; since we will need to have extensive knowledge of the devices and characteristics that we are selecting (what they mean and what implications they may have), in addition to the consequences that the decisions we make may imply.

3. Configuration and compilation process

Configuring the kernel [Vasb] is a costly process and requires extensive knowledge on the part of the person doing it, it is also one of the critical tasks on which the system's stability depends, given the nature of the kernel, which is the system's central component.

Any error in the procedure can cause instability or the loss of the system. Therefore, it is advisable to make a backup of user data, configurations we have tailored, or, if we have the required devices, to make a complete system backup. It is also advisable to have a start up diskette (or Live CD distribution with tools) to help us in the event of any problem, or a rescue disk which most distributions allow us to create from the distribution's CDs (or by directly providing a rescue CD for the distribution).

Without meaning to exaggerate, if the steps are followed correctly, we know what we are doing and take the necessary precautions, errors almost never occur.

Let's look at the process required to install and configure a Linux kernel. In the following sections, we look at:

- 1) The case of old 2.4.x versions.
- 2) Some considerations regarding migrating to 2.6.x
- 3) Specific details regarding versions 2.6.x.
- 4) A particular case with the Debian distribution, which has its own more flexible compilation system (*debian way*).

Versions 2.4.x are practically no longer offered by current distributions, but we should consider that on more than one occasion we may find ourselves obliged to migrate a specific system to new versions or to maintain it on the old ones, due to incompatibilities or the existence of old unsupported hardware.

The general concepts of the compilation and configuration process will be explained in the first section (2.4.x), since most of them are generic, and we will subsequently see the differences with regard to the new versions.

Note

The process of obtaining a new personalised kernel involves obtaining the sources, adapting the configuration, and compiling and installing the obtained kernel on the system.

3.1. Kernel compilation versions 2.4.x

The instructions are specifically for the Intel x86 architecture, by root user (although part of the process can be done as a normal user):

1) Obtaining the kernel: for example, we can visit www.kernel.org (or its FTP server) and download the version we would like to test. There are mirrors for different countries. In most GNU/Linux distributions, such as Fedora/Red Hat or Debian, the kernel's source code is also offered as a package (normally with some modifications included), if we are dealing with the version of the kernel that we need, it may be preferable to use these (through the kernel-source packages or similar). If we want the latest kernels, perhaps they are not available in the distribution and we will have to go to kernel.org.

2) Unpack the kernel: the sources of the kernel were usually placed and unpacked from the directory `/usr/src`, although we advise using a separate directory so as not to mix with source files that the distribution may carry. For example, if the sources come in a compressed file of the bzip2 type:

```
bzip2 -dc linux-2.4.0.tar.bz2 | tar xvf -
```

If the sources come in a gz file, we will replace bzip2 with gzip. When we decompress the sources, we will have generated a directory `linux-version_kernel` that we will enter in order to configure the kernel.

Before taking the steps prior to compilation, we should make sure that we have the right tools, especially the gcc compiler, make and other complementary gnu utilities for the process. For example, the *modutils*, the different utilities for using and handling the dynamic kernel modules. Likewise, for the different configuration options we should take into account a number of pre-requirements in the form of libraries associated to the configuration interface used (for example ncurses for the menuconfig interface).

In general, we advise checking the kernel documentation (whether via the package or in the root directory of the sources) to know what pre-requirements and versions of the kernel source will be needed for the process. We advise studying the README files in this root directory of the kernel source, and *Documentation/Changes* or the documentation index of the kernel in *Documentation/00-INDEX*.

If we have made previous compilations in the same directory, we need to make sure that the directory we use is clear of previous compilations; we can clear it using *make mrproper* (from the "root" directory).

For the process of configuring the kernel [Vasb], we have several alternative methods, which offer us different interfaces for adjusting the various parameters of the kernel (which tend to be stored in a configuration file, normally *.config* in the "root" directory of the sources). The different alternatives are:

- **make config**: from the command line we are asked for each option, and we are asked for confirmation (y/n) – yes or no, the option, or we are asked for the required values. Or the long configuration, where we are asked for many answers, and depending on each version, we will likewise have to answer almost a hundred questions (or more depending on the version).
- **make oldconfig**: it is useful if we want to reuse an already used configuration (normally stored in a *.config* file, in the root directory of the sources), we need to take into account that it is only valid if we are compiling the same version of the kernel, since different kernel versions can have variable options.
- **make menuconfig**: configuration based on text menus, fairly convenient; we can enable or disable what we want and it is faster than *make config*.
- **make xconfig**: the most convenient, based on graphic dialogues in X Window. We need to have tcl/tk libraries installed, since this configuration is programmed in this language. The configuration is based on tables of dialogues and buttons/checkboxes, can be done fairly quickly and has help with comments on most options. But it has a defect, which is that some options may not appear (it depends on whether the configuration program is updated and sometimes it is not). In this last case, *make config* (or *menuconfig*) is the only one we can be sure will offer all the options we can choose; for the other types of configuration it depends on whether the programs have been adapted to the new options in time for the kernel being released. Although in general they try to do it at the same time.

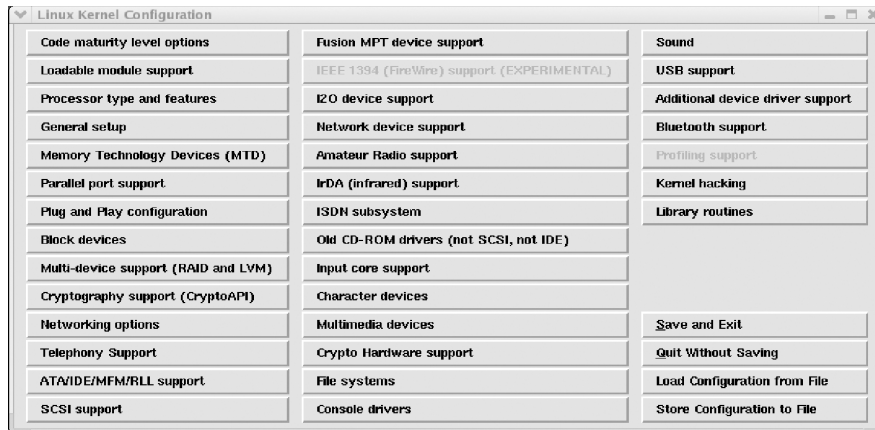


Figure 2. Configuration of the kernel (make xconfig) from graphic interface in X Window

Once the configuration process has been done, we need to save the file (*.config*), since the configuration requires a considerable amount of time. Also, it may be useful to have the configuration done if the plan is to do it on several similar or identical machines.

Another important issue concerning configuration options is that in many cases we will be asked if we want a specific characteristic integrated into the kernel or as a module (in the section on modules we will provide more details on them). This is a fairly important decision, since in certain cases our choice will influence the performance of the kernel (and therefore of the entire system).

The Linux kernel has become very large, due both to its complexity and to the device controllers (drivers) [AR01] that it includes. If we integrated everything, we could create a very large kernel file that would occupy a lot of memory and, therefore, slow down some functioning aspects. The modules of the kernel [Hen] are a method that makes it possible to divide part of the kernel into smaller sections, which will be loaded dynamically upon demand or when they are necessary for either explicit load or use of a feature.

The normal choice is to integrate what is considered fundamental for functioning or critical for performance within the kernel and to leave parts or controllers that will be used sporadically as modules for future extensions of the equipment.

- A clear case are the device controllers: if we are updating the machine, it may be that when it comes to creating the kernel we are not sure what hardware it will have: for example, what network card; but we do know that it will be connected to a network, so, the network support will be integrated into the kernel, but for the card controllers we can select a few (or all) of them and install them as modules. Then, when we have the card we can load the required module or

if we need to change one card for another later, we will just have to change the module to be loaded. If just one controller were integrated into the kernel and we changed the card, we would be forced to reconfigure and recompile the kernel with the new card's controller.

- Another case that arises (although it is not very common) is when we have two devices that are incompatible with each other, or when one or the other is functioning (for example, this tends to happen with a parallel cable printer and hardware connected to the parallel port). Therefore, in this case, we need to put the controllers as modules and load or download the one we need.

- Another example is the case of file systems. Normally we would hope that our system would have access to some of them, like ext2 or ext3 (belonging to Linux), VFAT (belonging to Windows 95/98/ME), and we will enable them in configuring the kernel. If at some moment we have to read another unexpected type, for example data stored on a disk or partition of the Windows NT/XP NTFS system, we would not be able to: the kernel would not know how to or would not have support to do so. If we have foreseen that at some point (but not usually) we may need to access these systems, we could leave the other file systems as modules.

3) Compiling the kernel

We will start the compilation using *make*, first we will have to generate the possible dependencies between the code and then the type of image of the kernel that we want (in this case, a compressed image, which tends to be the normal case):

```
make dep
make bzImage
```

When this process is completed, we will have the integrated part of the kernel; we are missing the parts that we have set as modules:

```
make modules
```

At this point we have done the configuring and compiling of the kernel. This part could be done by a normal user or by the root user, but now we will definitely need the root user, because we will move onto the installation part.

4) Installation

We'll start by installing the modules:

```
make modules_install
```

And the installation of the new kernel (from the directory `/usr/src/linux-version` or the one we have used as temporary):

```
cp arch/i386/boot/bzImage /boot/vmlinuz-2.4.0
cp System.map /boot/System.map-2.4.0
```

the file `bzImage` is the newly compiled kernel, which is placed in the `/boot` directory. Normally, we will find the old kernel in the same `/boot` directory with the name `vmlinuz` or `vmlinuz-previous-version` as a symbolic link to the old kernel. Once we have our kernel, it is better to keep the old one, in case any faults occur or the new one functions badly, so that we can recover the old one. The file `System.map` contains the symbols available for the kernel and is necessary for the processing of starting it up; it is also placed in the same directory.

On this point, we also need to consider that when the kernel starts up it may need to create `initrd` type files, which serve as a compound image of some basic drivers and is used when loading the system, if the system needs those drivers before booting certain components. In some cases, it is vital because in order to boot the rest of the system, certain drivers need to be loaded in a first phase; for example specific disk controllers such as RAID or volume controllers, which would be necessary so that in a second phase, the disk can be accessed for booting the rest of the system.

The kernel can be generated with or without an `initrd` image, depending on the needs of the hardware or system in question. In some cases, the distribution imposes the need to use an `initrd` image, in other cases it will depend on our hardware. It is also often used to control the size of the kernel, so that its basics can be loaded through the `initrd` image and later the rest in a second phase in the form of modules. In the case of requiring the `initrd` image, it would be created using the `mkinitrd` utility (see `man`, or chapter workshop), within the `/boot` directory.

5) The following step is to tell the system what kernel it needs to boot with, although this depends on the Linux booting system:

- From booting with lilo [Zan][Skoa], whether in the MBR (*master boot record*) or from an own partition, we need to add the following lines to the configuration file (in: `/etc/lilo.conf`):

```
image = /boot/vmlinuz-2.4.0
label = 2.4.0
```

where *image* is the kernel to be booted, and *label* is the name that the option will appear with during booting. We can add these lines or modify the ones of the old kernel. We recommend adding them and leaving the old kernel, in case any problems occur, so that the old one can be recovered. In the file */etc/lilo.conf* we may have one or more start up configurations, for either Linux or other systems (such as Windows).

Every start up is identified by its line *image* and the label that appears in the boot menu. There is a line *default = label* that indicates the label that is booted by default. We can also add *root = /dev/...* to the preceding lines to indicate the disk partition where the main file system is located (the '/'), remembering that the disks have devices such as */dev/hda* (1st disk ide) */dev/hdb* (2 disk ide) or */dev/sdx* for SCSI (or emulated) disks, and the partition would be indicated as *root = /dev/hda2* if the '/' of our Linux were on the second partition of the first ide disk. Using "*append =*" we can also add parameters to the kernel start up [Gor]. If the system uses *initrd*, we will also have to indicate which is the file (which will also be located in */boot/initrd-versionkernel*), with the option "*initrd=*". After changing the lilo configuration, we need to write it for it to boot:

```
/sbin/lilo -v
```

We reboot and start up with the new kernel.

If we have problems, we can recover the old kernel, by selecting the option of the old kernel, and then, using the retouch *lilo.conf*, we can return to the old configuration or study the problem and reconfigure and recompile the kernel.

- Boot with grub [Kan01][Pro]. In this case, handling is simple, we need to add a new configuration consisting of the new kernel and adding it as another option to the grub file. Next, reboot in a similar way as with lilo, but remembering that in grub it is sufficient to edit the file (typically */boot/grub/menu.lst*) and to reboot. It is also better to leave the old configuration in order to recover from potential errors.

3.2. Migration to kernel 2.6.x

In the case of having to update versions of old distributions, or changing the kernel generation using the source code, we will have to take some aspects into account, due to the novelties introduced into kernel branch 2.6.x.

Here is a list of some of the specific points to consider:

- Some of the kernel modules have changed their name, and some may have disappeared, we need to check the situation of the dynamic modules that are loaded (for example, examine `/etc/modules` and/or `/etc/modules.conf`) and edit them to reflect the changes.
- New options have been added to the initial configuration of the kernel: like `make gconfig`, a configuration based on `gtk` (Gnome). In this case, as a prerequisite, we will need to look out for Gnome libraries. The option `make xconfig` has now been implemented with the `qt` libraries (KDE).
- The minimum required versions of various utilities needed for the compilation process are increased (consult Documentation/Changes in the kernel sources). Especially, the minimum `gcc` compiler version.
- The default package for the module utilities has changed, becoming `module-init-tools` (instead of `modutils` used in 2.4.x). This package is a prerequisite for compiling kernels 2.6.x, since the modules loader is based on this new version.
- The `devfs` system becomes obsolete in favour of `udev`, the system that controls the *hotplug* start up (connection) of devices (and their initial recognition, in fact simulating a hotplug start up when the system boots), dynamically creating inputs in the directory `/dev`, only for devices that are actually present.
- In Debian as of certain versions of branch 2.6.x, for the binary images of the kernels, headers and source code, the name of the packages changes from `kernel-images/source/headers` to `linux-image/source/headers`.
- In some cases, new technology devices (like SATA) may have moved from `/dev/hdX` to `/dev/sdX`. In these cases, we will have to edit the configurations of `/etc/fstab` and the bootloader (`lilo` or `grub`) in order to reflect the changes.
- There may be some problems with specific input/output devices. The change in name of kernel modules has affected, among others, mouse devices, which likewise can affect the running of X-Window, until the required models are verified and the correct modules are loaded (for example `psmouse`). At the same time, the kernel integrates the `Alsa` sound drivers. If we have the old `OSS`, we will have to eliminate them from the loading of modules, since `Alsa` already takes care of emulating these.
- Regarding the architectures that the kernel supports, we need to bear in mind that kernel 2.6.x, in its different revisions, has been increasing the supported architectures which will allow us to have the binary images of the kernel in the distributions (or the options for compiling the kernel) best suited to supporting our processors. Specifically, we can find archi-

tectures such as i386 (for Intel and AMD): supporting the compatibility of Intel in 32 bits for the entire family of processors (some distributions use the 486 as the general architecture), some distributions integrate differentiated versions for i686 (Intel from pentium pro thereafter), for k7 (AMD Athlon thereafter), and those specific to 64 bits, for AMD 64 bits, and Intel with em64t extensions of 64 bits such as Xeon, and multicores. At the same time, there is also the IA64 architecture for 64bit Intel Itanium models. In most cases, the architectures have SMP capabilities activated in the kernel image (unless the distribution supports versions with and without SMP, created independently, in this case, the suffix *-smp* is usually added to the image that supports it).

- In Debian, to generate *inirtd* images, as of certain versions of the kernel ($\geq 2.6.12$) the *mkinitrd* tools are considered obsolete, and are replaced with new utilities such as *initramfs* tools or *yaird*. Both allow the *initrd* image to be built, but the former is the recommended one (by Debian).

3.3. Compilation of the kernel versions 2.6.x

In versions 2.6.x, bearing in mind the abovementioned considerations, the compilation takes place in a similar way to the one described above:

Having downloaded the kernel 2.6.x (with x the number or pair of numbers of the kernel revision) to the directory that will be used for the compilation and checking the required versions of the basic utilities, we can proceed to the step of compiling and cleaning up previous compilations:

```
# make clean mrproper
```

configuration of parameters (remember that if we have a previous *.config*, we will not be able to start the configuration from zero). We do the configuration through the selected *make* option (depending on the interface we use):

```
# make menuconfig
```

construction of the kernel's binary image

```
# make dep
# make bzImage
```

construction of the modules (those specified as such):

```
# make modules
```

installation of the created modules (/lib/modules/version)

```
# make modules_install
```

copying of the image to its final position (assuming i386 as the architecture):

```
# cp arch/i386/boot/bzimage /boot/vmlinuz-2.6.x.img
```

and finally, creating the initrd image that we consider necessary, with the necessary utilities according to the version (see subsequent comment). And adjustment of the lilo or grub bootloader depending on which one we use.

The final steps (vmlinuz, system.map and initrd) of moving files to /boot can normally also be done with the process:

```
# make install
```

but we need to take into account that it does the entire process and will update the bootloaders, removing or altering old configurations; at the same time, it may alter the default links in the */boot* directory. We need to bear this in mind when it comes to thinking of past configurations that we wish to save.

Regarding the creation of the initrd, in Fedora/Red Hat it will be created automatically with the *install* option. In Debian we should either use the techniques of the following section or create it expressly using *mkinitrd* (versions $\leq 2.6.12$) or, subsequently, with *mkinitramfs*, or a utility known as *update-initramfs*, specifying the version of the kernel (it is assumed that it is called *vmlinuz-version* within the */boot* directory):

```
# update-initramfs -c -k 'version'
```

3.4. Compilation of the kernel in Debian (Debian way)

In Debian, in addition to the examined methods, we need to add the configuration using the method known as Debian Way. A method that allows us to build the kernel in a fast and flexible manner.

For the process, we will need several utilities (install the packages or similar): *kernel-package*, *ncurses-dev*, *fakeroot*, *wget*, *bzip2*.

We can see the method from two perspectives, rebuilding a kernel equivalent to the one provided by the distribution or tailoring it and then using the method for building an equivalent personalised kernel.

In the first case, we initially obtain the version of the kernel sources provided by the distribution (meaning x the revision of the kernel 2.6):

```
# apt-get install linux-source-2.6.x
$ tar -xvjf /usr/src/linux-source-2.6.x.tar.bz2
```

where we obtain the sources and decompress them (the package leaves the file in */usr/src*).

Installing the basic tools:

```
# apt-get install build-essential fakeroot
```

Checking source dependencies

```
# apt-get build-dep linux-source-2.6.x
```

And construction of the binary, according to the pre-established package configuration (similar to that included in the official image packages of the kernel in Debian):

```
$ cd linux-source-2.6.x
$ fakeroot debian/rules binary
```

There are some extra procedures for creating the kernels based on different patch levels provided by the distribution and possibilities of generating different final configurations (view the reference note to complement these aspects).

In the second, more common case, when we would like a personalised kernel, we will have to follow a similar process through a typical tailoring step (for example, using *make menuconfig*); the steps would be:

obtaining and preparing the directory (here we obtain the distribution's packages, but it is equivalent to obtaining the sources from *kernel.org*):

```
# apt-get install linux-source-2.6.x
$ tar xjf /usr/src/linux-source-2.6.x.tar.bz2
$ cd linux-source-2.6.x
```

next, we configure the parameters, as always, we can base ourselves on *.config* files that we have used previously, to start from a known configuration (for tailoring we can also use any of the other methods, *xconfig*, *gconfig*...):

```
$ make menuconfig
```

Note

We can see the Debian way process in a detailed manner in: <http://kernel-handbook.alioth.debian.org/>

final construction of the kernel depending on `initrd` or not, without `initrd` available (we need to take care with the version we use; as of a certain version of the kernel, the use of the `initrd` image can be mandatory):

```
$ make-kpkg clean
$ fakeroot make-kpkg --revision=custom.1.0 kernel_image
```

or if we have `initrd` available (already built)

```
$ make-kpkg clean
$ fakeroot make-kpkg - -initrd - -revision=custom.1.0 kernel_image
```

The process will end with adding the associated package to the kernel image, which we will finally be able to install:

```
# dpkg -i ../linux-image-2.6.x_custom.1.0_i386.deb
```

In this section, we will also add another peculiarity to be taken into consideration in Debian, which is the existence of utilities for adding dynamic kernel modules provided by third parties. In particular, the *module-assistant* utility helps to automate this process on the basis of the module sources.

We need to have the headers of the kernel installed (package `linux-headers-version`) or the sources we use for compiling the kernel. As of here, the *module-assistant* can be used interactively, allowing us to select from an extensive list of previously registered modules in the application, and it can be responsible for downloading the module, compiling it and installing it in the existing kernel.

Also from the command line, we can simply specify (`m-a` is equivalent to `module-assistant`):

```
# m-a prepare
# m-a auto-install module_name
```

which prepares the system for possible dependencies, downloads the module sources, compiles them and, if there are no problems, installs them for the current kernel. We can see the name of the module on the interactive list of the module assistant.

4. Patching the kernel

In some cases the application of patches to the kernel [lkm] is also common.

A patch file in relation to the Linux kernel is an ASCII text file that contains the differences between the original source code and the new code, with additional information on file names and code lines. The patch program (see *man patch*) serves to apply it to the tree of the kernel source code (normally in */usr/src*).

The patches are usually necessary when special hardware requires some modification of the kernel or some bugs (errors) have been detected subsequent to a wide distribution of a kernel version or else a new specific feature is to be added. In order to correct the problem (or add the new feature), it is usual to distribute a patch instead of an entire new kernel. When there are already several of these patches, they are added to various improvements of the preceding kernel to form a new version of the kernel. In all events, if we have problematic hardware or the error affects the functioning or stability of the system and we cannot wait for the next version of the kernel; we will have to apply the patch.

The patch is usually distributed in a compressed file of the type bz2 (bunzip2, although you can also find it in gzip with the extension .gz), as in the case of for example:

```
patchxxxx-2.6.21-pversion.bz2
```

where xxxx is usually any message regarding the type or purpose of the patch 2.6.21 would be the version of the kernel to which the patch is to be applied, and pversion would refer to the version of the patch, of which there can also be several. We need to bear in mind that we are speaking of applying patches to the sources of the kernel (normally installed, as we have already seen, in */usr/src/linux* or a similar directory).

Once we have the patch, we must apply it, we will find the process to follow in any readme file that accompanies the patch, but generally the process follows the steps (once the previous requirements are checked) of decompressing the patch in the source files directory and applying it over the sources of the kernel, for example:

```
cd /usr/src/linux (or /usr/src/linux-2.6.21 or any other version).
```

```
bunzip2 patch-xxxxx-2.6.21-version.bz2  
patch -p1 < patch-xxxxx-2.6.21-version
```

and afterwards we will have to recompile the kernel in order to generate it again.

The patches can be obtained from different places. Normally, we can find them in the kernel storage site (www.kernel.org) or else in www.linuxhq.com, which has a complete record of them. Some Linux communities (or individual users) also offer corrections, but it is better to search the standard sites in order to ensure that the patches are trustworthy and to avoid possible security problems with "pirate" patches. Another way is the hardware manufacturer, which may offer certain modifications of the kernel (or controllers) so that its devices work better (one known example is Linux NVIDIA and the device drivers for its graphic cards).

Finally, we should point out that many of the GNU/Linux distributions (Fedora/Red Hat, Mandriva...), already offer the kernels patched by themselves and systems for updating them (some even automatically, as in the case of Fedora/Red Hat and Debian). Normally, in production systems it is more advisable to keep up with the manufacturer's updates, although it does not necessarily offer the latest published kernel, but rather the one that it finds most stable for its distribution, at the expense of missing the latest generation features or technological innovations included in the kernel.

Note

For systems that we want to update, for testing reasons or because we need the latest features, we can always go to www.kernel.org and obtain the latest published kernel.

5. Kernel modules

The kernel is capable of loading dynamic portions of code (modules) on demand [Hen], in order to complement its functionality (this possibility is available from kernel version 1.2 and higher). For example, the modules can add support for a file system or for specific hardware devices. When the functionality provided by the module is not necessary, the module can be downloaded, freeing up memory.

On demand, the kernel usually identifies a characteristic not present in the kernel at that moment it makes contact with a thread of the kernel known as `kmod` (in kernel versions 2.0.x the daemon was called *kernel*d), this executes a command, `modprobe`, to try and load the associated module from or of a chain with the name of the module or else from an generic identifier; this information is found in the file `/etc/modules.conf` in the form of an alias between the name and the identifier.

Next, we search in `/lib/modules/version_kernel/modules.dep`

to find out whether there are dependencies with other modules. Finally, with the `insmod` command the module is loaded from `/lib/modules/version_kernel/` (the standard directory for modules), the `version_kernel` is the current version of the kernel using the `uname -r` command in order to set it. Therefore, the modules in binary form are related to a specific version of the kernel, and are usually located in `/lib/modules/version-kernel`.

If we need to compile them, we will need to have the sources and/or headers of the version of the core for which it is designed.

There are some utilities that allow us to work with modules (they usually appear in a software package called *modutils*, which was replaced by the module `-init-tools` for managing modules of the 2.6.x branch):

- **lsmod**: we can see the loaded modules in the kernel (the information is obtained from the pseudofile `/proc/modules`). It lists the names and dependencies with others (in `[ ]`), the size of the module in bytes, and the module use counter; this allows it to be downloaded if the count is zero.

Note

The modules offer the system a large degree of flexibility, allowing it to adapt to dynamic situations.

Example

Some modules in a Debian distribution:

Module	Size	Used by	Tainted: P
agpgart	37.344	3	(autoclean)
apm	10.024	1	(autoclean)
parport_pc	23.304	1	(autoclean)
lp	6.816	0	(autoclean)
parport	25.992	1	[parport_pc lp]
snd	30.884	0	
af_packet	13.448	1	(autoclean)
NVIDIA	1.539.872	10	
es1371	27.116	1	
soundcore	3.972	4	[snd es1371]
ac97_codec	10.9640	0	[es1371]
gameport	1.676	0	[es1371]
3c59x	26.960	1	

- **modprobe**: tries the loading of a module and its dependencies.
- **insmod**: loads a specific module.
- **depmod**: analyses dependencies between modules and creates a file of dependencies.
- **rmmod**: removes a module from the kernel.
- Other commands can be used for debugging or analysing modules, like *mod-info*, which lists some information associated to the module or *ksyms*, which (only in versions 2.4.x) allows examination of the symbols exported by the modules (also in */proc/ksyms*).

In order to load the module the name of the module is usually specified, either by the kernel itself or manually by the user using *insmod* and specific parameters optionally. For example, in the case of devices, it is usual to specify the addresses of the I/O ports or IRQ or DMA resources. For example:

```
insmod soundx io = 0x320 irq = 5
```

6. Future of the kernel and alternatives

At certain moments, advances in the Linux kernel were released at very short intervals, but now with a fairly stable situation regarding the kernels of the 2.6.x series, more and more time elapses between kernel versions, which in some ways is very positive. It allows time for correcting errors, seeing what ideas did not work well, and trying new ideas, which, if they work, are included.

In this section, we'll discuss some of the ideas of the latest kernels and some of those planned for the near future in the development of the kernel.

The previous series, series 2.4.x [DBo], included in most current distributions, contributions were made in:

- Fulfilling IEEE POSIX standards, this means that many existing UNIX programs can be recompiled and executed in Linux.
- Improved devices support: PnP, USB, Parallel Port, SCSI...
- Support for new file systems, like UDF (CD-ROM rewritable like a disc). Other journaled systems, like Reiser from IBM or the ext3, these allow having a log (*journal*) of the file system modifications and thus they are able to recover from errors or incorrect handling of files.
- Memory support up to 4 GB, in its day some problems arose (with the 1.2x kernels) which would not support more memory than 128 MB (at that time it was a lot of memory).
- The /proc interface was improved. This is a pseudo-filesystem (the directory /proc) that does not really exist on the disc, but that is simply a way of accessing the data of the kernel and of the hardware in an organised manner.
- Sound support in the kernel: Alsa controllers, which were configured separately beforehand, were partially added,.
- Preliminary support for RAID software and the dynamic volumes manager LVM1 was included.

In the current series, kernel branch 2.6.x [Pra] has made important advances in relation to the previous one (with the different.x revisions of the 2.6 branch):

Note

The kernel continues to evolve, incorporating the latest in hardware support and improved features.

- Improved SMP features, important for the multi-core processors widely used in business and scientific environments.
- Improvements in the CPU scheduler.
- Improvements in the multithread support for user applications. New models of threads NGPT (IBM) and NPTL (Red Hat) are incorporated (over time NPTL was finally consolidated).
- Support for USB 2.0.
- Also sound controllers incorporated in the kernel.
- New architectures for 64-bit CPUs, supporting AMD x86_64 (also known as amd64) and PowerPC 64 and IA64 (Intel Itanium architecture).
- Support for journaled file systems: JFS, JFS2 (IBM), and XFS (Silicon Graphics).
- Improved I/O features, and new models of unified controllers.
- Improvements in implementing TCP/IP, and the NFSv4 system (sharing of the file system with other systems via the network).
- Significant improvements for a preemptive kernel: allowing the kernel to manage internally various tasks that can interrupt each other, essential for the efficient implementation of real time systems.
- System suspension and restoration after rebooting (by kernel).
- UML, User Mode Linux, a sort of virtual Linux machine on Linux that allows us to see a Linux (in user mode) running on a virtual machine. This is ideal for debugging now that a version of Linux can be developed and tested on another system, which is useful for the development of the kernel itself and for analysing its security.
- Virtualisation techniques included in the kernel: the distributions have gradually been incorporating different virtualisation techniques, which require extensions to the kernel; we should emphasise, for example, kernels modified for Xen, or Virtual Server (Vserver).
- New version of the volumes support LVM2.
- New pseudo file system `/sys`, designed to include the system information and devices that will be migrating from the `/proc` system, leaving the latter

with information regarding the processes and their development during execution.

- FUSE module for implementing file systems on user space (above all the NTFS case).

In the future, improvement of the following aspects is planned:

- Increasing the virtualisation technology in the kernel, for supporting different operating system configurations and different virtualisation technologies, in addition to better hardware support for virtualisation included in the processors that arise with new architectures.
- The SMP support (multi-processor machines) of 64-bit CPUs (Intel's Itanium, and AMD's Opteron), the support of multi-core CPUs.
- Improved file systems for clustering and distributed systems.
- Improvement for kernels optimised for mobile devices (PDA, teléfonos...).
- Improved fulfilment of the POSIX standard etc.
- Improved CPU scheduling; although in the initial series of the 2.6.x branch many advances were made in this aspect, there is still low performance in some situations, in particular in the use of interactive desktop applications, different alternatives are being studied to improve this and other aspects.

Also, although it is separate from the Linux systems, the FSF (Free Software Foundation) and its GNU project continue working on the project to finish a complete operating system. It is important to remember that the main objective of the GNU project was to obtain a free software UNIX clone and the GNU utilities are just the necessary software for the system. In 1991, when Linux managed to combine its kernel with some GNU utilities, the first step was taken towards the culmination in today's GNU/Linux systems. But the GNU project continues working on its idea to finish the complete system. Right now, they already have a core that can run its GNU utilities. This core is known as Hurd; and a system built with it known as GNU/Hurd. There are already some test distributions, specifically, a Debian GNU/Hurd.

Hurd was designed as a core for the GNU system around 1990 when its development started, since most of the GNU software had already been developed at the time, and the only thing that was missing was the kernel. It was in 1991 when Linus combined GNU with his Linux kernel that the history of GNU/

Web site

POSIX specifications
www.UNIX-systems.org/

Web site

The GNU project:
<http://www.gnu.org/gnu/thegnuproject.html>

Reference

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

Linux systems began. But Hurd continues to develop. The development ideas for Hurd are more complex, since Linux could be considered a conservative design, based on already known and implemented ideas.

Specifically, Hurd was conceived as a collection of servers implemented on a Mach microkernel [Vah96], which is a kernel design of the microkernel type (unlike Linux, which is of the monolithic type) developed by the University of Carnegie Mellon and subsequently by that of Utah. The basic idea was to model the functionalities of the UNIX kernel as servers that would be implemented on a basic Mach kernel. The development of Hurd was delayed while the design of the Mach was being finished and this was finally published as free software, which would allow its use for developing Hurd. At this point, we should mention the importance of Mach, since many operating systems are now based on ideas extracted from it; the most outstanding example is Apple's MacOS X.

The development of Hurd was further delayed due to its internal complexity, because it had several servers with different tasks of the multithread type (execution of multiple threads), and debugging was extremely difficult. But nowadays, the first production versions of GNU/Hurd are already available, as well as test versions of a GNU/Hurd distribution.

It could be that in the not too distant future GNU/Linux systems will coexist with GNU/Hurd, or even that the Linux kernel will be replaced with the Hurd kernel, if some lawsuits against Linux prosper (read the case of SCO against IBM), since it would represent a solution for avoiding later problems. In all events, both systems have a promising future ahead of them. Time will tell how the balance will tip.

7. Tutorial: configuring the kernel to the requirements of the user

In this section we will have a look at a small interactive workshop for the process of updating and configuring the kernel in the two distributions used: Debian and Fedora.

The first essential thing, before starting, is to know the current version of the kernel we have with `uname -r`, in order to determine which is the the next version that we want to update to or personalise. And the other is to have the means to boot our system in case of errors: the set of installation CDs, the floppy disc (or CD) for recovery (currently the distribution's first CD is normally used) or some Live CD distribution that allows us to access the machine's file system, in order to redo any configurations that may have caused problems. It is also essential to back up our data or important configurations.

We will look at the following possibilities:

- 1) Updating the distribution's kernel. Automatic case of Debian.
- 2) Automatic update in Fedora.
- 3) Adapting a generic kernel (Debian or Fedora). In this last case, the steps are basically the same as those presented in the section on configuration, but we will make a few more comments:

7.1. Configuring the kernel in Debian

In the case of the Debian distribution, the installation can also be done automatically, using the APT packages system. It can be done either from the command line or with graphic APT managers (synaptic, gnome-apt...).

We are going to carry out the installation using the command line with `apt-get`, assuming that the access to the apt sources (above all to the Debian originals) is properly configured in the `/etc/apt/sources.list` file. Let's look at the steps:

- 1) To update the list of packages.

```
# apt-get update
```

- 2) To list the packages associated with images of the kernel:

```
# apt-cache search linux-image
```

3) To select a version suitable for our architecture (generic, 386/486/686 for Intel, k6 or k7 for amd or in particular for 64Bits versions amd64, intel and amd or ia64, for Intel Itanium). The version is accompanied by kernel version, Debian revision of the kernel and architecture. For example: 2.6.21-4-k7, kernel for AMD Athlon, Debian revision 4 of the kernel 2.6.21.

4) Check for the selected version that the extra accessory modules are available (with the same version number) With apt-cache we will search for whether there are other dynamic modules that could be interesting for our hardware, depending on the version of the kernel to be installed. Remember that, as we saw in the Debian way, there is also the module-assistant utility, which allows us to automate this process after compiling the kernel. If the necessary modules are not supported, this could prevent us from updating the kernel if we consider that the functioning of the problematic hardware is vital for the system.

5) Search, if we also want to have the source code of the kernel, the Linux-source-version (only 2.6.21, that is, the principal numbers) and the corresponding kernel headers, in case we later want to make a personalised kernel: in this case, the corresponding generic kernel patched by Debian.

6) Install what we have decided: if we want to compile from the sources or simply to have the code:

```
# apt-get install linux-image-version
# apt-get install xxxx-modules-version (if some modules are
necessary)
```

and

```
# apt-get install linux-source-version-generic
# apt-get install linux-headers-version
```

7) Install the new kernel, for example in the lilo bootloader (check the boot utility used, some recent Debian versions use grubas boot loader), this is done automatically. If we are asked if the initrd is active, we will have to verify the lilo file (/etc/lilo.conf) and, in the lilo configuration of the new image, include the new line:

```
initrd = /initrd.img-version (or /boot/initrd.img-version)
```

once this is configured, we would have to have a lilo of the mode (fragment), supposing that initrd.img and vmlinuz are links to the position of the files of the new kernel:

```
default = Linux

image = /vmlinuz
    label = Linux
    initrd = /initrd.img
# restricted
# alias = 1
image = /vmlinuz.old
    label = LinuxOLD
    initrd = /initrd.img.old
# restricted
# alias = 2
```

We have the first image by default, the other is the former kernel. Thus, from the lilo menu we can ask for one or the other or, simply by changing the default, we can recover the former. Whenever we make any changes in `/etc/lilo.conf` we should not forget to rewrite in the corresponding sector with the command `/sbin/lilo` or `/sbin/lilo -v`.

7.2. Configuring the kernel in Fedora/Red Hat

Updating the kernel in the Fedora/Red Hat distribution is totally automatic by means of its package management service or else by means of the graphic programs that the distribution includes for updating; for example, in business versions of Red Hat there is one called `up2date`. Normally, we will find it in the task bar or in the Fedora/Red Hat system tools menu (check the available utilities in tools/Administration menus, the currently available graphic tools are highly distribution version dependent).

This updating program basically checks the packages of the current distribution against a Fedora/Red Hat database and offers the possibility of downloading the updated packages, including those of the kernel. This Red Hat service for businesses works via a service account and Red Hat offers it for payment. With this type of utilities the kernel is updated automatically.

For example, in figure 10, we can see that once running, a new available version of the kernel has been detected, which we can select for downloading:



Figure 3. The Red Hat updating service (Red Hat Network up2date) shows the available kernel update and its sources.

In Fedora we can either use the equivalent graphic tools or simply use yum directly, if we know that new kernels are available:

```
# yum install kernel kernel-source
```

Once downloaded, we proceed to install it, normally also as an automatic process, whether with grub or lilo as boot managers. In the case of grub, it is usually automatic and leaves a pair of options on the menu, one for the newest version and the other for the old one. For example, in this grub configuration (the file is in /boot/grub/grub.conf or else /boot/grub/menu.lst), we have two different kernels, with their respective version numbers.

```
#file grub.conf
default = 1
timeout = 10
splashimage = (hd0,1)/boot/grub/splash.xpm.gz

title Linux (2.6.20-2945)
root (hd0,1)
kernel /boot/vmlinuz-2.6.20-2945 ro root = LABEL = /
initrd /boot/initrd-2.6.20-18.9.img

title LinuxOLD (2.6.20-2933)
root (hd0,1)
kernel /boot/vmlinuz-2.4.20-2933 ro root = LABEL = /
initrd /boot/initrd-2.4.20-2933.img
```

Each configuration includes a title that appears during start up. The root or partition of the disc from where it boots, the directory where the file corresponding to the kernel is found and the corresponding initrd file.

In the case of having lilo (by default grub is used) in the Fedora/Red Hat as manager, the system will also update it (file `/etc/lilo.conf`), but then we will have to rewrite the boot manually with the command `/sbin/lilo`.

It is also important to mention that with the previous installation we had the possibility of downloading the sources of the kernel; these, once installed, are in `/usr/src/linux-version` and can be compiled and configured following the usual procedure as if it was a generic kernel. We should mention that the Red Hat company carries out a lot of work on the patches and corrections for the kernel (used after Fedora) and that its kernels are modifications to the generic standard with a fair number of additions, which means that it could be better to use Red Hat's own sources, unless we want a newer or more experimental kernel than the one supplied.

7.3. Configuring a generic kernel

Let's look at the general case of installing a kernel starting from its sources. Let's suppose that we have some sources already installed in `/usr/src` (or the corresponding prefix). Normally, we would have a Linux directory or `linux-version` or simply the version number. This will be the tree of the sources of the kernel.

These sources can come from the distribution itself (or we may have downloaded them during a previous update), first it will be interesting to check whether they are the latest available, as we have already done before with Fedora or Debian. Or if we want to have the latest and generic versions, we can go to `kernel.org` and download the latest available version (better the stable one than the experimental ones), unless we are interested in the kernel's development. We download the file and in `/usr/src` (or another selected directory, even better) decompress the kernel sources. We can also search to see if there are patches for the kernel and apply them (as we have seen in section 4.4).

Next, we will comment on the steps that will have to be carried out: we will do it briefly, as many of them have been mentioned before when working on the configuration and tailoring.

1) Cleaning the directory of previous tests (where applicable):

```
make clean mrproper
```

2) Configuring the kernel with, for example: *make menuconfig* (or *xconfig*, *gconfig* or *oldconfig*). We saw this in section 4.3.

See also

It would be advisable to reread section 3.4.3.

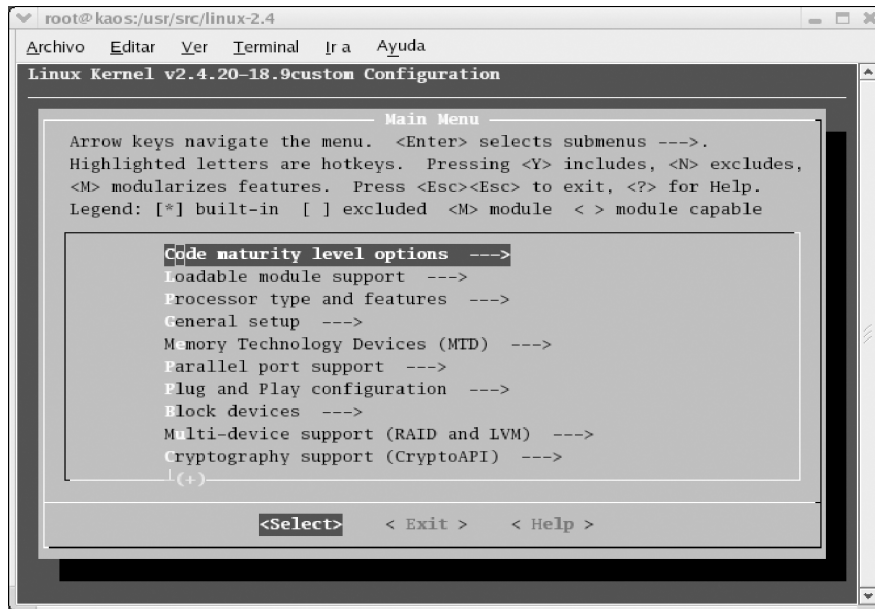


Figure 4. Configuring the kernel using text menus

4) Dependencies and cleaning of previous compilations:

```
make dep
```

5) Compiling and creating an image of the kernel: `make bzImage`. `zImage` would also be possible if the image was smaller, but `bzImage` is more normal, as it optimises the loading process and compression of larger kernels. On some ancient hardware it may not work and `zImage` may be necessary. The process can last from a few minutes to an hour on modern hardware and hours on older hardware. When it finishes, the image is found in: `/usr/src/directory-sources/arch/i386/boot`.

6) Now we can compile the modules with `make modules`. Until now we have not changed anything in our system. Now we have to proceed to the installation.

7) In the case of the modules, if we try an older version of the kernel (branch 2.2 or the first ones of 2.4), we will have to be careful, since some used to overwrite the old ones (in the last 2.4.x or 2.6.x it is no longer like this).

But we will also need to be careful if we are compiling a version that is the same (exact numbering) as the one we have (the modules are overwritten), it is better to back up the modules:

```
cd /lib/modules
tar -cvzf old_modules.tgz versionkernel-old/
```

This way we have a version in .tgz that we can recover later if there is any problem And, finally, we can install the modules with:

```
make modules install
```

8) Now we can move on to installing the kernel, for example with:

```
# cd /usr/src/directory-sources/arch/i386/boot
# cp bzImage /boot/vmlinuz-versionkernel
# cp System.map /boot/System.map-versionkernel
# ln -s /boot/vmlinuz-versionkernel /boot/vmlinuz
# ln -s /boot/System.map-versionkernel /boot/System.map
```

This way we store the symbols file of the kernel (System.map) and the image of the kernel.

9) Now all we have to do is put the required configuration in the configuration file of the boot manager, whether lilo (/etc/lilo.conf) or grub (/boot/grub/grub.conf) depending on the configurations we already saw with Fedora or Debian. And remember, in the case of lilo, that we will need to update the configuration again with /sbin/lilo or /sbin/lilo -v.

10) Restart the machine and observe the results (if all has gone well).

Activities

- 1) Determine the current version of the Linux kernel incorporated into our distribution. Check the available updates automatically, whether in Debian (*apt*) or in Fedora/Red Hat (via *yum*).
- 2) Carry out an automatic update of our distribution. Check possible dependencies with other modules used (whether *pcmcia* or others) and with the bootloader (*lilo* or *grub*) used. A backup of important system data (account users and modified configuration files) is recommended if we do not have another system that is available for tests.
- 3) For our branch of the kernel, to determine the latest available version (consult <http://www.kernel.org>) and carry out a manual installation following the steps described in the unit. The final installation can be left optional, or else make an entry in the bootloader for testing the new kernel.
- 4) In the case of the Debian distribution, in addition to the manual steps, we saw how there is a special way (recommended) of installing the kernel from its sources using the *kernel-package*.

Bibliography

Other sources of reference and information

[Kerb] Site that provides a repository of the different versions of the Linux kernel and its patches.

[Kera] [lkm] Web sites that refer to a part of the Linux kernel community. It offers various documentary resources and mailing lists of the kernel's evolution, its stability and the new features that develop.

[DBo] Book about the Linux 2.4 kernel, which details the different components, their implementation and design. There is a first edition about the 2.2 kernel and a new update to the 2.6 kernel.

[Pra] An article that describes some of the main innovations of the new 2.6 series of the Linux kernel.

[Ker] [Mur] Documentation projects of the kernel, incomplete but with useful material.

[Bac86] [Vah96] [Tan87] Some texts about the concepts, design and implementation of the kernels of different UNIX versions.

[Skoa][Zan01][Kan][Pro] For further information on lilo and grub loaders.

Local administration

Josep Jorba Esteve

PID_00148465



Universitat Oberta
de Catalunya

www.uoc.edu

Index

Introduction.....	5
1. Distributions: special features.....	7
2. Boot and run levels	9
3. Monitoring system state.....	12
3.1. System boot	12
3.2. kernel: /proc directory	13
3.3. kernel: /sys	14
3.4. Processes	14
3.5. System Logs	15
3.6. Memory	17
3.7. Disks and file systems	17
4. File Systems.....	21
4.1. Mount point	22
4.2. Permissions	25
5. Users and groups.....	27
6. Printing services.....	32
6.1. BSD LPD	36
6.2. LPRng	37
6.3. CUPS	39
7. Disk management.....	42
7.1. RAID software	44
7.2. Logical Volume Manager (LVM)	50
8. Updating Software.....	54
9. Batch jobs.....	56
10. Tutorial: combined practices of the different sections.....	58
Activities.....	67
Bibliography.....	68

Introduction

One of the administrator's first tasks will be to manage the machine's local resources. Some of these aspects were basically covered in the GNU/Linux course. In this course, we will cover these management tasks in more depth as well as some of the customisation and resource efficiency aspects.

We will start by analysing the process for starting up a GNU/Linux system, which will help us to understand the initial structure of the system and its relationship with the various services that it provides.

We will now learn how to obtain a general overview of the current state of the system, using different procedures and commands that are available for evaluating the various parts of the system; this will allow us to make administrative decisions if we detect any faults or deficiencies in the performance or if we find that we are missing any of the resources.

One of the administrator's main tasks is managing the user accounts, as any configuration of the machine will be designed for the users; we will see how we can define new user accounts and control the levels to which they may access the resources.

With regard to the system's peripherals, such as disks and printers, there are different management possibilities available, either through different servers (for printing) or different filing systems that we can treat, as well as some techniques for optimising the disks' performance.

We will also examine the need to update the system and how best to keep it updated; likewise, we will examine how to install new applications and software and how to make these programs available to the users. At the same time, we will analyse the problems involved in executing predetermined timed tasks in the system.

In the last tutorial, we will learn how to evaluate the state of a machine, following the points that we have seen in this module, and we will carry out some of the basic administrative tasks we have described. In this module, we will discuss some of the commands and subsequently, in the tutorial, we will examine some of these in more detail, with regard to how they work and the options available.

Note

Local administration covers many varied tasks, which are possibly the ones that the administrator will most use during their daily routines.

1. Distributions: special features

We will now try to outline some minor technical differences (which are constantly being reduced) in the distributions (Fedora/Red Hat and Debian) used [Mor03], which we will examine in more detail throughout the modules as they appear.

Modifications to or particularities of Fedora/Red Hat:

- Using the grub boot loader (a GNU utility); unlike previous versions of most distributions, which tend to use lilo, Fedora uses grub. GRUB (*grand unified bootloader*) has a text-mode configuration (usually in `/boot/grub/grub.conf`) that is quite simple and that can be modified when booting. It is possibly more flexible than lilo. Lately, distributions tend to use grub; Debian also includes it as an option.
- Management of alternatives. If there is more than one equivalent program present for a specific task, the alternative that will be used must be indicated through a directory (`/etc/alternatives`). This system was borrowed from Debian, which uses it a lot in its distribution.
- TCP/IP portscanning program based on xinetd; in `/etc/xinetd.d` we will find the modular configuration files for some of the TCP/IP services, along with the `/etc/xinetd.conf` configuration file. In classic UNIX systems, the program used for this was inetd, which had a single configuration file in `/etc/inetd.conf`, which was the case, for example, in the Debian distribution, which uses inetd, leaving xinetd as an option.
- Some special configuration directories: `/etc/profile.d`, files that are executed when a user opens a shell; `/etc/xinetd.d`, configuration of some net services; `/etc/sysconfig`, configuration data for various aspects of the system; `/etc/cron.`, various directories where the tasks that have to be performed regularly are specified (through crontab); `/etc/pam.d`, where the authentication modules are known as PAM: the permissions for the particular service or program are configured in each of the PAM files; `/etc/logrotate.d`, rotation configuration (when it is necessary to clean, compress etc.) of some of the log files for different services.
- There is a software library called kudzu, which examines the hardware at start-up to detect any possible changes (in some previous versions of Fedora) in the configuration and to create the appropriate elements or configurations. Although there is currently a progressive migration to API Hal that controls precisely this aspect.

Note

It is important to know the details of a distribution, as they are essential for performing a task or resolving an issue (for example, if there are extra tools available).

In Debian's case:

- In-house packaging system based on DEB packages, with tools at various levels for working with packages such as: `dpkg`, `apt-get`, `dselect`, `tasksel`.
- Debian follows FHS, over the directories structure, adding some particulars in `/etc`, such as: `/etc/default`, configuration files and default values for some programs; `/etc/network`, data and network interfaces configuration scripts; `/etc/dpkg y /etc/apt`, information on the configuration of the package management tools; `/etc/alternatives`, links to the default programs, in which there are (or may be) various available alternatives.
- Configuration system for many software packages using the `dpkg-reconfigure` tool. For example:

```
dpkg-reconfigure gdm
```

makes it possible to select the incoming manager for X, or:

```
dpkg-reconfigure X-Window-system
```

allows us to configure the different elements of X.

- Uses the TCP/IP services configuration through `inetd`; the configuration is in file `/etc/inetd.conf`; there is an `update-inetd` tool for disabling or creating services entries.
- Some special configuration directories: `/etc/cron.`, several directories where the tasks that have to be performed regularly are specified (though `crontab`); `/etc/pam.d`, where PAM are authentication modules.

2.Boot and run levels

A first important point in the analysis of a system's local performance is how it works on the runlevels, which determine the current work mode of the system and the services provided (on the level) [Wm02].

A service is a functionality provided by the machine, normally based on daemons (or background execution processes that control network requests, hardware activity or other programs that provide any task).

The services can be activated or halted using scripts. Most standard processes, which are usually configured in the /etc directory, tend to be controlled with the scripts in /etc/init.d/. Scripts with names similar to those of the service to which they correspond usually appear in this directory and starting or stopping parameters are usually accepted. The following actions are taken:

/etc/init.d/service start	start the service.
/etc/init.d/service stop	stop the service.
/etc/init.d/service restart	stop and subsequent
	restart of the service.

When a GNU/Linux system starts up, first the system's kernel is loaded, then the first process begins; this process is called init and it has to execute and activate the rest of the system, through the management of different runlevels.

A runlevel is basically a configuration of programs and services that will be executed in order to carry out determined tasks.

The typical levels, although there may be differences in the order, especially at levels 2-5 (in the configuration table in Fedora and that recommended in the LSB standard), are usually:

Runlevel	Function	Description
0	Halt	Halts or shuts down the active services and programs, and umounts active file systems for CPU.
1	Single-user mode	Halts or shuts down most services, only permitting the (root) administrator to login. Used for maintenance tasks and correcting critical errors.
2	Multi-user mode without networking	No networking services are started and only local logins are allowed.
3	Multi-user	Starts up all the services except the graphics associated to X Window.

Runlevel	Function	Description
4	Multi-user	Not usually used; normally the same as 3.
5	Multi-user X	As with 3, but with X support for user logins (graphic login).
6	Reboot	For all programs and services. Reboots the system.

On the other hand, it should be noted that Debian uses a model in which practically no distinction is made between runlevels 2-5 and performs exactly the same task (although this may change in a future version, so that these levels correspond with the LSB).

These runlevels are usually configured in GNU/Linux systems (and UNIX) by two different systems: BSD or System V (sometimes abbreviated to sysV). In the cases of Fedora and Debian, System V is used, which is the one that we will examine, but other UNIX and some GNU/Linux distributions (such as Slackware) use the BSD model.

In the case of the runlevel model of System V, when the init process begins, it uses a configuration file called `/etc/inittab` to decide on the execution mode it will enter. This file defines the runlevel by default (*initdefault*) at start-up (by installation, 5 in Fedora and 2 in Debian), and a series of terminal services that must be activated so that users may log in.

Afterwards, the system, according to the selected runlevel, will consult the files contained in `/etc/rcn.d`, where *n* is the number associated to the runlevel (the selected level), which contains a list of services that should be started or halted if we boot up in the runlevel or abandon it. Within the directory, we will find a series of scripts or links to the scripts that control the service.

Each script has a number related to the service, an S or K initial that indicates whether it is the script for starting (S) or killing (K) the service, and a number that shows the order in which the services will be executed.

A series of system commands help us to handle the runlevels; we must mention:

- The scripts, which we have already seen, in `/etc/init.d/` allow us to start-up, halt or reboot individual services.
- `telinit`, allows us to change the runlevel; we simply have to indicate the number. For example, we have to perform a critical task in root; with no users working, we can perform a `telinit 1` (S may also be used) to pass to the single-user runlevel and then, after the task, a `telinit 3` to return to multi-user mode. The `init` command may also be used for the same task, although `telinit` does provide a few extra parameters. For example, the typical reboot of a UNIX system would be performed with `sync; init 6`, the

sync command forces the buffers of the files system to empty, and then we reboot at runlevel 6.

- shutdown allows us to halt ("h") or reboot the system ("r"). This may be performed in a given period of time or immediately. There are also the halt and reboot commands for these tasks.
- wall allows us to send warning messages to the system users. Specifically, the administrator may warn users that the machine is going to stop at a determined moment. Commands such as shutdown usually use them automatically.
- pidof permits us to find out the process ID associated to a process. With ps we obtain the lists of the processes, and if we wish to eliminate a service or process through a kill, we will need its PID.

There are some small changes in the distributions, with regard to the start-up model:

- Fedora/Red Hat: runlevel 4 has no declared use. The /etc/rcn.d directories exist as links to /etc/rc.d subdirectories, where the start-up scripts are centralised. The directories are as follows: /etc/rc.d/rcn.d; but as the links exist, it is transparent to the user. The default runlevel is 5 when starting up with X.

The commands and files associated to the system's start-up are in the sysvinit and initscripts software packages.

Regarding the changes to files and scripts in Fedora, we must point out that in /etc/sysconfig we can find files that specify the default values for the configuration of devices or services. The /etc/rc.d/rc.sysinit script is invoked once when the system starts-up; The /etc/rc.d/rc.local script is invoked at the end of the process and serves to indicate the machine's specific boots.

The real start-up of the services is carried out through the scripts stored in /etc/rc.d/init.d. There is also a link from /etc/init.d. In addition, Fedora provides some useful scripts for handling the services: /sbin/service to halt or start-up a service by the name; and /sbin/chkconfig, to add links to the S and K files that are necessary for a service or to obtain information on the services.

- Debian has management commands for the runlevels such as update-rc.d, that allows us to install or delete services by booting them or halting them in one or more runlevels; invoke-rc.d, allows the classic operations for starting-up, halting or rebooting the service.

The default runlevel in Debian is 2, the X Window System is not managed from /etc/inittab; instead there is a manager (for example, gdm or kdm) that works as if it were another of the services of runlevel 2.

3. Monitoring system state

One of the main daily tasks of the (root) administrator will be to verify that the system works properly and check for any possible errors or saturation of the machine's resources (memory, disks etc.). In the following subsections, we will study the basic methods for examining the state of the system at a determined point in time and how to perform the operations required to avoid any subsequent problems.

In this module's final tutorial, we will carry out a full examination of a sample system, so that we may see some of these techniques.

3.1. System boot

When booting a GNU/Linux system, there is a large extraction of interesting information; when the system starts-up, the screen usually shows the data from the processes detecting the machine's characteristics, the devices, system services boots etc., and any problems that appear are mentioned.

In most distributions, this can be seen directly in the system's console during the booting process. However, either the speed of the messages or some of the modern distributions that hide the messages behind graphics can stop us from seeing the messages properly, which means that we need a series of tools for this process.

Basically, we can use:

- `dmesg` command: shows the messages from the last kernel boot.
- `/var/log/messages` file: general system log that contains the messages generated by the kernel and other daemons (there may be many different log files, normally in `/var/log`, and depending on the configuration of the `syslog` service).
- `uptime` command: indicates how long the system has been active.
- `/proc` system: pseudo file system (`procfs`) that uses the kernel to store the processes and system information.
- `/sys` system: pseudo file system (`sysfs`) that appeared in the kernel 2.6.x branch to provide a more coherent method of accessing the information on the devices and their drivers.

3.2. kernel: /proc directory

When booting up, the kernel starts up a pseudo-file system called /proc, in which it dumps the information compiled on the machine, as well as many other internal data, during the execution. The /proc directory is implemented on memory and not saved to disk. The contained data are both static and dynamic (they vary during execution).

It should be remembered that, as /proc heavily depends on the kernel, the structure tends to depend on the system's kernel and the included structure and files can change.

One of the interesting points is that we can find the images of the processes that are being executed in the /proc directory, along with the information that the kernel handles on the processes. Each of the system's processes can be found in the /proc/<process_pid> directory, where there is a directory with files that represent its state. This information is basic for debugging programs or for the system's own commands such as *ps* or *top*, which can use it for seeing the state of the processes. In general, many of the system's utilities consult the system's dynamic information from /proc (especially some of the utilities provided in the *procps* package).

Note

The /proc directory is an extraordinary resource for obtaining low-level information on the system's working and many system commands rely on it for their tasks.

On another note, we can find other files on the global state of the system in /proc. We will look at some of the files that we can examine to obtain important information briefly:

File	Description
/proc/bus	Directory with information on the PCI and USB buses.
/proc/cmdline	<i>Kernel startup line</i>
/proc/cpuinfo	CPU data
/proc/devices	List of system character devices or block devices
/proc/driver	Information on some hardware kernel modules
/proc/filesystems	Systems of enabled files in the kernel
/proc/ide	Directory of information on the IDE bus, disks characteristics
/proc/interrupts	Map of the hardware interrupt requests (IRQ) used
/proc/ioports	I/O ports used
/proc/meminfo	Data on memory usage
/proc/modules	Modules of the kernel
/proc/mounts	File systems currently mounted

File	Description
/proc/net	Directory with all the network information
/proc/scsi	Directory of SCSI devices or IDEs emulated by SCSI
/proc/sys	Access to the dynamically configurable parameters of the kernel
/proc/version	Version and date of the kernel

As of kernel version 2.6, a progressive transition of procfs (*/proc*) to sysfs (*/sys*) has begun, in order to migrate all the information that is not related to the processes, especially the devices and their drivers (modules of the kernel) to the */sys* system.

3.3. kernel: /sys

The sys system is in charge of making the information on devices and drivers, which is in the kernel, available to the user space so that other APIs or applications can access the information on the devices (or their drivers) in a more flexible manner. It is usually used by layers such as HAL and the udev service for monitoring and dynamically configuring the devices.

Within the sys concept there is a tree data structure of the devices and drivers (let us say the fixed conceptual model) and how it can subsequently be accessed through the sysfs file system (the structure of which may change between different versions).

When an added object is detected or appears in the system, a directory is created in sysfs in the driver model tree (drivers, devices including their different classes). The parent/child node relationship is reflected with subdirectories under */sys/devices/* (reflecting the physical layer and its identifiers). Symbolic links are placed in the */sys/bus* subdirectory reflecting the manner in which the devices belong to the different physical buses of the system. And the devices are shown in */sys/class*, grouped according to their class, for example *network*, whereas */sys/block/* contains the block devices.

Some of the information provided by */sys* can also be found in */proc*, but it was decided that this method involved mixing different elements (devices, processes, data, hardware, kernel parameters) in a manner that was not very coherent and this was one of the reasons that */sys* was created. It is expected that the information will migrate from */proc* to */sys* to centralise the device data.

3.4. Processes

The processes that are executing at a given moment will be of a different nature, generally. We may find:

- **System processes**, whether they are processes associated to the machine's local workings, kernel, or processes (known as daemons) associated to the control of different services. On another note, they may be local or networked, depending on whether the service is being offered (we are acting as a server) or we are receiving the results of the service (we are acting as clients). Most of these processes will appear associated to the root user, even if we are not present at that moment as users. There may be some services associated to other system users (lp, bin, www, mail etc.), which are virtual non-interactive users that the system uses to execute certain processes.
- **Processes of the administering user**: when acting as the root user, our interactive processes or the launched applications will also appear as processes associated to the root user.
- **System users processes**: associated to the execution of their applications, whether they are interactive tasks in text mode or in graphic mode.

We can use the following as faster and more useful:

- *ps*: the standard command, list of processes with the user data, time, process identifier and the command line used. One of the most commonly used options is *ps -ef* (or *-ax*), but there are many options available (see man).
- *top*: one version that provides us with an updated list by intervals, dynamically monitoring the changes. And it allows us to order the list of processes sorted by different categories, such as memory usage, CPU usage, so as to obtain a ranking of the processes that are taking up all the resources. It is very useful for providing information on the possible source of the problem, in situations in which the system's resources are all being used up.
- *kill*: this allows us to eliminate the system's processes by sending commands to the process such as *kill -9 pid_of_process* (9 corresponding to SIGKILL), where we set the process identifier. It is useful for processes with unstable behaviour or interactive programs that have stopped responding. We can see a list of the valid signals in the system with *man 7 signal*

3.5. System Logs

Both the kernel and many of the service daemons, as well as the different GNU/Linux applications or subsystems, can generate messages that are sent to log files, either to obtain the trace of the system's functioning or to detect

errors or fault warnings or critical situations. These types of logs are essential in many cases for administrative tasks and much of the administrator's time is spent processing and analysing their contents.

Most of the logs are created in the `/var/log` directory, although some applications may modify this behaviour; most of the logs of the system itself are located in this directory.

A particular daemon of the system (important) is daemon *Syslogd*. This daemon is in charge of receiving the messages sent by the kernel and other service daemons and sends them to a log file that is located in `/var/log/messages`. This is the default file, but Syslogd is also configurable (in the `/etc/syslog.conf` file), so as to make it possible to create other files depending on the source, according to the daemon that sends the message, thereby sending it to the log or to another location (classified by source), and/or classify the messages by importance (priority level): *alarm*, *warning*, *error*, *critical* etc.

Note

The Syslogd daemon is the most important service for obtaining dynamic information on the machine. The process of analysing the logs helps us to understand how they work, the potential errors and the performance of the system.

Depending on the distribution, it can be configured in different modes by default; in `/var/log` in Debian it is possible to create (for example) files such as: *kern.log*, *mail.err*, *mail.info*... which are the logs of different services. We can examine the configuration to determine where the messages come from and in which files they are saved. An option that is usually useful is the possibility of sending the messages to a virtual text console (in `/etc/syslog.conf` the destination console, such as `/dev/tty8` or `/dev/xconsole`, is specified for the type or types of message), so that we can see the messages as they appear. This is usually useful for monitoring the execution of the system without having to constantly check the log files at each time. One simple modification to this method could be to enter, from a terminal, the following instruction (for the general log):

```
tail -f /var/log/messages
```

This sentence allows us to leave the terminal or terminal window so that the changes that occur in the file will progressively appear.

Other related commands:

- *uptime*: time that the system has been active. Useful for checking that no unexpected system reboot has occurred.
- *last*: analyses the in/out log of the system (`/var/log/wtmp`) of the users, and the system reboots. Or last log control of the last time that the users were seen in the system (information in `/var/log/lastlog`).

- Various utilities for combined processing of logs, that issue summaries (or alarms) of what has happened in the system, such as: logwatch, logcheck (Debian), log_analysis (Debian)...

3.6. Memory

Where the system's memory is concerned, we must remember that we have: a) the physical memory of the machine itself, b) virtual memory that can be addressed by the processes. Normally (unless we are dealing with corporate servers), we will not have very large amounts, so the physical memory will be less than the necessary virtual memory (4GB in 32bit systems). This will force us to use a swap zone on the disk, to implement the processes associated to the virtual memory.

This swap zone may be implemented as a file in the file system, but it is more usual to find it as a *swap* partition, created during the installation of the system. When partitioning the disk, it is declared as a Linux Swap type.

To examine the information on the memory, we have various useful commands and methods:

- */etc/fstab* file: the swap partition appears (if it exists). With an *fdisk* command, we can find out its size (or check */proc/swaps*).
- *ps* command: allows us to establish the processes that we have, with the options on the percentage and memory used.
- *top* command: is a dynamic *ps* version that is updatable by periods of time. It can classify the processes according to the memory that they use or CPU time.
- *free* command: reports on the global state of the memory. Also provides the size of the virtual memory.
- *vmstat* command: reports on the state of the virtual memory and the use to which it is assigned.
- Some packages, like *dstat*, allow us to collate data on the different parameters (memory, swap and others) by intervals of time (similar to *top*).

3.7. Disks and file systems

We will examine which disks are available, how they are organised and which partitions and file systems we have.

When we have a partition and we have a determined accessible file system, we will have to perform a mounting process, so as to integrate it in the system, whether explicitly or as programmed at startup/boot. During the mounting process, we connect the file system associated to the partition to a point in the directory tree.

In order to find out about the disks (or storage devices) present in the system, we can use the system boot information (dmesg), when those available are detected, such as the /dev/hdx for IDE devices or /dev/sdx for SCSI devices. Other devices, such as hard disks connected by USB, flash disks (pen drive types), removable units, external CD-ROMs etc., may be devices with some form of SCSI emulation, so they will appear as devices as this type.

Any storage device will present a series of space partitions. Typically, an IDE disk supports a maximum of four physical partitions or more if they are logical (they permit the placement of various partitions of this type on one physical partition). Each partition may contain different file system types, whether they are of one same operative or different operatives.

To examine the structure of a known device or to change its structure by partitioning the disk, we can use the *fdisk* command or any of its more or less interactive variants (*cfdisk*, *sfdisk*). For example, when examining a sample disk ide /dev/hda, we are given the following information:

```
# fdisk -j /dev/hda
```

```
Disk /dev/hda: 20.5 GB, 20520493056 bytes 255 heads, 63
sectors/track, 2494 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes
```

Device	Boot	Start	End	Blocks	Id System
/dev/hda1	*	1	1305	10482381	7 HPFS/NTFS
/dev/hda2	*	1306	2429	9028530	83 Linux
/dev/hda3		2430	2494	522112+	82 Linux swap

20 GB disk with three partitions (they are identified by the number added to the device name), where we observe two NTFS and Linux-type boot partitions (Boot column with *), which indicates the existence of a Windows NT/2000/XP/Vista along with a GNU/Linux distribution and a last partition that is used as a swap for Linux. In addition, we have information on the structure of the disk and the sizes of each partition.

Some of the disks and partitions that we have, some will be mounted in our file system, or will be ready for set up upon demand, or they may be set up when the resource becomes available (in the case of removable devices).

We can obtain this information in different ways (we will see this in more detail in the final workshop):

- The */etc/fstab* file indicates the devices that are ready to be mounted on booting or the removable devices that may be mounted. Not all of the system devices will appear necessarily; only the ones that we want to appear when booting. We can mount the others upon demand using the *mount* command or remove them with *umount*.
- *mount* command. This informs us of the file systems mounted at that moment (whether they are real devices or virtual file systems such as */proc*). We may also obtain this information from the */etc/mtab* file.
- *df -k* command. This informs us of the storage file systems and allows us to verify the used space and available space. It's a basic command for controlling the available disk space.

With regard to this last *df -k* command, one of our basic tasks as an administrator of the machine is to control the machine's resources and, in this case, the space available in the file systems used. These sizes have to be monitored fairly frequently to avoid a system crash; a file system must never be left at less than 10 or 15% (especially if it is the */*), as there are many process daemons that are normally writing temporary information or logs, that may generate a large amount of information; a particular case is that of the core files that we have already mentioned and which can involve very large files (depending on the process). Normally, some precautions should be taken with regard to system hygiene if any situations of file system saturation are detected:

- Eliminate old temporary files. The */tmp* and */var/tmp* directories tend to accumulate many files created by different users or applications. Some systems or distributions take automatic hygiene measures, such as clearing */tmp* every time the system boots up.
- Logs: avoiding excessive growth, according to the system configuration (for example, *Syslogd*), as the information generated by the messages can be very large. Normally, the system will have to be cleared regularly, when certain amounts of space are taken up and, in any case, if we need the information for subsequent analyses, backups can be made in removable devices. This process can be automated using cron scripts or using specialised tools such as *logrotate*.
- There are other parts of the system that tend to grow a lot, such as: a) user core files: we can delete these periodically or eliminate their generation;

b) the email system: stores all of the emails sent and received; we can ask the users to clean them out regularly or implement a quota system; c) the caches of the browsers or other applications: other elements that usually occupy a lot of space, which require regular clearing, are: d) the accounts of the users themselves: they may have quotas so that pre-established allocated spaces are not exceeded etc.

4. File Systems

In each machine with a GNU/Linux system, we will find different types of file systems [Hin].

To start with, it is typical to find the actual Linux file systems created in various partitions of the disks [Koe]. The typical configuration is to have two partitions: that corresponding to "/" (*root file system*) and that corresponding to the swap file. Although, in more professional configurations, it is usual to separate partitions with "differentiated" parts of the system, a typical technique is, for example (we will examine other options later), to create different partitions so:

```
/ /boot /home /opt /tmp /usr /var swap
```

That will certainly be found mounted from different sources (different disks, or even the network in some cases). The idea is to clearly separate the static and dynamic parts of the system, so as to make it easier to extend the partitions when any overload problems arise. Or more easily isolate the parts to perform backups (for example, the user accounts in the /home partition).

The swap partitions are Linux swap type partitions and that corresponding to / tends to be one of the standard file systems, either ext2 (the default type up to kernels 2.4), or the new ones ext3, ext4, which is an upgrade of ext2 with journaling, which makes it possible to have a log of what goes on in the file system, for faster recoveries in the event of an error. Other file system types, such as Reiser or XFS are also typical.

Another typical configuration may be that of having three partitions: /, swap, /home, in which the /home will be used for the user accounts. This makes it possible to separate the system's user accounts, isolating two separate partitions and allocating the necessary space for the accounts in another partition.

Another configuration that is widely used is that of separating the static parts of the system from the dynamic ones, in different partitions; for example one partition is used for placing / with the static part (/bin /sbin and /usr in some cases), which is not expected to grow or, if it does, not by much, and another or various partitions with the dynamic part (/var /tmp /opt), supposing that /opt, for example, is the installation point for new software. This makes it possible to better adjust the disk space and to leave more space for the parts of the system that need it.

Where the supported file systems are concerned, we must point out the variety of these; we can currently find (among others):

- Systems associated to GNU/Linux, such as the ext2, ext3 and ext4 standards, developed from the previous concept of journaling (support log for operations performed in the file system that allows us to recover it in the event of any disaster that renders it inconsistent).
- Compatibility with non- GNU/Linux environments: MSDOS, VFAT, NTFS, access to the different systems of FAT16, FAT32 and NTFS. In particular, we must point out that the kernel support, in the case of the kernel, is read-only. But, as we have mentioned, there are user space solutions (through FUSE, a kernel module that allows us to write file systems in the user space), that make read/write possible, such as the abovementioned NTFS-3g. There is also compatibility with other environments such as Mac with HFS and HFSplus.
- Systems associated to physical supports, such as CD/DVDs, for example ISO9660 and UDF.
- Systems used in different Unix, which generally provide better performance (sometimes at the cost of a greater consumption of resources, in CPU for example), such as JFS2 (IBM), XFS (SGI), or ReiserFS.
- Network file systems (more traditional): NFS, Samba (smbfs, cifs), permit us to access the file systems available in other machines transparently using the network.
- Systems distributed in the network: such as GFS, Coda.
- Pseudo file systems, such as procfs (/proc) or sysfs (/sys).

Note

The file systems howto document provides brief explanations of the various file systems as well as the websites that you may consult for each of these.

In most of these file systems (except in some special cases), GNU/Linux will allow us to create partitions of these types, build the file systems of the required type and mount them as an integrating part of the directory tree, either temporarily or permanently.

4.1. Mount point

Apart from the /root file system and its possible extra partitions (/usr /var /tmp /home), it should be remembered that it is possible to leave mount points prepared for mounting other file systems, whether they are disk partitions or other storage devices.

In the machines in which GNU/Linux shares the partition with other operating systems, through some bootloader (lilo or grub), there may be various partitions assigned to the different operating systems. It is often good to share

data with these systems, whether for reading or modifying their files. Unlike other systems (that only register their own data and file systems and in some versions of which some of the actual file systems are not supported), GNU/Linux is able to treat, as we have seen, an enormous amount of file systems from different operating systems and to share the information.

Example

If we have installed GNU/Linux in the PCs, we will certainly find more than one operating system, for example, another version of GNU/Linux with ext2 or 3 of the file system, we may find an old MSDOS with its FAT file system, a Windows98/ME/XP Home with FAT32 (or VFAT for Linux), or a Windows NT/2000/XP/Vista with NTFS systems (NTFS for Linux) and FAT32 (VFAT) at the same time.

Our GNU/Linux system can read data (in other words, files and directories) from all these file systems and write in most of them.

In the case of NTFS, up until certain points, there were problems with writing, which was experimental in most of the kernel drivers that appeared. Due mainly to the different versions of the file system that progressively appeared, as there were two main versions called NTFS and NTFS2, and some extensions such as the so-called dynamic volumes or the encrypted file systems. And accessing with certain drivers caused certain incompatibilities, which could result in data corruption or faults in the file system.

Thanks to FUSE, a module integrated in the kernel (as of version 2.6.11), it has been possible to develop the file systems more flexibly, directly in the user space (in fact, FUSE acts as a "bridge" between the kernel requests, and access from the driver).

Thanks to the features of FUSE, we have more or less complete support for NTFS, (provided Microsoft does not make any further changes to the specifications), especially since the appearance of the driver (based on FUSE) ntfs-3g (<http://www.ntfs-3g.org>), and the combination with the ntfsprogs utilities.

Depending on the distribution, different ones are used, or we may also create it ourselves. Normally, they exist either as root subdirectories, for example /cdrom /win /floppy or subdirectories within /mnt, the standard mount point (they appear as /mnt/cdrom /mnt/floppy...), or the /media directory, which is lately preferred by the distributions. According to the FHS standard, /mnt should be used for temporary mounting of file systems, whereas /media should be used to mount removable devices.

The mounting process is performed through the mount command, with the following format:

```
mount -t filesystem-type device mount-point
```

The type of file system may be: MSDOS (FAT), VFAT (FAT32), NTFS (NTFS read), ISO9660 (for CD-ROM)... (of the possible ones).

The device is the in point in the /dev directory corresponding to the location of the device, the IDEs had /dev/hdxy where x is a,b,c, or d (1 master, 1 slave, 2 master, 2 slave) e and, the partition number, the SCSI (/dev/sdx) where x is a,b,c,d ... (according to the associated SCSI ID, 0,1,2,3,4 ...).

We will now look at some examples:

```
mount -t iso9660 /dev/hdc /mnt/cdrom
```

This would mount the CD-ROM (if it is the IDE that is in the second IDE in master mode) at point /mnt/cdrom.

```
mount -t iso9660 /dev/cdrom /mnt/cdrom
```

This would mount the CD-ROM; /dev/cdrom is used as a synonym (it is a link) for the device where it is connected.

```
mount -t vfat /dev/fd0H1440 /mnt/floppy
```

This would mount the diskette, /dev/fd0H1440. It would be a high-density (1.44 MB) disk drive A; /dev/fd0 can also be used.

```
mount -t ntfs /dev/hda2 /mnt/winXP
```

This would mount the second partition of the first NTFS-type IDE device (C:) (for example, a Windows XP).

If these partitions are more or less stable in the system (in other words, they are not changed frequently) and we wish to use them, the best thing will be to include the mounts so that they take place during the execution period, when booting the system, through the configuration of file /etc/fstab:

```
# /etc/fstab: Static information on the file system
#
```

#<Sys. files>	<Mount point>	<Type>	<Options>	<dump>	<pass>
/dev/hda2	/	ext3	errors = remountro	0	1
/dev/hdb3	none	swap	sw	0	0
proc	/proc	proc	defaults	0	0
/dev/fd0	/floppy	auto	user,noauto	0	0
/dev/cdrom	/cdrom	iso9660	ro,user,noauto	0	0

/dev/sdb1	/mnt/usb	vfat	user,noauto	0	0

For example, this configuration includes some of the standard systems, such as the root in /dev/hda2, the swap partition that is in hdb3, the proc system (which uses the kernel to save its information). The diskette, the CD-ROM and, in this case, a Flash-type USB disk (which is detected as a SCSI device). In some cases, auto is specified as a type of file system. This permits the autodetection of the file system. If unknown, it is better to indicate it in the configuration and, on another note, the noauto option will mean that it is not always mounted automatically, but upon request (or access).

If we have this information in the file, the mounting process is simplified, as it will take place either on execution, when booting up, or upon demand (noautos). And it may now be performed by simply asking that the mount point or device be mounted:

```
mount /mnt/cdrom
mount /dev/fd0
```

given that the system already has the rest of the information.

The reverse process, unmounting, is quite simple, the umount command with the mount point or device:

```
umount /mnt/cdrom
umount /dev/fd0
```

When using removable devices, such as CD-ROMs (or others), eject may be used to extract the physical support:

```
eject /dev/cdrom
```

or, in this case, only:

```
eject
```

The mount and umount commands mount or umount all the available systems. The file */etc/mtab* maintains a list of the mounted systems at a specific point in time, which can be consulted, or a mount, without parameters, may be executed to obtain this information.

4.2. Permissions

Another subject that we will have to control in the cases of files and directories is the permissions that we wish to establish for each of them, whilst remembering that that each file may have a series of permissions: rwxrwxrwx

where they correspond with the owner rwx, the group rwx to which the user belongs, and the rwx for other users. In each one, we may establish the access rights for reading (r), writing (w) or executing (x). In the case of a directory, x denotes the permission for being able to access that directory (with the cd command, for example).

In order to modify the access rights to a directory or file, we have the commands:

- chown: change file owner.
- chgrp: change file owner group.
- chmod: change specific permissions (rwx) of the files.

The commands also provide the -R option, which is recursive if affecting a directory.

5. Users and groups

The users of a GNU/Linux system normally have an associated account (defined with some of their data and preferences) along with an allocated amount of space on the disk in which they can develop their files and directories. This space is allocated to the user and may only be used by the user (unless the permissions specify otherwise).

Among the accounts associated to users, we can find different types:

- The administrator account, with the root identifier, which should only be used for administration operations. The root user is the one with most permissions and complete access to the machine and the configuration files. Consequently, this user is also the one that most damage can cause due to any faults or omissions. It is better to avoid using the root account as if it were that of just another user; it is therefore recommended that it should only be used for administration operations.
- User accounts: the normal accounts for any of the machine's users have the permissions restricted to the use of their account files and to some particular zones (for example, the temporary files in /tmp), and to the use of the particular devices that they have been authorised to use.
- Special service accounts: lp, news, wheel, www-data... accounts that are not used by people but by the system's internal services, which uses them under these user names. Some of the services are also used under the root account.

A user account is normally created by specifying a name (or user identifier), a password and a personal associated directory (the account).

The information on the system's users is included in the following files:

```
/etc/passwd  
/etc/shadow  
/etc/group  
/etc/gshadow
```

Example of some lines of the /etc/passwd:

```
juan:x:1000:1000:Juan Garcia,,,:/home/juan:/bin/bash  
root:x:0:0:root:/root:/bin/bash
```

where (if the :: appear together, the box is empty):

- `juan`: identifier of the user of the system.
- `x`: encoded user password; if there is an "x" then it is located in the `/etc/shadow` file.
- `1000`: user code, which the system uses as the identity code of the user.
- `1000`: code of the main group to which the user belongs, the group's information is in `/etc/group`.
- `Juan García`: comment, usually the user's full name.
- `/home/juan`: personal directory associated to his account.
- `/bin/bash`: interactive shell that the user uses when interacting with the system, in text mode, or through the graphic shell. In this case, the GNU Bash, which is the shell used by default. The `/etc/passwd` file used to contain the user passwords in an encrypted form, but the problem was that any user could see this file and, at the time, cracks were designed to try and find out the passwords directly using the encrypted password as the starting point (word encoded with the crypt system).

In order to avoid this, the passwords are no longer placed in this file; only an "x" is, to indicate that they are located in another file, which can only be read by the root user, `/etc/shadow`, the contents of which may be something similar to the following:

```
juan:algNcs82ICst8CjVJS7ZFCVnu0N2pBcn/:12208:0:99999:7:::
```

where the user identifier is located, along with the encrypted password. In addition, they appear as spaces separated by ":":

- Days since 1st January 1970 in which the password was changed for the last time.
- Days left for it to be changed (0 it does not have to be changed).
- Days after which the password must be changed (in other words, change period).
- Days on which the user will be warned before the password expires.
- Days, after expiry, after which the account will be disabled.
- Days since 1st January 1970 that the account has been disabled.
- And a reserved space.

In addition, the encryption codes can be more difficult, as it is now possible to use a system called md5 (it usually appears as an option when installing the system) to protect the users' passwords. We will examine some more details in the unit on security.

In */etc/group* we will find the information on the user groups:

```
jose:x:1000:
```

where we have:

```
name-group:password-group:identifier-of-group:list-users
```

The list of the users in the group may or may not be present; given that this information is already in */etc/passwd*, it is not usually placed in */etc/group*. If it is placed there, it usually appears as a list of users separated by commas. The groups may also possess an associated password (although this is not that common), as in the case of the user, there is also a shadow file: */etc/gshadow*.

Other interesting files are the ones in */etc/skel* directory, which contains the files that are included in each user account when it is created. We must remember that, as we saw with the interactive shells, we could have some configuration scripts that execute when we enter or exit the account. The "skeletons", which are copied in user account when they are created, are saved in the *skel* directory. The administrator is usually in charge of creating adequate files for the users, providing the necessary execution paths, initialising the system's variables that are needed for the software etc.

We will now see a series of useful commands for the administration of users (we will mention their functions and perform some tests in the workshop):

- *useradd*: adding a user to the system.
- *userdel*: to delete a user from the system.
- *usermod*: to modify a user of the system.
- *groupadd*, *groupdel*, *groupmod* the same for groups.
- *newusers*, *chpasswd*: these can be very useful in large installations with many users, as they allow us to create various accounts from the information entered into a *newusers* file or change the passwords for a large number of users (*chpasswd*).
- *chsh*: to change the user login shell.

- *chfn*: to change the user information present in the `/etc/passwd` comment file.
- *passwd*: to change a user's password. This may be executed as a user, and it will then ask for the old password and the new one. When doing this, the root account has to specify the user whose password will be changed (otherwise, they would be changing the account's password) and the old password is not necessary. This is perhaps the command that the root most uses, when users forget their old password.
- *su*: a kind of identity change. It is used both by users and by the root to change the current user. In the case of the administrator, it is used quite a lot to test that the user account works properly; there are different variants: *su* (without parameters, it serves to switch to root user, after identification, making it possible for us to pass, when we are in a user account, to the root account to perform a task). The *su iduser* sentence (changes the user to *iduser*, but leaves the environment as it is, in other words, in the same directory...). The *su - iduser* mandate (which performs a complete substitution, as if the second user had logged in the system).

With regard to the administration of users and groups, what we have mentioned here refers to the local administration of one sole machine. In systems with multiple machines that the users share, a different management system is used for the information on users. These systems, generically called network information systems, such as NIS, NIS+ or LDAP, use databases for storing the information on the users and groups, effectively using servers, where the database and other client machines are stored and where this information can be consulted. This makes it possible to have one single copy of the user data (or various synchronised copies) and makes it possible for them to enter any available machine of the set administered with these systems. At the same time, these systems incorporate additional concepts of hierarchies and/or domains/machine and resource zones, that make it possible to adequately represent the resources and their use in organisations with different organisational structures for their own personnel and internal departments.

We can check whether we are in a NIS-type environment by seeing if *compat* appears in the `passwd` line and group configuration file, `/etc/nsswitch.conf`, if we are working with local files, or `nis` or `nisplus` according to the system on which we are working. Generally, this does not involve any modification for the simple user, as the machines are managed transparently, more so if it is combined with files shared by NFS that makes the account available, regardless of the machine used. Most of the abovementioned commands can still be used without any problem under NIS or NIS+, in which they are equivalent, except for the command for changing the password, which, instead of *passwd*, we

usually use *yppasswd* (NIS) or *nispasswd* (NIS+); although it is typical for the administrator to rename them to *passwd*, (through a link), which means that users will not notice the difference.

We will look at this and other methods for configuring the network administration units.

6. Printing services

The GNU/Linux [Gt] [Smi02] printing server derives from UNIX's BSD variant; this system was called LPD (*line printer daemon*). This is a very powerful printing system, because it integrates the capacity to manage both local and network printers. And it provides this service within the system for both the client and the printing server.

LPD is a system that is quite old, as its origins date back to UNIX's BSD branch (mid 1980s). Consequently, LPD usually lacks support for modern devices, given that the system was not originally conceived for the type of printing that takes place now. The LPD system was not designed as a system based on device drivers, as it was typical to produce only printers in series or in parallel for writing text characters.

Currently, the LPD system combines with another common software, such as the Ghostscript system, which offers a postscript type output for a very wide range of printers for which it has the right drivers. At the same time, they are usually combined with filtering software, which, depending on the type of document that must be printed, selects the appropriate filters. Normally, the procedure that should be followed is (basically):

- 1) The work is started by a command in the LPD system.
- 2) The filtering system identifies the type of job (or file) that must be used and transforms the job into an outgoing postscript file, which is the one sent to the printer. In GNU/Linux and UNIX, most of the applications assume that the job will be sent to a postscript printer and many of them directly generate a postscript output, which is why the following step needs to be taken.
- 3) The Ghostscript has to interpret the postscript file it receives, and, depending on the driver of the printer to which the file has been sent, it performs the transformation to the driver's own format. If the printer is a postscript type printer, the printing process is direct; if not, it has to "translate" the job. The job is sent to the printing queue.

Apart from the LPD printing system (that originated with UNIX's BSD), there is also the system known as System V (originally in the other System V branch of UNIX). Normally, for compatibility reasons, most UNIX systems integrate both systems, so that either one or the other is used as the main one and the other emulates the main one. In the case of GNU/Linux, a similar process occurs, depending on the installation that we have, we can have only the LPD commands of the printing system, but it will also be common to have the

Note

The UNIX systems provide, possibly, the most powerful and complex printing systems, which provide a lot of flexibility to printing environments.

Web site

Ghostscript: <http://www.ghostscript.com/>

System V commands. A simple way of identifying the two systems (BSD or System V) is using the main printing command (which sends the jobs to the system), in BSD, it is `lpr`, and it is `lp` in System V.

This is the initial situation for the GNU/Linux printing systems, although over the last few years, more systems have appeared, which provide more flexibility and make more drivers available for the printers. The two main systems are CUPS and, to a lesser extent, LPRng. In fact, recently, CUPS is GNU/Linux's de facto standard, although the other systems must be supported for compatibility with the existing UNIX systems.

Web sites

LPRng: <http://www.lprng.org>
CUPS: <http://www.cups.org>

Both (both CUPS and LPRng) are a type of higher-level system, but they are not all that perceptibly different for average users, with regard to the standard BSD and System V systems; for example, the same client commands (or compatible commands in the options) are used for printing. There are perceptible differences for the administrator, because the configuration systems are different. In one way, we can consider LPRng and CUPS as new architectures for printing systems, which are compatible for users with regard to the old commands.

In the current GNU/Linux distributions, we can find different printing systems. If the distribution is old, it may only incorporate the BSD LPD system; in the current distributions: both Debian and Fedora/Red Hat use CUPS. In older versions of Red Hat, there was a tool, Print switch, which made it possible to change the system, switching the printing system, although recently only CUPS is available. In Debian, it is possible to install both systems, but they are mutually exclusive: only one may be used for printing.

In the case of Fedora Core, the default printing system is CUPS (as LPRng disappeared in Fedora Core 4), and the Print Switch tool no longer exists, as it is no longer necessary: `system-config-printer` is used to configure devices. By default, Debian uses BSD LPD, but it is common to install CUPS (and we can expect it to continue to be the default option in future new versions) and LPRng may also be used. In addition, we must remember that we also had the possibility (seen in the unit on migration) of interacting with Windows systems through the Samba protocols, which allowed you to share printers and access to these printers.

Regarding each of the [Gt] systems:

- **BSD LPD:** this is one of UNIX's standards, and some applications assume that the commands and the printing system will be available, for which both LPRng and CUPS emulate the functions and commands of BSD LPD. The LPD system is usable but not very configurable, especially with regard to access control, which is why the distributions have been moved to other, more modern, systems.

- LPRng: basically it was designed to replace BSD, and therefore, most of the configuration is similar and only some of the configuration files are different.
- CUPS: it is the biggest deviation from the original BSD and the configuration is the same. Information is provided to the applications on the available printers (also in LPRng). In CUPS, both the client and the server have to have CUPS software.

The two systems emulate the printing commands of System V.

For GNU/Linux printing, various aspects have to be taken into account:

- Printing system that is used: BSD, LPRng or CUPS.
- Printing device (printer): it may have a local connection to a machine or be on the network. The current printers may be connected to a machine using local connections, through interfaces in series, in parallel, USB etc. Or they may simply be on the network, as another machine, or with special ownership protocols. Those connected to the network can normally act themselves as a printing server (for example, many HP laser printers are BSD LPD servers) or they can be connected to a machine that acts as a printing server for them.
- Communication protocols used with the printer or the printing system: whether it is direct TCP/IP connection (for example, an HP with LPD) or high level ones based on TCP/IP, such as IPP (CUPS), JetDirect (some HP printers) etc. This parameter is important, as we have to know it so as to install the printer in a system.
- Filtering systems used: each printing system supports one or more.
- Printer drivers: in GNU/Linux, there are quite a few different types; we might mention, for example CUPS drivers, the system's or third parties' (for example, HP and Epson provide them); Gimp, the image editing program also has drivers optimised for printing images; Foomatic is a driver management system that works with most systems (CUPS, LPD, LPRng and others); Ghostscript drivers etc. In almost all printers, there are one or more of the drivers in these sets.

Web site

Information on the most appropriate printers and drivers can be found at: http://www.openprinting.org/printer_list.cgi

With regard to the client part of the system, the basic commands are the same for the different systems and these are the BSD system commands (each system supports emulation of these commands):

- lpr: a job is sent to the default printing queue (or the one that is selected), and the printing daemon (lpd) then sends it to the corresponding queue and assigns a job number, which will be used with the other commands.

Normally, the default printer would be indicated by the PRINTER system variable or the first defined and existing one will be used or, in some systems, the Ip queue will be used (as the default name).

Example

Lpr example:

```
lpr -Pepson data.txt
```

This command sends the data.txt file to the print queue associated to a printer that we have defined as "epson".

- **lpq:** This allows us to examine the jobs in the queue.

Example

Example

# lpq -P epson				
Rank	Owner	Job Files	Total	Size
1st	juan	15	data.txt	74578 bytes
2nd	marta	16	fpppp.F	12394 bytes

This command shows us the jobs in the queue, with the respective order and sizes; the files may appear with different names, as this depends on whether we have sent them with lpr or with another application that might change the names when it sends them or if any filters have had to be used when converting them.

- **lprm:** eliminates jobs from the queue and we can specify a job number or the user, to cancel these operations.

Example

```
lprm -Pepson 15
```

Delete the job with id 15 from the queue.

With regard to the administrative side (in BSD), the main command would be *lpc*; this command can be used to activate or deactivate queues, move jobs in the queue order and activate or deactivate the printers (jobs may be received in the queues but they are not sent to the printers).

We should also point out that, in the case of System V, the printing commands are usually also available, normally simulated on the basis of the BSD commands. In the client's case, the commands are: lp, lpstat, cancel and, for administrative subjects, lpadmin, accept, reject, lpmove, enable, disable, lpshut.

In the following sections we will see that it is necessary to configure a printer server for the three main systems. These servers may be used both for local printing and for the network clients' prints (if they are enabled).

6.1. BSD LPD

In the case of the BSD LPD server, there are two main files that have to be examined: on the one hand, the definition of the printers in `/etc/printcap` and, on the other, the network access permissions in `/etc/hosts.lpd`.

With regard to the permissions, by default, BSD LPD only provides local access to the printer and, therefore, it has to be expressly enabled in `/etc/hosts.lpd`.

Example

The file may be:

```
#file hosts.lpd
second
first.the.com
192.168.1.7
+@groupnis
-three.the.com
```

which would indicate that it is possible to print to a series of machines, listed either by their DNS name or by the IP address. Machine groups that belong to a NIS server (groupnis, as shown in the example) may be added or it is possible to deny access to several machines by indicating this with a dash (-).

With regard to the configuration of the server in `/etc/printcap`, we define inputs, in which each represents a printing system queue that can be used to stop the printing jobs. The queue may be associated to a local device or a remote server, whether this is a printer or another server.

The following options may exist in each port:

- `lp =`, indicates the device to which the printer is connected, for example, `lp = /dev/lp0` would indicate the first parallel port. If the printer is an LPD-type printer, for example, a network printer that accepts the LPD protocol (such as an HP), then we can leave the box empty and fill in the following.
- `rm =`, address with name or IP of the remote machine that will use the printing queue. If it is a network printer, it will be this printer's address.
- `rp =`, name of the remote queue, in the machine indicated before with `rm`.

Let us examine an example::

```
# Local printer input
lp|epson|Epson C62:\
```

```

:lp=/dev/lp1:sd=/var/spool/lpd/epson:\
:sh:pw#80:pl#72:px#1440:mx#0:\
:if = /etc/magicfilter/StylusColor@720dpi-filter:\filter
:af = /var/log/lp-acct:lf = /var/log/lp-errs:
# Remote printer input
hpremove|hpr|remote hp of the department|:\
:lp = :\
:rm = server:rp = queuehp:\
:lf = /var/adm/lpd_rem_errs:\log file.
:sd = /var/spool/lpd/hpremove:local associated spool

```

6.2. LPRng

In the case of the LPRng system, as this was made to maintain BSD compatibility, and, among other improvements with regard to access, the system is compatible in terms of the configuration of queues and this is performed through the same file format, `/etc/printcap`, with some additional intrinsic operations.

Where the configuration is different is with regard to access: in this case, we generally obtain access through a `/etc/lpd.perms` file that is general for the whole system and there may also be individual configurations for each queue with the `lpd.perms` file placed in the directory corresponding to the queue, usually `/var/spool/lpd/name-queue`.

These `lpd.perms` files have a greater capacity for configuring the access and permit the following basic commands:

```

DEFAULT ACCEPT
DEFAULT REJECT
ACCEPT [ key = value[,value] * ] *
REJECT [ key = value[,value] * ] *

```

where the first two allow us to establish the default value, of accepting everything or rejecting everything, and the next two of accepting or rejecting a specific configuration in the line. It is possible to accept (or reject) requests from a specific host, user or IP port. Likewise, it is possible to configure the type of service that will be provided to the element: X (may be connected), P (job printing), Q (examine queue with `lpq`), M (remove jobs from the queue, `lprm`), C (control printers, `lpc` command `lpc`), among others, as with the file:

```

ACCEPT SERVICE = M HOST = first USER = jose
ACCEPT SERVICE = M SERVER REMOTEUSER = root
REJECT SERVICE = M

```

Deleting jobs from the queue is allowed for the (first) user of the machine and the root user from the server where the printing service is hosted (*localhost*) and, in addition, whatsoever other requests for deleting jobs from the queue that are not the already established are rejected.

With this configuration, we have to be very careful, because in some distributions, the LPRng services are open by default. The connection may be limited, for example, with:

```
ACCEPT SERVICE = X SERVER
REJECT SERVICE = X NOT REMOTEIP = 100.200.0.0/255
```

Connection service only accessible to the server's local machine and denying access if the machine does not belong to our subnet (in this case, we are assuming that it is 100.200.0.x).

For the administration of line commands, the same tools as the standard BSD are used. With regard to the graphical administration of the system, we should point out the *lprngtool* tool (not available in all versions of the LPRng system).

Names (name alias1 ...)	lp	?
Comments		?
Spool Directory	/var/spool/lpd/%P	?
Hostname/IP of Printer	h14	?
Port number	9100	?
IFHP	User Specified	Filter /usr/libexec/filters/ifhp
Select Printer Model and Filter Options		default
Job Options	Select LPR Job and Filter Options	landscape
Printcap for:		?
☒ Server and Client (BOTH) ☐ Server Only (:server) ☐ Client Only (:client)		
Spool action:		
☒ Localhost (:force_localhost) ☐ Remote Queue or Device (:force_localhost@) ☒ Default		
Printer Type:		?
☐ Device ☐ Queue ☒ TCP/IP Socket ☐ SMB/Novell/AppleTalk		
☐ Load Balance ☐ Dummy ☐ Unknown		
OK		Cancel
		Advanced Options

Figure 1. lprngtool, configuration of a printer

There are various software packages related to LPRng; for example, in a Debian, we might find:

```
lprng - lpr/lpd printer spooling system
lprng-doc - lpr/lpd printer spooling system (documentation)
lprngtool - GUI front-end to LPRng based /etc/printcap
printop - Graphical interface to the LPRng print system.
```

6.3. CUPS

CUPS is a new architecture for the printing system that is quite different; it has a layer of compatibility with BSD LPD, which means that it can interact with servers of this type. It also supports a new printing protocol called IPP (based on http), but it is only available when the client and the server are CUPS-type clients and servers. In addition, it uses a type of driver called PPD that identifies the printer's capacities; CUPS comes with some of these drivers and some manufacturers also offer them (HP and Epson).

CUPS has an administration system that is completely different, based on different files: `/etc/cups/cupsd.conf` centralises the configuration of the printing system, `/etc/cups/printers.conf` controls the definition of printers and `/etc/cups/classes.conf` the printer groups.

In `/etc/cups/cupsd.conf`, we can configure the system according to a series of file sections and the directives of the different actions. The file is quite big; we will mention some important directives:

- **Allow:** this permits us to specify which machines may access the server, either in groups or individually, or segments of the network's IP.
- **AuthClass:** makes it possible to indicate whether the user clients will be asked to authenticate their accounts or not.
- **BrowseXXX:** there is a series of directives related to the possibility of examining a network to find the served printers; this possibility is activated by default (*browsing on*), which means that we will normally find that all the printers available in the network are available. We can deactivate it, so that we only see the printers that we have defined. Another important option is **BrowseAllow**, which we use to determine who is permitted to ask for our printers; it is activated by default, which means that anyone can see our printer from our network.

We must point out that CUPS is, in principle, designed so that both clients and the server work under the same system; if the clients use LPD or LPRng, it is necessary to install a compatibility daemon called `cups-lpd` (normally in packages such as `cupsys-bsd`). In this case, CUPS accepts the jobs that come from an LPD or LPRng system, but it does not control the accesses (`cupsd.conf` only works for the CUPS system itself and therefore, it will be necessary to implement some strategy for controlling access, like a firewall, for example (see unit on security).

For administering from the commands line, CUPS is somewhat peculiar, in that it accepts both LPD and System V commands in the clients, and the administration is usually performed with the SystemV's `lpadmin` command.

Where the graphic tools are concerned, we have the `gnome-cups-manager`, `gtklp` or the web interface which comes with the same CUPS system, accessible at `http://localhost:631`.



Figure 2. Interface for the administration of the CUPS system

With regard to the software packages listed with CUPS, in Debian, we can find (among others):

```
cupsys - Common UNIX Printing System(tm) - server
cupsys-bsd - Common UNIX Printing System(tm) - BSD commands
cupsys-client - Common UNIX Printing System(tm) - client programs (SysV)
cupsys-driver-gimpprint - Gimp-Print printer drivers for CUPS
cupsys-pt - Tool for viewing/managing print jobs under CUPS
cupsomatic-ppd - linuxprinting.org printer support - transition package
foomatic-db - linuxprinting.org printer support - database
foomatic-db-engine - linuxprinting.org printer support - programs
foomatic-db-gimp-print - linuxprinting - db Gimp-Print printer drivers
foomatic-db-hpijs - linuxprinting - db HPIJS printers
foomatic-filters - linuxprinting.org printer support - filters
foomatic-filters-ppds - linuxprinting - prebuilt PPD files
foomatic-gui - GNOME interface for Foomatic printer filter system
```

gimpprint-doc - Users' Guide for GIMP-Print and CUPS
gimpprint-locals - Local data files for gimp-print
gnome-cups-manager - CUPS printer admin tool for GNOME
gtklp - Front-end for cups written in gtk

7. Disk management

In respect of the storage units, as we have seen, they have a series of associated devices, depending on the type of interface:

- IDE: devices
/dev/had master disk, first IDE connector;
/dev/hdb slave disk of the first connector,
/dev/hdc master second connector,
/dev/hdd slave second connector.
- SCSI: /dev/sda, /dev/sdb devices... following the numbering of the peripheral devices in the SCSI Bus.
- Diskettes: /dev/fdx devices, with x diskette number (starting in 0). There are different devices depending on the capacity of the diskette, for example, a 1.44 MB diskette in disk drive A would be /dev/fd0H1440.

With regard to the partitions, the number that follows the device indicates the partition index within the disk and it is treated as an independent device: /dev/hda1 first partition of the first IDE disk, or /dev/sdc2, second partition of the third SCSI device. In the case of the IDE disks, these allow four partitions, known as primary partitions, and a higher number of logical partitions. Therefore, if /dev/hdan, n is less than or equal to 4, then it will be a primary partition; if not, it will be a logical partition with n being higher than or equal to 5.

With the disks and the associated file systems, the basic processes that we can carry out are included in:

- Creation of partitions or modification of partitions. Through commands such as fdisk or similar (*cfdisk*, *sfdisk*).
- Formatting diskettes: different tools may be used for diskettes: *fdformat* (low-level formatting), *superformat* (formatting at different capacities in MSDOS format), *mformat* (specific formatting creating standard MSDOS file systems).
- Creation of Linux file systems, in partitions, using the *mkfs* command. There are specific versions for creating diverse file systems, *mkfs.ext2*, *mkfs.ext3* and also non-Linux file systems: *mkfs.ntfs*, *mkfs.vfat*, *mkfs.msdos*, *mkfs.minix*, or others. For CD-ROMs, commands such as *mkisofs* for creating the ISO9660s (with joliet or rock ridge extensions), which may be an image that might subsequently be recorded on

a CD/DVD, which along with commands such as `cdrecord` will finally allow us to create/save the CD/DVDs. Another particular case is the `mkswap` order, which allows us to create swap areas in the partitions, which will subsequently be activated or deactivated with `swapon` and `swapoff`.

- Setting up file systems: *mount*, *umount*. commands
- Status verification: the main tool for verifying Linux file systems is the *fsck* command. This command checks the different areas of the file system to verify the consistency and check for possible errors and to correct these errors where possible. The actual system automatically activates the command on booting when it detects situations where the system was not switched off properly (due to a cut in the electricity supply or an accidental shutting down of the machine) or when the system has been booted a certain number of times; this check usually takes a certain amount of time, usually a few minutes (depending on the size of the data). There are also particular versions for other file systems: *fsck.ext2*, *fsck.ext3*, *fsck.vfat*, *fsck.msdos* etc. The *fsck* process is normally performed with the device in read only mode with the partitions mounted; it is advisable to unmount the partitions for performing the process if errors are detected and it is necessary to correct the errors. In certain cases, for example, if the system that has to be checked is the root system (/) and a critical error is detected, we will be asked to change the system's runlevel execution mode to the root execution mode and to perform the verification process there. In general, if it is necessary to verify the system; this should be performed in superuser mode (we can switch between the runlevel mode with the *init* or *telinit* commands).
- Backup processes: whether in the disk, blocks of the disk, partitions, file systems, files... There are various useful tools for this: *tar* allows us to copy files towards file or tape units; *cpio*, likewise, can perform backups of files towards a file; both *cpio* and *tar* maintain information on the permissions and file owners; *dd* makes it possible to make copies, whether they are files, devices, partitions or disks to files; it is slightly complex and we have to have some low-level information, on the type, size, block or sector, and it can also be sent to tapes.
- Various utilities: some individual commands, some of which are used by preceding processes to carry out various treatments: *badblocks* for finding defective blocks in the device; *dumpe2fs* for obtaining information on Linux file systems; *tune2fs* makes it possible to carry out Linux file system tuning of the *ext2* or *ext3* type and to adjust different performance parameters.

We will now mention two subjects related to the concept of storage space, which are used in various environments for the basic creation of storage space. The use of RAID software and the creation of dynamic volumes.

7.1. RAID software

The configuration of disks using RAID levels is currently one of the most widely-used high-availability storage schemes, when we have various disks for implementing our file systems.

The main focus on the different existing techniques is based on a fault-tolerance that is provided from the level of the device and the set of disks, to different potential errors, both physical or in the system, to avoid the loss of data or the lack of coherence in the system. As well as in some schemes that are designed to increase the performance of the disk system, increasing the bandwidth of these available for the system and applications.

Today we can find RAID in hardware mainly in corporate servers (although it is beginning to appear in desktops), where there are different hardware solutions available that fulfil these requirements. In particular, for disk-intensive applications, such as audio and/or video streaming, or in large databases.

In general, this hardware is in the form of cards (or integrated with the machine) of RAID-type disk drivers, which implement the management of one or more levels (of the RAID specification) over a set of disks administered with this driver.

In RAID a series of levels (or possible configurations) are distinguished, which can be provided (each manufacturer of specific hardware or software may support one or more of these levels). Each RAID level is applied over a set of disks, sometimes called RAID array (or RAID disk matrix), which are usually disks with equal sizes (or equal to group sizes). For example, in the case of an array, four 100 GB disks could be used or, in another case, 2 groups (at 100 GB) of 2 disks, one 30 GB disk and one 70 GB disk. In some cases of hardware drivers, the disks (or groups) cannot have different sizes; in others, they can, but the array is defined by the size of the smallest disk (or group).

We will describe some basic concepts on some levels in the following list (it should be remembered that, in some cases, the terminology has not been fully accepted, and it may depend on each manufacturer):

- RAID 0: The data are distributed equally between one or more disks without information on parity or redundancy, without offering fault-tolerance. Only data are being distributed; if the disk fails physically, the information will be lost and we will have to recover it from the backup copies. What does increase is the performance, depending on the RAID 0 imple-

mentation, given that the read and write options will be divided among the different disks.

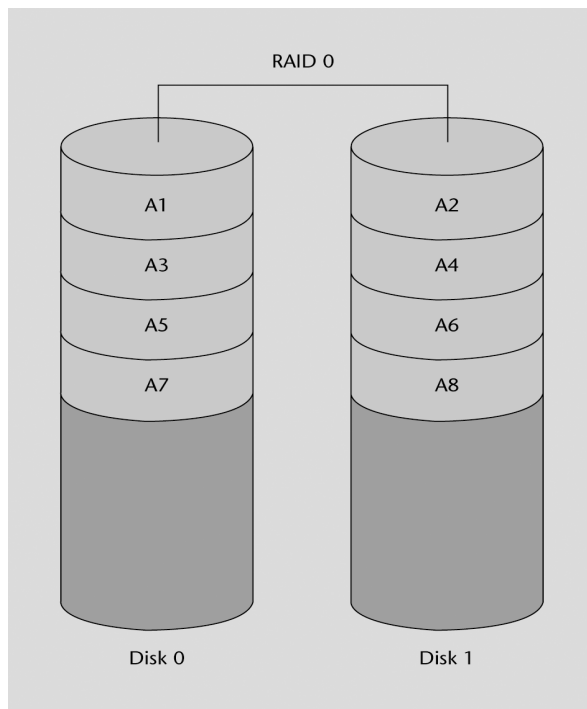


Figure 3

- RAID 1: An exact (mirror) copy is created in a set of two or more disks (known as a RAID array). In this case, it is useful for the reading performance (which can increase lineally with the number of disks) and especially for having a tolerance to faults in one of the disks, given that (for example, with two disks) the same information is available. RAID 1 is usually adequate for high-availability, such as 24x7 environments, where we critically need the resources. This configuration also makes it possible (if the hardware supports this) to hot swap disks. If we detect a fault in any of the disks, we can replace the disk in question without switching off the system with another disk.

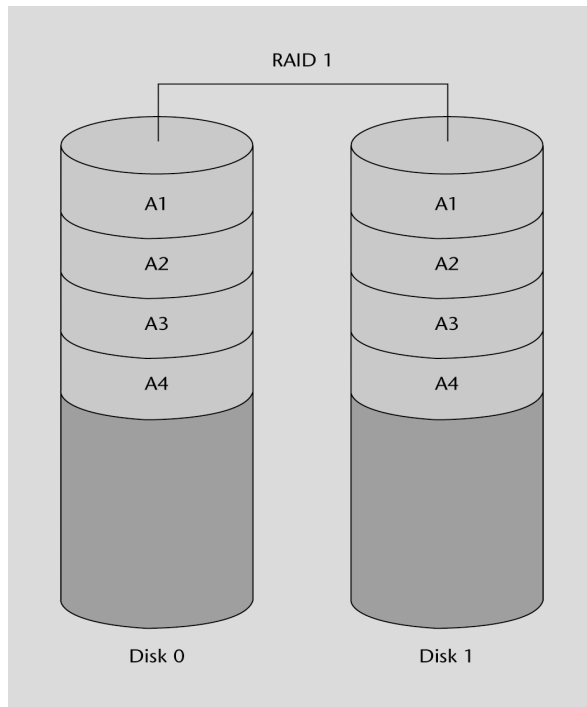


Figure 4

- RAID 2: In the preceding systems, the data would be divided in blocks for subsequent distribution; here, the data are divided into bits and redundant codes are used to correct the data. It is not widely used, despite the high performance levels that it could provide, as it ideally requires a high number of disks, one per data bit, and various for calculating the redundancy (for example, in a 32 bit system, up to 39 disks would be used).
- RAID 3: It uses byte divisions with a disk dedicated to the parity of blocks. This is not very widely used either, as depending on the size of the data and the positions, it does not provide simultaneous accesses. RAID 4 is similar, but it stripes the data at the block level, instead of byte level, which means that it is possible to service simultaneous requests when only a single block is requested.
- RAID 5: Block-level striping is used, distributing the parity among the disks. It is widely used, due to the simple parity scheme and due to the fact that this calculation is implemented simply by the hardware, with good performance levels.

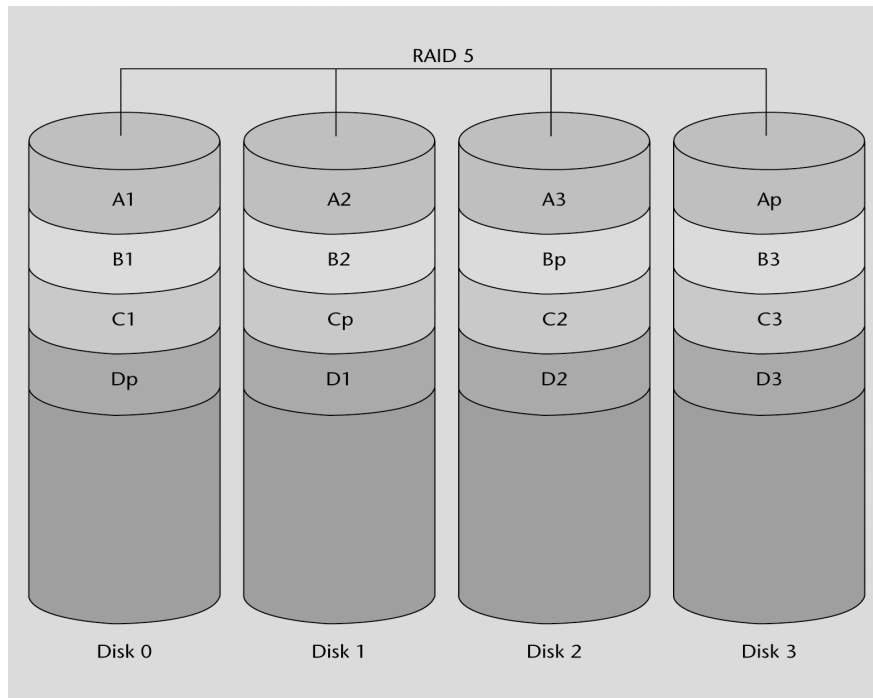


Figure 5

- RAID 0+1 (or O1): A mirror stripe is a nested RAID level; for example, we implement two groups of RAID 0, which are used in RAID 1 to create a mirror between them. An advantage is that, in the event of an error, the RAID 0 level used may be rebuilt thanks to the other copy, but if more disks need to be added, they have to be added to all the RAID 0 groups equally.
- RAID 10 (1+0): striping of mirrors, groups of RAID 1 under RAID 0. In this way, in each RAID 1 group, a disk may fail without ensuing loss of data. Of course, this means that they have to be replaced, otherwise the disk that is left in the group becomes another possible error point within the system. This configuration is usually used for high-performance databases (due to the fault tolerance and the speed, as it is not based on parity calculations).

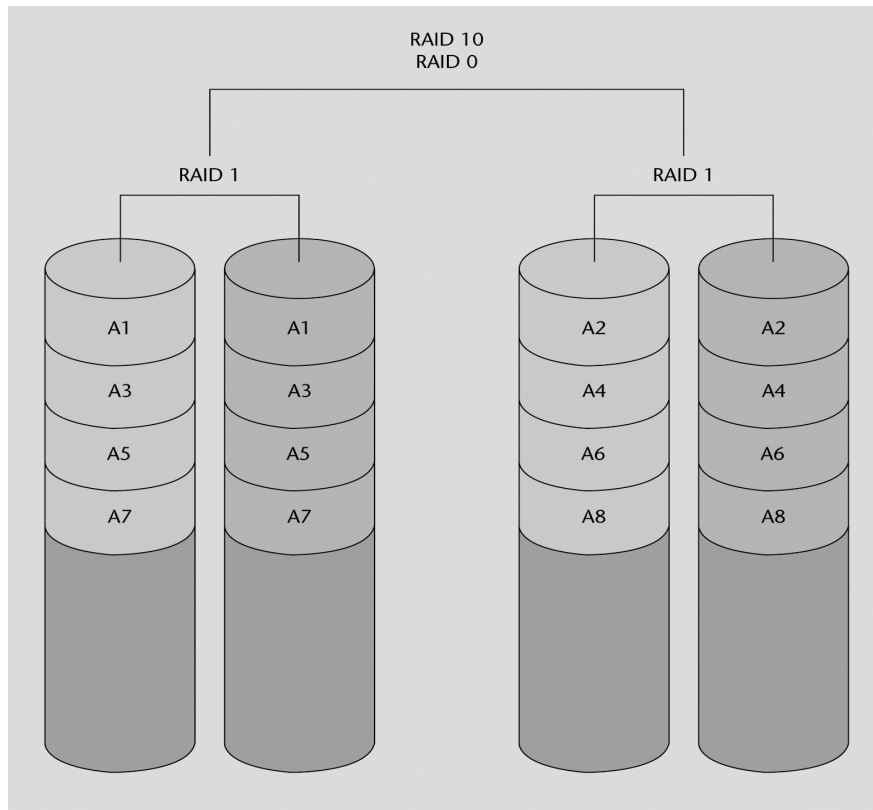


Figure 6

Some points that should be taken into account with regard to RAID in general:

- RAID improves the system's uptime, as some of the levels make it possible for the system to carry on working consistently when disks fail and, depending on the hardware, it is even possible to hot swap the problematic hardware without having to stop the system, which is especially important in critical systems.
- RAID can improve the performance of the applications, especially in systems with mirror implementations, where data striping permits the lineal read operations to increase significantly, as the disks can provide simultaneous read capability, increasing the data transfer rate.
- RAID does not protect data; evidently, it does not protect data from other possible malfunctions (virus, general errors or natural disasters). We must rely on backup copy schemes.
- Data recovery is not simplified. If a disk belongs to a RAID array, its recovery should be attempted within that environment. Software that is specific to the hardware drivers is necessary to access the data.

- On the other hand, it does not usually improve the performance of typical user applications, even if they are desktop applications, because these applications have components that access RAM and small sets of data, which means they will not benefit from lineal reading or sustained data transfers. In these environments, it is possible that the improvement in performance and efficiency is hardly even noticed.
- Information transfer is not improved or facilitated in any way; without RAID, it is quite easy to transfer data, by simply moving the disk from one system to another. In RAID's case, it is almost impossible (unless we have the same hardware) to move one array of disks to another system.

In GNU/Linux, RAID hardware is supported through various kernel modules, associated to different sets of manufacturers or chipsets of these RAID drivers. This permits the system to abstract itself from the hardware mechanisms and to make them transparent to the system and the end user. In any case, these kernel modules allow us to access the details of these drivers and to configure their parameters at a very low level, which in some cases (especially in servers that support a high I/O load) may be beneficial for tuning the disks system that the server uses in order to maximise the system's performance.

The other option that we will analyse is that of carrying out these processes through software components, specifically GNU/Linux's RAID software component.

GNU/Linux has a kernel of the so-called Multiple Device (md) kind, which we can consider as a support through the driver of the kernel for RAID. Through this driver we can generally implement RAID levels 0,1,4,5 and nested RAID levels (such as RAID 10) on different block devices such as IDE or SCSI disks. There is also the linear level, where there is a lineal combination of the available disks (it doesn't matter if they have different sizes), which means that disks are written on consecutively.

In order to use RAID software in Linux, we must have RAID support in the kernel, and, if applicable, the md modules activated (as well as some specific drivers, depending on the case (see available drivers associated to RAID, such as in Debian with modconf). The preferred method for implementing arrays of RAID disks through the RAID software offered by Linux is either during the installation or through the *mdadm* utility. This utility allows us to create and manage the arrays.

Let's look at some examples (we will assume we are working with some SCSI /dev/sda, /dev/sdb disks... in which we have various partitions available for implementing RAID):

Creation of a linear array:

```
# mdadm -create -verbose /dev/md0 -level=linear -raid-devices=2 /dev/sda1 /dev/sdb1
```

where we create a linear array based on the first partitions of `/dev/sda` and `/dev/sdb`, creating the new device `/dev/md0`, which can already be used as a new disk (supposing that the mount point `/media/diskRAID` exists):

```
# mkfs.ext2fs /dev/md0
# mount /dev/md0 /media/diskRAID
```

For a RAID 0 or RAID 1, we can simply change the level (`-level`) to `raid0` or `raid1`. With `mdadm -detail /dev/md0`, we can check the parameters of the newly created array.

We can also consult the `mdstat` entry in `/proc` to determine the active arrays and their parameters. Especially in the cases with mirrors (for example, in levels 1, 5...) we can examine the initial backup reconstruction in the created file; in `/proc/mdstat` we will see the reconstruction level (and the approximate completion time).

The `mdadm` utility provides many options that allow us to examine and manage the different RAID software arrays created (we can see a description and examples in `man mdadm`).

Another important consideration are the optimisations that should be made to the RAID arrays so as to improve the performance, through both the monitoring of its behaviour to optimise the file system parameters, as well as to use the RAID levels and their characteristics more effectively.

7.2. Logical Volume Manager (LVM)

There is a need to abstract from the physical disk system and its configuration and number of devices, so that the (operating) system can take care of this work and we do not have to worry about these parameters directly. In this sense, the logical volume management system can be seen as a layer of storage virtualisation that provides a simpler view, making it simpler and smoother to use.

In the Linux kernel, there is an LVM (*logical volume manager*), which is based on ideas developed from the storage volume managers used in HP-UX (HP's proprietary implementation of UNIX). There are currently two versions and LVM2 is the most widely used due to a series of added features.

The architecture of an LVM typically consists of the (main) components:

Note

The optimisation of the RAID arrays, may be an important resource for system tuning and some questions should be examined in:
Software-RAID-Howto, or in the actual `mdadm` man.

- **Physical volumes (PV):** PVs are hard disks or partitions or any other element that appears as a hard disk in the system (for example, RAID software or hardware).
- **Logical volumes (LV):** These are equivalent to a partition on the physical disk. The LV is visible in the system as a raw block device (completely equivalent to a physical partition) and it may contain a file system (such as the users' /home). Normally, the volumes make more sense for the administrators, as names can be used to identify them (for example, we can use a logical device, named stock or marketing instead of hda6 or sdc3).
- **Volume groups (VG):** This is the element on the upper layer. The administrative unit that includes our resources, whether they are logical volumes (LV) or physical volumes (PV). The data on the available PVs and how the LVs are formed using the PVs are saved in this unit. Evidently, in order to use a Volume Group, we have to have physical PV supports, which are organised in different logical LV units.

For example, in the following figure, we can see volume groups where we have 7 PVs (in the form of disk partitions, which are grouped to form two logical volumes (which have been completed using /usr and /home to form the file systems)):

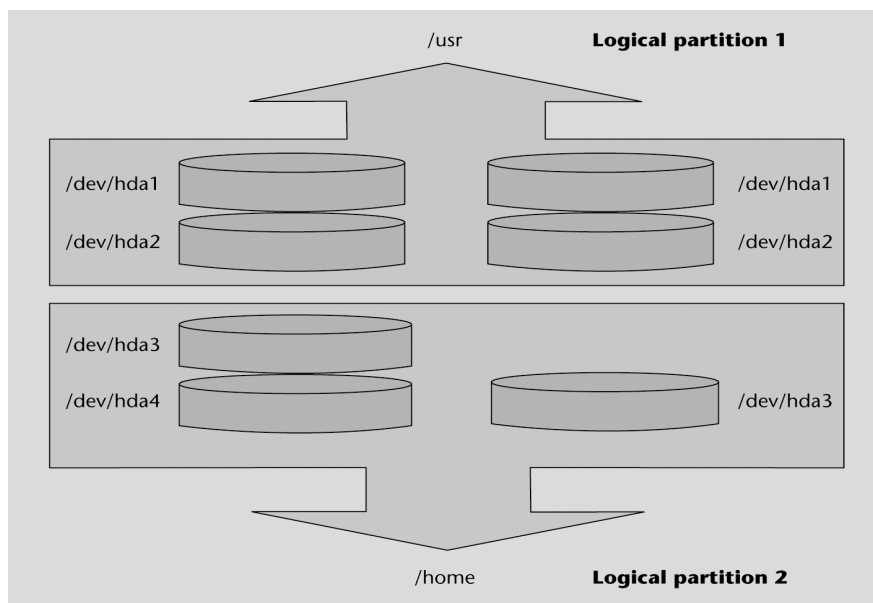


Figure 7. Scheme of an example of LVM

By using logical volumes, we can treat the storage space available (which may have a large number of different disks and partitions) more flexibly, according to the needs that arise, and we can manage the space by the more appropriate identifiers and by operations that permit us to adapt the space to the needs that arise at any given moment.

Logical Volume Management allows us to:

- Resize logical groups and volumes, using new PVs or extracting some of those initially available.
- Snapshots of the file system (reading in LVM1, and reading and/or writing in LVM2). This makes it possible to create a new device that is a snapshot of the situation of an LV. Likewise, we can create the snapshot, mount it, try various operations or configure new software or other elements and, if these do not work as we were expecting, we can return the original volume to the state it was in before performing the tests.
- RAID 0 of logical volumes.

RAID levels 1 or 5 are not implemented in LVM; if they are necessary (in other words, redundancy and fault tolerance are required), then either we use RAID software or RAID hardware drivers that will implement it and we place LVM as the upper layer.

We will provide a brief, typical example (in many cases, the distributor installer carries out a similar process if we set an LVM as the initial storage system). Basically, we must: 1) create physical volumes (PV). 2) create the logical group (VG) and 3) create the logical volume and finally use the following to create and mount a file system:

1) example: we have three partitions on different disks, we have created three PVs and started-up the contents:

```
# dd if=/dev/zero of=/dev/hda1 bs=1k count=1
# dd if=/dev/zero of=/dev/hda2 bs=1k count=1
# dd if=/dev/zero of=/dev/hdb1 bs=1k count=1
# pvcreate /dev/hda1
Physical volume "/dev/sda1" successfully created
# pvcreate /dev/hda2
Physical volume "/dev/hda2" successfully created
# pvcreate /dev/hdb1
Physical volume "/dev/hdb1" successfully created
```

2) placement of a VG created from the different PVs:

```
# vgcreate group_disks /dev/hda1 /dev/hda2 /dev/hdb1
```

Volume group "group_disks" successfully created

3) we create the LV (in this case, with a size of 1 GB) based on the elements that we have in group VG group (*-n* indicates the name of the volume):

```
# lvcreate -L1G -n logical_volume group_disks
lvcreate -- doing automatic backup of "group_disks"
lvcreate -- logical volume "/dev/group_disks/ logical_volume"
successfully created
```

And finally, we create a file system (a ReiserFS in this case):

```
# mkfs.reiserfs /dev/group_disks/logical_volume
```

Which we could, for example, place as backup space

```
# mkdir /mnt/backup
# mount -t reiserfs /dev/group_disks/logical_volume /mnt/
  backup
```

Finally, we will have a device as a logical volume that implements a file system in our machine.

8. Updating Software

In order to administer the installation or to update the software in our system, we will, in the first instance, depend on the type of software packages used by our system:

- RPM: packages that use the Fedora/Red Hat distribution (and derivatives). They are usually handled through the *rpm* command. Contains information on the dependencies that the software has on other software. At a high level, through *Yum* (or *up2date* in some distributions derived from Red Hat).
- DEB: Debian packages that are usually handled with a set of tools that work on different levels with individual packages or groups. Among these, we must mention: *dselect*, *tasksel*, *dpkg*, and *apt-get*.
- Tar or the tgz (also tar.gz): these are simply package files that have been joined and compressed using standard commands such as *tar*, and *gzip* (these are used for decompressing). The packages do not contain information on any dependencies and can normally be installed in different places if they do not carry any absolute root (path) information.

There are various graphical tools for handling these packages, such as RPM: Kpackage; DEB: Synaptic, Gnome-apt; Tgz: Kpackage, or from the actual graphic file manager itself (in Gnome or KDE). There are also usually package conversion utilities. For example, in Debian we have the *alien* command, with which we can change RPM packages to DEB packages. Although it is necessary to take the appropriate precautions, so that the package does not unexpectedly modify any behaviour or file system, as it has a different destination distribution.

Depending on the use of the types of packages or tools: it will be possible to update or install the software in our system in different ways:

- 1) From the actual system installation CDs; normally, all the distributions search for the software on the CDs. But the software should be checked to ensure that it is not old and does not, therefore, include some patches like updates or new versions with more features; consequently, if a CD is used for installation, it is standard practice to check that it is the latest version and that no more recent version exists.

- 2) Through updating or software search services, whether they are free, as is the case with Debian's apt-get tool or yum in Fedora, or through subscription services (paid services or services with basic facilities), such as the Red Hat Network of the commercial Red Hat versions.
- 3) Through software repositories that offer pre-built software packages for a determined distribution.
- 4) From the actual creator or distributor of the software, who may offer a series of software installation packages. We may find that we are unable to locate the type of packages that we need for our distribution.
- 5) Unpackaged software or with compression only, without any type of dependencies.
- 6) Only source code, in the form of a package or compressed file.

9. Batch jobs

In administration tasks, it is usually necessary to execute certain tasks at regular intervals, either because it is necessary to program the tasks so that they take place when the machine is least being used or due to the periodic nature of the tasks that have to be performed.

There are various systems that allow us to set up a task schedule (planning task execution) for performing these tasks out-of-hours, such as periodic or programmed services:

- *nohup* is perhaps the simplest command used by users, as it permits the execution of a non-interactive task once they have logged out from their account. Normally, when users log out, they lose their processes; *nohup* allows them to leave the processes executing even though the user has logged out.
- *at* permits us to launch a task for later, programming the determined point in time at which we wish for it to start, specifying the time (hh:mm) and date, or specifying whether it will be today or tomorrow. Examples:
at 10pm task
to perform the task at ten o'clock at night.
at 2am tomorrow task
to perform the task at two o'clock in the morning.
- *cron*: it permits us to establish a list of tasks that will be performed with the corresponding programming; this configuration is saved in */etc/crontab*; specifically, in each entry in this file, we have: hour and minutes at which the task will be performed, which day of the month, which month, which day of the week, along with which element (which might be a task or a directory where the tasks that are to be executed are located). For example, the standard content is similar to:

```
25 6 * * * root test -e /usr/sbin/anacron || run-parts --report /etc/cron.daily
47 6 * * 7 root test -e /usr/sbin/anacron || run-parts --report /etc/cron.weekly
52 6 1 * * root test -e /usr/sbin/anacron || run-parts --report /etc/cron.monthl
```

where a series of tasks are programmed to execute: each day ("*" indicates 'whichever'), weekly (7th day of the week) or monthly (the 1st day of each month). Normally, the tasks will be executed with the *crontab* command, but the cron system assumes that the machine is always switched on, and if this is not the case, it is better to use *anacron*, which checks whether the task was performed when it was supposed to be or not, and if not, it executes the task.

Each line in the preceding file is checked to ensure that the *anacron* command is there and the scripts associated to each action are executed; in this case, they are saved in directories assigned for this.

There may also be *cron.allow* or *cron.deny* files to limit who can (or cannot) put tasks in *cron*. Through the *crontab* command, a user may define tasks in the same format as we have seen before, which are usually saved in */var/spool/cron/crontabs*. In some cases, there is also a */etc/cron.d* directory where we can place the tasks and they are treated as through they were an extension to the */etc/crontab* file.

10. Tutorial: combined practices of the different sections

We will begin by examining the general state of our system. We will carry out different steps in a Debian system. It is an unstable Debian system (the unstable version, but more updated); however, the procedures are, mostly, transferable to other distributions such as Fedora/Red Hat (we will mention some of the most important changes). The hardware consists of a Pentium 4 at 2.66 Ghz with 768 MB RAM and various disks, DVD and CD-writer, as well as other peripherals, on which we will obtain information as we proceed step by step.

First we will see how our system booted up the last time:

```
# uptime
17:38:22 up 2:46, 5 users, load average: 0.05, 0.03, 0.04
```

This command tells us the time that the system has been up since it last booted, 2 hours and 47 minutes and, in this case, we have 5 users. These will not necessarily correspond to five different users, but they will usually be opened user sessions (for example, through one terminal). The *who* command provides a list of these users. The load average is the system's average load over the last 1, 5 and 15 minutes.

Let's look at system's boot log (*dmesg* command), and the lines that were generated when the system booted up (we have removed some lines for the purpose of clarity):

```
Linux version 2.6.20-1-686 (Debian 2.6.20-2) (waldi@debian.org)
(gcc version 4.1.2 20061115 (prerelease) (Debian 4.1.1-21)) #1 SMP Sun Apr
15 21:03:57 UTC 2007
BIOS-provided physical RAM map:
  BIOS-e820: 0000000000000000 - 000000000009f800 (usable)
  BIOS-e820: 000000000009f800 - 00000000000a0000 (reserved)
  BIOS-e820: 00000000000ce000 - 00000000000d0000 (reserved)
  BIOS-e820: 00000000000dc000 - 0000000000100000 (reserved)
  BIOS-e820: 0000000000100000 - 0000000002f6e000 (usable)
  BIOS-e820: 0000000002f6e000 - 0000000002f6f000 (ACPI data)
  BIOS-e820: 0000000002f6f000 - 0000000002f70000 (ACPI NVS)
  BIOS-e820: 0000000002f70000 - 0000000002f78000 (usable)
  BIOS-e820: 0000000002f78000 - 0000000030000000 (reserved)
  BIOS-e820: 00000000fff80000 - 00000000ffc00000 (reserved)
  BIOS-e820: 00000000fffffc00 - 0000000100000000 (reserved)
OMB HIGHMEM available.
```



```
759MB LOWMEM available.
```

These first lines already indicate some interesting data: the Linux kernel is version 2.6.20-1-686, one version 2.6 revision 20 at revision 1 of Debian and for 686 machines (Intel x86 32 bits architecture). They also indicate that we are booting a Debian system, with this kernel which was compiled with a GNU gcc compiler, version 4.1.2 and the date. There is then a map of the memory zones used (reserved) by the BIOS and then the total memory detected in the machine: 759 MB, to which we would have to add the first 1 MB, making a total of 760 MB.

```
Kernel command line: BOOT_IMAGE=LinuxNEW ro root=302 lang=es acpi=force
Initializing CPU#0
Console: colour dummy device 80x25
Memory: 766132k/777728k available (1641k kernel code, 10968k reserved, 619k da-
ta, 208k init, 0k highmem)
Calibrating delay using timer specific routine.. 5320.63 BogoMIPS (lpj=10641275)
```

Here, we are told how the machine booted up and which command line has been passed to the kernel (different options may be passed, such as lilo or grub). And we are booting in console mode with 80 x 25 characters (this can be changed). The BogoMIPS are internal measurements of the kernel of the CPU speed. There are architectures in which it is difficult to detect how many MHz the CPU works with and this is why this speed measurement is used. Subsequently, we are given more data on the main memory and what it is being used for at this booting stage.

```
CPU: Trace cache: 12K uops, L1 D cache: 8K
CPU: L2 cache: 512K
CPU: Hyper-Threading is disabled
Intel machine check architecture supported.
Intel machine check reporting enabled on CPU#0.
CPU0: Intel P4/Xeon Extended MCE MSRs (12) available
CPU0: Intel(R) Pentium(R) 4 CPU 2.66GHz stepping 09
```

Likewise, we are given various data on the CPU: the size of the first-level cache, the internal CPU cache, L1 divided in a TraceCache of the Pentium 4 (or cache instruction), and the data cache and the unified second-level cache (L2), the type of CPU, its speed and the system's bus.

```

PCI: PCI BIOS revision 2.10 entry at 0xfd994, last bus=3
Setting up standard PCI resources
...
NET: Registered protocol
IP route cache hash table entries: 32768 (order: 5, 131072 bytes)
TCP: Hash tables configured (established 131072 bind 65536)
checking if image is initramfs... it is
Freeing initrd memory: 1270k freed
fb0: VESA VGA frame buffer device
Serial: 8250/16550 driver $Revision: 1.90 $ 4 ports, IRQ sharing enabled
serial8250: ttyS0 at I/O 0x3f8 (irq = 4) is a 16550A
00:09: ttyS0 at I/O 0x3f8 (irq = 4) is a 16550A
RAMDISK driver initialized: 16 RAM disks of 8192K size 1024 blocksize
PNP: PS/2 Controller [PNP0303:KBC0,PNP0f13:MSE0] at 0x60,0x64 irq 1,12
i8042.c: Detected active multiplexing controller, rev 1.1.
serial: i8042 KBD port at 0x60,0x64 irq 1
serial: i8042 AUX0 port at 0x60,0x64 irq 12
serial: i8042 AUX1 port at 0x60,0x64 irq 12
serial: i8042 AUX2 port at 0x60,0x64 irq 12
serial: i8042 AUX3 port at 0x60,0x64 irq 12
mice: PS/2 mouse device common for all mice

```

The kernel and devices continue to boot, mentioning the initiation of the network protocols. The terminals, the serial ports ttyS0 (which would be com1) and ttyS01 (com2). It provides information on the RAM disks that are being used, the detection of PS2 devices, keyboard and mouse.

```

ICH4: IDE controller at PCI slot 0000:00:1f.1

ide0: BM-DMA at 0x1860-0x1867, BIOS settings: hda:DMA, hdb:pio
ide1: BM-DMA at 0x1868-0x186f, BIOS settings: hdc:DMA, hdd:pio
Probing IDE interface ide0...
hda: FUJITSU MHT2030AT, ATA DISK drive
ide0 at 0x1f0-0x1f7,0x3f6 on irq 14
Probing IDE interface ide1...
hdc: SAMSUNG CDRW/DVD SN-324F, ATAPI CD/DVD-ROM drive
ide1 at 0x170-0x177,0x376 on irq 15
SCSI subsystem initialized
libata version 2.00 loaded.
hda: max request size: 128KiB
hda: 58605120 sectors (30005 MB) w/2048KiB Cache, CHS=58140/16/63<6>hda:
hw_config=600b
, UDMA(100)
hda: cache flushes supported
hda: hda1 hda2 hda3
kjournald starting. Commit interval 5 seconds
EXT3-fs: mounted file system with ordered data mode.
hdc: ATAPI 24X DVD-ROM CD-R/RW drive, 2048kB Cache, UDMA(33)
Uniform CD-ROM driver Revision: 3.20
Addinf 618492 swap on /dev/hda3.

```

Detection of IDE devices, detecting the IDE chip in the PCI bus and reporting what is driving the devices: hda, and hdc, which are, respectively: a hard disk (Fujitsu), a second hard disk, a Samsung DVD Samsung, and a CD-writer (given that in this case, we have a combo unit). It indicates active partitions. Subsequently, the machine detects the main Linux file system, a journaled ext3, that activates and adds the swap space available in a partition.

```
usbcore: registered new interface driver usbfs
usbcore: registered new interface driver hub
usbcore: registered new device driver usb
input: PC Speaker as /class/input/input1
USB Universal Host Controller Interface driver v3.0
hub 1-0:1.0: USB hub found
hub 1-0:1.0: 2 ports detected
uhci_hcd 0000:00:1d.1: UHCI Host Controller
uhci_hcd 0000:00:1d.1: new USB bus registered, assigned bus number 2
uhci_hcd 0000:00:1d.1: irq 11, io base 0x00001820
usb usb2: configuration #1 chosen from 1 choice
hub 2-0:1.0: USB hub found
hub 2-0:1.0: 2 ports detected
hub 4-0:1.0: USB hub found
hub 4-0:1.0: 6 ports detected
```

More detection of devices, USB (and the corresponding modules); in this case, two hub devices (with a total of 8 USB ports) have been detected.

```
parport: PnPBIOS parport detected.
parport0: PC-style at 0x378 (0x778), irq 7, dma 1
[PCSP,TRISTATE,COMPAT,EPP,ECP,DMA]
input: IMPS/2 Logitech Wheel Mouse as /class/input/input2
ieee1394: Initialized config rom entry 'ip1394'
eepro100.c:v1.09j-t 9/29/99 Donald Becker
Synaptics Touchpad, model: 1, fw: 5.9, id: 0x2e6eb1, caps: 0x944713/0xc0000
input: SynPS/2 Synaptics TouchPad as /class/input/input3

agpgart: Detected an Intel 845G Chipset
agpgart: Detected 8060K stolen Memory
agpgart: AGP aperture is 128M
eth0: OEM i82557/i82558 10/100 Ethernet, 00:00:F0:84:D3:A9, IRQ 11.
Board assembly 000000-000, Physical connectors present: RJ45
e100: Intel(R) PRO/100 Network Driver, 3.5.17-k2-NAPI
usbcore: registered new interface driver usbkbd
Initializing USB Mass Storage driver...
usbcore: registered new interface driver usb-storage
USB Mass Storage support registered.

lp0: using parport0 (interrupt-driven).
ppdev: user-space parallel port driver
```

And the final detection of the rest of the devices: Parallel port, mouse model, FireWire port (IEEE1394) network card (Intel), a touchscreen, the AGP video card (i845). More data on the network card, an intel pro 100, registry of usb as mass storage (indicates a USB storage device as an external disk) and detection of parallel port.

We can also see all this information, which we accessed through the `dmesg` command, dumped in the system's main log, `/var/log/messages`. In this log, we will find the kernel messages, among others, the messages of the daemons and network or device errors, which communicate their messages to a special daemon called `syslogd`, which is in charge of writing the messages in this file. If we have recently booted the machine, we will observe that the last lines contain exactly the same information as the `dmesg` command,

for example, if we look at the final part of the file (which is usually very large):

```
# tail 200 /var/log/messages
```

We observe the same lines as before and some more information such as:

```
shutdown[13325]: shutting down for system reboot
kernel: usb 4-1: USB disconnect, address 3
kernel: nfsd: last server has exited
kernel: nfsd: unexporting all file systems
kernel: Kernel logging (proc) stopped.
kernel: Kernel log daemon terminating.

exiting on signal 15
syslogd 1.4.1#20: restart.

kernel: klogd 1.4.1#20, log source = /proc/kmsg started.
Linux version 2.6.20-1-686 (Debian 2.6.20-2) (waldi@debian.org) (gcc version 4.1.2
20061115 (prerelease) (Debian 4.1.1-21)) #1 SMP Sun Apr 15 21:03:57 UTC 2007
kernel: BIOS-provided physical RAM map:
```

The first part corresponds to the preceding shutdown of the system, informing us that the kernel has stopped placing information in `/proc`, that the system is shutting down... At the beginning of the new boot, the Syslogd daemon that generates the log is activated, and the system begins to load, which tells us that the kernel will begin to write information in its system, `/proc`; we look at the first lines of the `dmesg` mentioning the version of the kernel that is being loaded and we then find what we have seen with `dmesg`.

At this point, another useful command for finding out how the load process has taken place is *lsmod*, which will tell us which modules have been loaded in the kernel (summarised version):

```
# lsmod

Module Size Used by
nfs 219468      0
nfsd 202192     17
exportfs 5632      1 nfsd
lockd 58216      3 nfs,nfsd
nfs_acl 3616      2 nfs,nfsd
sunrpc 148380     13 nfs,nfsd,lockd,nfs_acl
ppdev 8740      0
lp 11044      0
button 7856      0
ac 5220      0
battery 9924      0
md_mod 71860      1
dm_snapshot 16580      0
dm_mirror 20340      0
dm_mod 52812     2 dm_snapshot,dm_mirror
```

i810fb 30268	0
vgastate 8512	1 i810fb
eeeprom 7184	0
thermal 13928	0
processor 30536	1 thermal
fan 4772	0
udf 75876	0
ntfs 205364	0
usb_storage 75552	0
hid 22784	0
usbkbd 6752	0
eth1394 18468	0
e100 32648	0
eeepro100 30096	0
ohci1394 32656	0
ieee1394 89208	2 eth1394,ohci1394
snd_intel8x0 31420	1
snd_ac97_codec 89412	1 snd_intel8x0
ac97_bus 2432	1 snd_ac97_codec
parport_pc 32772	1
snd 48196	6 snd_intel8x0,snd_ac97_codec,snd_pcm,snd_timer
ehci_hcd 29132	0
ide_cd 36672	0
cdrom 32960	1 ide_cd
soundcore 7616	1 snd
psmouse 35208	0
uhci_hcd 22160	0
parport 33672	3 ppdev,lp,parport_pc
intelfb 34596	0
serio_raw 6724	0
pcspkr 3264	0
pci_hotplug 29312	1 shpchp
usbcore 122312	6 dvb_usb,usb_storage,usbkbd,ehci_hcd,uhci_hcd
intel_agp 22748	1
agpgart 30504	5 i810fb,drm,intelfb,intel_agp
ext3 121032	1
jbd 55368	1 ext3
ide_disk 15744	3
ata_generic 7876	0
ata_piix 15044	0
libata 100052	2 ata_generic,ata_piix
scsi_mod 133100	2 usb_storage,libata
generic 4932	0 [permanent]
piix 9540	0 [permanent]
ide_core 114728	5 usb_storage,ide_cd,ide_disk,generic,piix

We see that we basically have the drivers for the hardware that we have detected and other related elements or those necessary by dependencies.

This gives us, then, an idea of how the kernel and its modules have been loaded. In this process, we may already have observed an error, if the hardware is not properly configured or there are kernel modules that are not properly compiled (they were not compiled for the appropriate kernel version), inexistent etc.

The next step for examining the processes in the system, such as the *ps* (for process status) command, for example (only the system processes are shown, not the user ones):

```
# ps -ef
UID PID PPID C STIME TTY TIME CMD
```

Processes information, UID user that has launched the process (or the identifier with which it has been launched), PID and process code assigned by the system are consecutively shown, as the processes launch; the first is always 0, which corresponds to the init process. PPID is the id of the current parent process. STIME, time in which the process was booted, TTY, terminal assigned to the process (if there is one), CMD, command line with which it was launched.

```
root 1 0 0 14:52 ? 00:00:00 init [2]
root 3 1 0 14:52 ? 00:00:00 [ksoftirqd/0]
root 143 6 0 14:52 ? 00:00:00 [bdf flush]
root 145 6 0 14:52 ? 00:00:00 [kswapd0]
root 357 6 0 14:52 ? 00:00:01 [kjournald]
root 477 1 0 14:52 ? 00:00:00 udevd --daemon
root 719 6 0 14:52 ? 00:00:00 [khubd]
```

Various system daemons, such as the kswapd daemon, which controls the virtual memory swaps. Handling of system buffers (bdf flush). Handling of file system journal (kjournald), USB handling (khubd). Or the udev daemon that controls the hot device connection. In general, the daemons are not always identified by a d at the end, and if they have a k at the beginning, they are normally internal threads of the kernel.

```
root 1567 1 0 14:52 ? 00:00:00 dhclient -e -pf ...
root 1653 1 0 14:52 ? 00:00:00 /sbin/portmap
root 1829 1 0 14:52 ? 00:00:00 /sbin/syslogd
root 1839 1 0 14:52 ? 00:00:00 /sbin/klogd -x
root 1983 1 0 14:52 ? 00:00:09 /usr/sbin/cupsd
root 2178 1 0 14:53 ? 00:00:00 /usr/sbin/inetd
```

We have `dhclient`, which indicates that the machine is the client of a DHCP server, for obtaining its IP. `Syslogd`, a daemon that sends messages to the log. The cups daemon, which, as we have discussed, is related to the printing system. And `inetd`, which, as we shall see in the section on networks, is a type of "superserver" or intermediary of other daemons related to network services.

```
root    2154 1 0 14:53 ?      00:00:00 /usr/sbin/tpc.mountd
root    2241 1 0 14:53 ?      00:00:00 /usr/sbin/sshd
root    2257 1 0 14:53 ?      00:00:00 /usr/bin/xfs -daemon
root    2573 1 0 14:53 ?      00:00:00 /usr/sbin/atd
root    2580 1 0 14:53 ?      00:00:00 /usr/sbin/cron
root    2675 1 0 14:53 ?      00:00:00 /usr/sbin/apache
www-data 2684 2675 0 14:53 ?    00:00:00 /usr/sbin/apache
www-data 2685 2675 0 14:53 ?    00:00:00 /usr/sbin/apache
```

There is also `sshd`, a safe remote access server (an improved version that permits services compatible with telnet and FTP). `xfs` is the fonts server (character types) of X Window. The `atd` and `cron` commands can be used for handling programmed tasks at a determined moment. Apache is a web server, which may have various active threads for attending to different requests.

```
root 2499 2493 0 14:53 ?    00:00:00 /usr/sbin/gdm
root 2502 2499 4 14:53 tty7  00:09:18 /usr/bin/X :0 -dpi 96 ...
root 2848 1 0 14:53 tty2    00:00:00 /sbin/getty 38400 tty2
root 2849 1 0 14:53 tty3    00:00:00 /sbin/getty 38400 tty3
root 3941 2847 0 14:57 tty1  00:00:00 -bash
root 16453 12970 0 18:10 pts/2 00:00:00 ps -ef
```

`gdm` is the graphical login of the Gnome desktop system (the entry point where we are asked for the login name and password) and the `getty` processes are the ones that manage the virtual text terminals (which we can see by pressing `Alt+Fx` (or `Ctrl+Alt+Fx` if we are in graphic mode)). `X` is the process of the X Window System graphic server and is essential for executing any desktop environment above this. An open shell (`bash`), and finally, the process that we have generated when requesting this `ps` from the command line.

The `ps` command provides various command line options for adjusting the information that we want on each process, whether it is the time that it has been executing, the percentage of CPU used, memory used etc. (see man of `ps`). Another very interesting command is `top`, which does the same as `ps` but dynamically; in other words, it updates every certain period of time, we can classify the processes by use of CPU or memory and it also provides information on the state of the overall memory.

Other useful commands for resources management are `free` and `vmstat`, which provide information on the memory used and the virtual memory system:

```
# free      total used free shared buffers cached
Mem: 767736 745232 22504 0 89564 457612
-/+ buffers/cache: 198056 569680
```

Note

See man of the commands to interpret outputs.

Swap: 618492 1732 616760

```
# vmstat
procs -----memory----- ---swap-- -----io-- --system-- ----cpu----
r b swpd free buff cache si so bi bo in cs us sy id wa
1 0 1732 22444 89584 457640 0 0 68 137 291 418 7 1 85 7
```

The free command also shows the swap size, approximately 600 MB, which are not currently used intensely as there is sufficient physical memory space; there are still 22 MB free (which indicates a high use of the physical memory and the need to use swap soon). The memory space and swap (as of kernels 2.4) add to each other to comprise the total memory in the system, which in this case, means that there is a total of 1.4 GB available. This may seem a lot, but it will depend on the applications that are being executed.

Activities

- 1) The swap space makes it possible to add to the physical memory so that there is more virtual memory. Depending on the amounts to add extra space to the physical memory and swap space, can all the memory get used up? Can we resolve this in any other way that does not involve adding more physical memory?
- 2) Suppose that we have a system with two Linux partitions: one / and one swap partition. How do we solve the situation if the user accounts use up all the disk space? And if we have an isolated /home partition, which was also being used up, how would we solve this?
- 3) Install the CUPS printing system, define our printer so that it works with CUPS and try administering through the web interface. As the system is now, would it be advisable to modify, in any way, CUPS' default settings? Why?
- 4) Examine the default setting that comes with the GNU/Linux system for non-interactive work using cron. Which jobs are there and when are they being performed? Any ideas for new jobs that have to be added?
- 5) Reproduce the workshop analysis (plus the other sections of the unit) on the machine that is available. Can we see any errors or irregular situations in the examined system? If so, how do we solve them?

Bibliography

Other sources of reference and information

[Wm02] [Fri02] [Smi02] GNU/Linux and UNIX administration manuals, which explain in detail the aspects on local administration and printing systems management.

[Gt] Updated information on the printing systems and their settings, as well as the details of some of the printers, can be found here. For specific details on the printer models and drivers, we can go to <http://www.linuxprinting.org/>.

[Hin][Koe] We can find information on the different file systems available and the schemes for creating partitions for the installation of the system.

Network administration

Remo Suppi Boldrito

PID_00148471



Universitat Oberta
de Catalunya

www.uoc.edu

Index

Introduction.....	5
1. Introduction to TCP/IP (TCP/IP suite).....	7
1.1. Services on TCP/IP	7
1.2. What is TCP/IP?	9
1.3. Physical network devices (hardware)	10
2. TCP/IP Concepts.....	13
3. How to assign an Internet address.....	16
4. How to configure the network.....	20
4.1. Configuration of the network interface controller (NIC)	20
4.1.1. Configuration of network in Fedora style	22
4.1.2. Configuration of a Wi-Fi (wireless) network	23
4.2. Configuration of Name Resolver	25
4.3. Configuration of routing	27
4.4. Configuration of inetd	28
4.5. Additional configuration: protocols and networks	31
4.6. Security aspects	31
4.7. IP Options	33
4.7.1. Commands for solving problems with the network	33
5. DHCP Configuration.....	35
6. IP aliasing.....	37
7. IP Masquerade.....	38
8. NAT with kernel 2.2 or higher.....	39
9. How to configure a DialUP and PPP connection.....	40
10. Configuring the network through <i>hotplug</i>.....	43
11. Virtual private network (VPN).....	45
11.1. Simple example	45
12. Advanced configurations and tools.....	48
Activities.....	55

Annex. Controlling the services linked to an FC6 network..... 56

Introduction

The UNIX (GNU/Linux) operating system is used as an example of a standard communications architecture. From the mythical UUCP (Unix-to-Unix CoPy or service for copying between UNIX operating systems) to the current networks, UNIX has always proven its versatility in aspects related to communication and information exchange. With the introduction of computer networks (Local Area Networks, Wide Area Networks or the latest Metropolitan Area Networks) offering multipoint connections at different speeds (from 56 kbits/sec to 1 Gbit/sec), new services that are based on faster protocols, portable between different computers and better adapted, such as TCP/IP (*transport control program / Internet protocol*), have arisen. [Com01, Mal96, Cis00, Gar98, KD00]

1. Introduction to TCP/IP (TCP/IP suite)

The TCP/IP protocol synthesises an example of a will to communicate and to standardise the communication on a global scale.

The TCP/IP is, in reality, a set of basic protocols that have been added to the original protocol, to meet the different needs in computer-to-computer communication, such as TCP, UDP, IP, ICMP, ARP. [Mal96]

TCP/IP is most frequently used by most current users to remotely connect to other computers (telnet, SSH Secure Shell), to use remote files (NFS *network file system*) or to transfer them (FTP *file transfer protocol*, HTTP *hypertext markup protocol*).

Note

Typical use of TCP/IP remote login:
telnet localhost Debian
GNU/Linux 4.0
login:

1.1. Services on TCP/IP

The most important traditional TCP/IP services are [Gar98]:

- **File transfer:** the *file transfer protocol* (FTP) allows the user of a computer to obtain files or send them from one computer to another. In order to do this, the user must have an account in the remote computer and identify themselves through their login name and password or the user must connect to computers containing an information repository (software, documentation etc.) under an anonymous account to read those computers on their computer. This is not the same as the more recent Network File Systems (NFS) (or netbios protocols over TCP/IP, a completely insecure "invention" in Windows, which should be replaced with an older but more secure version called netbeui) that make it possible to virtualise the file system in a machine so that it can be accessed interactively from another computer.
- **Remote connection (login):** the terminal network protocol (telnet) allows a user to remotely connect to a computer. The local computer is used as the remote computer's terminal and everything is executed over it, whilst the local computer remains invisible from the perspective of the user that started the session. This service has now been replaced by the SSH (*secure shell*), for security reasons. This can use a remote connection through telnet and the messages are sent as plain text; in other words, if someone "examines" the messages on the network, it is equivalent to looking at the user's screen. SSH encrypts the information (which is an added-value to

the communication) so that the packages on the network cannot be read by any foreign node.

- **Email:** this service makes it possible to send messages to users of other computers. This form of communication has become an essential element for users and allows email messages to be sent to a central server, so that they can then be recovered using specific programs (clients) or read through an internet connection.

The progress in the technology and the increasingly lower cost of computers has meant that determined services have specialised and are now configured on determined computers working in a client-server model. A server is a system that performs specific services for the rest of the network or connected clients. A client is another computer that uses this service. All of these services are generally offered within TCP/IP:

- **File systems in network file systems:** allows a system to access the files through a remote system in a manner that is more integrated than FTP. The storage devices (or part of them) are exported to the system that wishes to access the files and this system can "see" them as if they were local devices. This protocol permits in the server side to establish the rules and ways of accessing the files, which (if properly configured) makes the place where the information physically resides independent from the place where the information is "accessed".
- **Remote printing:** permits users to access printers connected to other computers.
- **Remote execution:** permits a user to execute a program on another computer. There are various ways of executing a program in this way: either through a command (*rsh*, *ssh*, *rexec*) or through systems with RPC (*remote procedure call*), which allows a program on a local computer to execute a function in a program on another computer. The RPC processes have been studied in-depth and there are various implementations, but the most common are Xerox's *Courier* and Sun's *RPC* (the latter has been adopted in most UNIX systems).
- **Name servers:** in large-scale networks of computers, there are data that have to be centralised so that they can be easily used; for example, user names, passwords, internet addresses etc. All of this makes it easier for a user to have an account for all the machines in an organisation. For example, Sun's *Yellow Pages* (*NIS* in the current *Sun* versions) is designed to handle all these types of data and it is available for most UNIX systems. The *DNS* (*domain name system*) is another domain-name service but one that keeps a direct relationship between the hostname and the logical identification name of this machine (IP address).

- **Terminal Servers:** connect terminals to a server that executes telnet so as to connect to the central computer. These types of setup are basically useful for reducing costs and improving the connections to the central computer (in some cases).
- **Graphical terminal servers** (*network-oriented window systems*): these permit a computer to visualise graphic information on a display that is connected to another computer. The most common of these systems is X Window.

1.2. What is TCP/IP?

TCP/IP is in fact two communication protocols between computers that are independent to each other.

On the one hand, TCP (*transmission control protocol*) defines the communication rules so that a (host) computer can talk to another computer (if we use the OSI/ISO communications model as a reference, it describes layer 4, see following table).

TCP is a connection-oriented protocol, in other words, it is equivalent to a telephone, and the communication is considered as a data stream.

IP (*Internet protocol*) defines the protocol to identify the networks and establish the pathways between different computers.

In other words, it routes the data between two computers through the networks. It corresponds to layer 3 of the OSI/ISO model and it is a connectionless protocol (see following table). [Com01, Rid00, Dra99]

An alternative to TCP is the UDP protocol (*user datagram protocol*), which treats the data as a message (datagram) and sends packets. It is a connectionless protocol (the recipient computer does not necessarily have to be listening when the other computer establishes communication with it) and it has the advantage of creating less overload on the network than a TCP connection, but it is less reliable (the packets may not arrive or arrive duplicated).

There is another alternative protocol called ICMP (*Internet control message protocol*). ICMP is used for error or control messages. For example, if one tries to connect to a host computer, the local computer may receive an ICMP message indicating "host unreachable". ICMP may also be used to extract information on a network. ICMP is similar to UDP in that it handles messages (datagrams),

but it is simpler than UDP, because it does not have port identification (the ports are mailboxes where the data packets are left and where the server applications read the packets) in the message header.

In the OSI/ISO communications model (OSI, *open systems interconnection reference model*, ISO, *International Standards Organization*), is a theoretical model applied by many networks. There are seven communication layers where each one has an interface for communicating with the preceding and following one.

Level	Name	Use
7	Application	SMTP, <i>simple mail transfer protocol</i> , the service itself
6	Introduction	Telnet, FTP implements the service protocol
5	Session	Generally not used
4	Transport	TCP, UDP transformation in accordance with the communication protocol.
3	Network	IP makes it possible to route the packet.
2	Link	Drivers - transformation in accordance with the physical protocol.
1	Physical	Ethernet, ADSL... physically sends the packet

To summarise, TCP/IP is a set of protocols including IP, TCP, UDP that provide a set of low-level functions used by most of the applications. [KD00, Dra99].

Some of the protocols that use the abovementioned services were designed by Berkeley, Sun or other organisations. They are not included (officially) as part of the *Internet protocol suite* (IPS). However, they are implemented using TCP/IP and they are therefore considered as a formal part of IPS. A description of the protocols available by Internet can be found in RFC 1011 (see references on RFC [IET]). There is currently a new version of protocol IPv6, also called IPng (IP *next generation*) which replaces IPv4. This protocol significantly improves the previous ones in elements such as having a greater number of nodes, traffic control, security or improvements in the routing.

1.3. Physical network devices (hardware)

From the physical point of view (layer 1 of the OSI model), the most commonly used hardware for LAN is that known as Ethernet (or FastEthernet or GigaEthernet). Its advantages consist of a lower cost, acceptable speeds (10, 100 or 1,000 megabits per second) and its user-friendly installation.

There are three connection modes, depending on the type of interconnection: thick, thin and twisted pair.

The first two are obsolete (they used coaxial cable) whereas the last is through twisted pair cables and connectors similar to those used by telephones (known as RJ45). The twisted pair connection is known as 10baseT or 100baseT (according to the speed) and it uses repeaters known as hubs as interconnection points. Ethernet technology uses intermediate communication elements (hubs, switches, routers) to configure multiple segments of the network and divide the traffic to improve the performance of the data transfer. Normally, in large organisations, these Ethernet LAN are interconnected through fibre optic cables using FDDI (*fibre distributed data interface*) technology, which is more expensive and more difficult to install, but with which we can obtain transmission speeds equivalent to Ethernet whilst not having the limits on distance involved in Ethernet (FDDI allows for distances of up to 200 km). The costs are justified when they are used between buildings or other network segments that are very congested. [Rid00, KD00].

At the same time, there are other types of hardware that are less common, but no less interesting, such as ATM (*asynchronous transfer mode*). This hardware allows us to set up a LAN with a high level of service quality and it is a good option when we have to set up high-speed and low-latency networks, such as those that require real time video streaming.

There is other hardware supported by GNU/Linux for interconnecting computers, of which we would mention: Frame Relay or X.25 (used in computers that access or interconnect WANs and for servers with large data transfer needs), Packet Radio (interconnection via radio using protocols such as AX.25, NetRom or Rose) or dial-up devices that use serial lines, which are slow but very cheap, through analogical or digital (RDSI, DSL, ADSL etc.) modems. The latter are the ones commonly used domestically or in small and medium-sized businesses, and they require another protocol for the transmission of packets, such as SLIP or PPP. In order to virtualise the diverse hardware on a network, TCP/IP defines an abstract interface through which all the packets that will be sent by a physical device (which includes a network or network segment) are concentrated. Consequently, for each communication device in the machine, we will have a corresponding interface in the operating system's kernel.

Example

In GNU/Linux, Ethernet is called with ethx (where, "x" indicates an order number beginning with 0), the interface to serial lines (modems) is called up with pppx (for PPP) or slx (for SLIP); fddix is used for FDDI. These names are used by the commands to configure them and assign them the identification that will subsequently permit them to communicate with other devices in the network.

In GNU/Linux, this may mean that we have to include the appropriate modules for the appropriate device (NIC *network interface card*) in the kernel or as modules, and this means compiling the kernel after choosing, the appropriate NIC, with, for example, *make menuconfig*, indicating it as internal or as a module (in the latter case, the appropriate module must also be compiled).

The network devices can be seen in the `/dev` directory, where there is a file (a special file, which may be a block file or a character file, according to the transfer) that represents each hardware device.[KD00, Dra99].

Note

How do we see the network interfaces that are available?

```
ifconfig -a
```

This command shows all of the default interfaces/parameters for each one.

2. TCP/IP Concepts

As we have observed, communication involves a series of concepts that we will now discuss [Mal96, Com01]:

- **Internet/intranet:** the term *intranet* refers to the application of Internet technology (the network of networks) within an organisation, basically to distribute the company's internal information and to have it available within the company. For example, the services offered by GNU/Linux as Internet and Intranet services include email, WWW, news etc.
- **Node:** the (host) node refers to a machine that is connected to the network (in a wider sense, a node may be a computer, a printer, a CD (rack) etc.); in other words, an active and differentiable element in the network that requires or provides some kind of service and/or shares information.
- **Ethernet Network Address** (*Ethernet address* or *MAC address*): a 48-bit number (for example 00:88:40:73:AB:FF –in octal– 0000 0000 1000 1000 0100 0000 0111 0011 1010 1011 1111 1111 –in binary–) that is inside the physical device (hardware) of the Ethernet driver (NIC) and that is recorded by the manufacturer (this number must be the only one in the world, each NIC manufacturer has a pre-allocated range).
- **Host name:** each node must also have a unique network name. These may simply be names or they may use a scheme based on a hierarchical domain naming scheme. The names of the nodes must be unique, which is easy in small networks, more complex in large networks and impossible on the Internet unless some form of control is implemented. The names must have a maximum of 32 characters within the a-z, A-Z and 0-9 ranges and they may not contain spaces or # beginning with an alphabetic character.
- **Internet Address** (*IP address*): this consists of four numbers within the range of 0-255 separated by dots (for example, 192.168.0.1) and it is used universally to identify the computers on a network or on the Internet. The names are translated into IP addresses by a DNS (*domain name system*) server, that transforms the node names (legible to humans) in IP addresses (this service is performed by an application called *named*).
- **Port:** numerical identifier of the mailbox in a node that allows a specific application to read a message (TCP,UDP) (for example, two machines that communicate by telnet, will do so through port 23, but if they have a FTP transaction they will do so through port 21). There may be different ap-

Note

Name of the machine:
`more /etc/hostname`

Note

Machine IP address:
`more /etc/hosts`

Note

Pre-assigned ports in UNIX:
`more /etc/services`
This command shows the ports predefined with support to TCP or UDP communications.

Note

Visualisation of the routing's configuration:
`netstat -r`

plications communicating between two nodes through various different ports simultaneously.

- **Router node** (*gateway*): it is a node that performs the routing (data transfer). A router, depending on its characteristics, may transfer information between two similar or different network protocols and may also be selective.
- **Domain name system** (DNS): makes it possible to ensure one single name and to provide the administration of the databases that perform the translation between the name and Internet address and that are structured in the form of a tree. In order to do this, domains separated by points are defined, of which the highest (from right to left) describes a category, institution or country (COM stands for Commercial, EDU for Education, GOV for Governmental, MIL for Military (government), ORG, non-profit Organisation, XX which could be any two letters to indicate the country, or special cases, such as CAT to indicate Catalan language and culture etc.). The second level represents the organisation and the third and remaining sections indicate the departments, sections or divisions within an organisation (for example, `www.uoc.edu` or `nteum@pirulo.remix.es`). The first two names (from right to left), *uoc.edu* in the first case, *remix.es* (in the second) must be assigned (approved) by the SRI-NIC (global organisation that manages the Internet domain registry) and the rest may be configured/assigned by the institution.
- **DHCP, bootp**: DHCP and bootp are protocols that permit a client node to obtain information on the network (such as the node's IP address). Many organisations with many machines use this mechanism to facilitate the administration of large networks or networks in which there are roaming users.
- **ARP, RARP**: in some networks (such as IEEE 802 LAN, which is the standard for Ethernet), the IP addresses are dynamically discovered through the use of two other members of the Internet protocol suite: *address resolution protocol* (ARP) and *reverse address resolution protocol* (RARP). ARP uses broadcast messages to determine the Ethernet address (MAC specification for layer 3 of the OSI model), corresponding to a particular network-layer address (IP). RARP uses broadcast messages (messages that reach all of the nodes) to determine the network-layer address associated with a particular hardware address. RARP is especially important to diskless nodes, for which network-layer addresses are usually unknown at boot time.
- **Socket Library**: in UNIX, all TCP/IP implementation is part of the kernel of the operating system (either within the same or as a module that loads at boot time, as is the case with the device drivers in GNU/Linux).

Note

Domain and our DNS server is:

```
more /etc/default domain
more /etc/resolv.conf
```

Note

arp tables: :

```
arp to NameNode
```


The way for a programmer to use them is through an API (*application programming interface*) which implements this source-code interface. For TCP/IP, the most common API is the Berkeley Socket Library (Windows uses an equivalent library that is called Winsocks). This library makes it possible to create a communication end-point (socket), associate it to a remote node and port (bind) and offer the communication service (through *connect*, *listen*, *accept*, *send*, *sendto*, *recv*, *recvfrom*, for example). The library also provides a more general communication mode (AF_INET family) and more optimised communications for cases in which the process are communicating within the same machine (AF_UNIX family). In GNU/Linux, the socket library is part of the C standard library, Libc, (Libc6 in current versions), and it supports AF_INET, AF_UNIX, AF_IPX (for Novell protocols), AF_X25 (for the X.25 protocol), AF_ATMPVC-AF_ATMSVC (for the ATM protocol) and AF_AX25, F_NETROM, AF_ROSE (for *amateur radio protocol*).

3. How to assign an Internet address

This address is assigned by the NIC and it has two section or parts. The one on the left represents network identification and the one on the right represents the node identification. In consideration of the point mentioned above (four numbers between 0-255, or 32 bits or four bytes), each byte represents either the network or the node. The NIC assigns the net and the institution (or provider) assigns the node.

There are some restrictions: **0** (for example, 0.0.0.0) in the network space is reserved for the routing by default and **127** (for example, 127.0.0.1) is reserved for the (local loopback or local host), **0** in the node part refers to this network (for example, 192.168.0.0) and **255** is reserved for sending packets to all (broadcast) machines (for example, 198.162.255.255). There may be different types of networks or addresses in the different assignments:

Class A (*network.host.host.host*): 1.0.0.1 to 126.254.254.254 (126 networks, 16 million nodes) define the large networks. The binary standard is: **0** + 7 network bits + 24 node bits.

Class B (*network.network.host.host*): 128.1.0.1 to 191.255.254.254 (16K networks, 65K nodes); (usually, the first node byte is used to identify subnets within an institution). The binary standard is **10** + 14 network bits + 16 node bits.

Class C (*net.net.net.host*): 192.1.1.1 to 223.255.255.254 (2 million of networks, 254 nodes). The binary standard is **110** + 21 network bits + 8 node bits.

Classes D and E (*network.network.network.host*): 224.1.1.1 to 255.255.255.254 reserved for *multicast* (from one node to a set of nodes that form part of the group) and experimental purposes.

Some address ranges have been reserved so that they do not correspond to public networks, and are considered private networks (interconnected computers without external connection; the messages will not be sent through Internet, but through an intranet). These address ranges are **class A** 10.0.0.0 to 10.255.255.255, **class B** 172.16.0.0 to 172.31.0.0 and **class C** 192.168.0.0 to 192.168.255.0.

The broadcast address is special, because each node in a network listens to all the messages (as well as its own address). This address makes it possible to send datagrams (generally routing information and warning messages) to a

network and all nodes on the network will be able to read them. For example, when ARP tries to find the Ethernet address corresponding to an IP, it uses a broadcast message, which is sent to all the machines on the network at the same time. Each node in the network reads this message and compares the IP that is being searched and sends back a message to the sender node if they match.

Two concepts that are related to the point described above are the **subnets and routing** between these subnets. **Subnets** subdivide the node part into smaller networks within the same network, so as to, for example, improve the traffic. A subnet is in charge of sending traffic to certain IP address ranges, extending to the same concept of Class A, B and C networks, but only applying this rerouting in the IP node part. The number of bits interpreted as a subnet identifier is provided by a netmask, which is a 32-bit number (as is an IP). In order to obtain the subnet identifier, we will have to perform a logical AND operation between the mask and the IP, which will provide us with the subnet IP. For example, an institution with a B class network, with number 172.17.0.0, would therefore have a netmask with number 255.255.0.0. Internally, this network is formed by small networks (one per floor in the building, for example). In this way, the range of addresses is reassigned in 20 subnets (floors in our example, except 172.17.1.0, that has a special role), 172.17.1.0 to 172.17.20.0. The point that connects all these floors, called the backbone, has its own address, for example 172.17.1.0.

These subnets share the same network IP, whereas the third is used to identify each of the subnets within it (which is why it will use the netmask 255.255.255.0).

The second concept, **routing**, represents the mode in which the messages are sent through the subnets. For example, let us say there are three departments with Ethernet subnets:

- 1) Purchases (subnet 172.17.2.0),
- 2) Clients (subnet 172.17.4.0),
- 3) Human Resources, (subnet 172.17.6.0)
- 4) Backbone with FFDI (subnet 172.17.1.0).

In order to route the messages between the computers on the three networks, we need three gateways that will each have two network interfaces to switch between Ethernet and FFDI. These would be:

- 1) PurchasesGW IPs:172.17.2.1 and 172.17.1.1,
- 2) ClientsGW IPs:172.17.4.1 and 172.17.1.2
- 3) HumanResourcesGW IPs:172.17.6.1 and 172.17.1.3, in other words, one IP on the subnet side and another on the backbone side.

When messages are sent between machines in the purchases area, it is not necessary to leave the gateway, as the TCP/IP will find the machine directly. The problem arises when the Purchases0 machine wishes to send a message to HumanResources3. The message must pass through the two respective gateways. When Purchases0 "sees" that HumanResources3 is on another network, it sends the packet through the PurchasesGW gateway, which in turn sends it to HumanResourcesGW, which, in turn, sends it to HumanResources3. The advantage of having subnets is obvious, given that the traffic between all the purchases machines, for example, will not affect the Clients or Human Resources machines (although this is more complex and expensive in terms of designing and building the network).

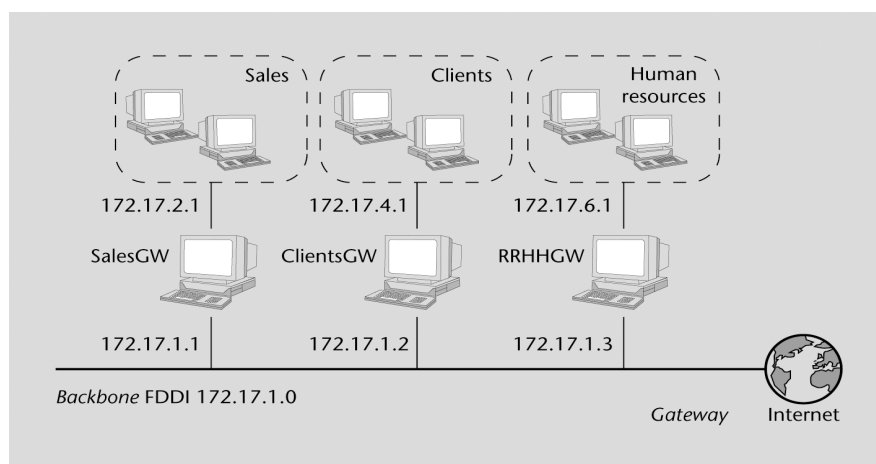


Figure 1. Configuration of segments and gateways in an intranet

IP uses a table to route the packets between the different networks, in which there is a default routing associated to net 0.0.0.0. All the addresses coincide with this one, as none of the 32 bits are necessary; they are sent through the default gateway to the indicated network. In the purchasesGW, for example, the table would be:

Address	Mask	Gateway	Interface
172.17.1.0	255.255.255.0	-	fddi0
172.17.4.0	255.255.255.0	172.17.1.2	fddi0
172.17.6.0	255.255.255.0	172.17.1.3	fddi0
0.0.0.0	0.0.0.0	172.17.2.1	fddi0
172.17.2.0	255.255.255.0	-	eth0

The '-' means that the machine is directly connected and does not need routing. The procedure for identifying whether routing is required or not consists of performing a very simple operation with the two logic ANDs (subnet AND

mask and origin AND mask) and comparing the two results. If they match, there is no routing, but the machine defined as gateway must be sent in each machine, so that this machine routes the message.

For example, a message from 172.17.2.4 to 172.17.2.6 would mean:

$$172.17.2.4 \text{ AND } 255.255.255.0 = 172.17.2.0$$

$$172.17.2.6 \text{ AND } 255.255.255.0 = 172.17.2.0$$

As the results are the same, there would be no routing. On the other hand, if we do the same from 172.17.2.4 to 172.17.6.6 we see that there will be routing through 172.17.2.1 with an interface change (*eth0* to *ffdi0*) to 172.17.1.1 and from here to 172.17.1.2 with another interface change (*fdi0* to *eth0*) and then to 172.17.6.6. The default routing will be used when none of the rules match. If two rules match, the routing that matches the most precisely, in other words, the one with the least zeros, will be used. In order to build the routing tables, we can use the *route* command during machine startup; however, if it is necessary to use more complex rules (or automatic routing), we can use the *routing information protocol* (RIP) command or, between independent systems, the *external gateway protocol* (EGP) or also the *border gateway protocol* (BGP) commands. These protocols are implemented through the *gated* command.

In order to install a machine on an existing network, it is necessary to have the following information, obtained from the network provider or the administrator: node IP address, network IP address, broadcast address, netmask address, router address and DNS address.

If we are setting up a network that will never have an Internet connection, we can choose the addresses that we wish, but it is advisable to maintain an appropriate order corresponding to the size of the network that will be needed, so as to avoid administrative problems within the network in question. We will now see how to define the network and node for a private network (we have to be careful, as, if the machine is connected to the network, we can inconvenience another user to whom this address has been assigned): node address 192.168.110.23, netmask 255.255.255.0, net part 192.168.110., node part .23, net address 192.168.110.0, broadcast address 192.168.110.255.

4. How to configure the network

4.1. Configuration of the network interface controller (NIC)

Once the GNU/Linux kernel has loaded, it executes the `init` command, which, in turn, reads the configuration file `/etc/inittab` and begins the start up process. Generally, the `inittab` has sequences such as: `si::sysinit: /etc/init.d/boot`, which represents the name of the commands file (script) that controls the booting sequences. Generally, this script calls the other scripts, which include the network startup script.

Example

In Debian, `/etc/init.d/network` is executed to configure the network interface, depending on the boot level; For example, in boot level 2, all the `S*` files in directory `/etc/rc2.d` (which are links to the `/etc/init.d` directory) will execute, and on the boot down level, all the `K*` files in the same directory. In this way, the script is only there once (`/etc/init.d`) and, depending on the services required in that status, a link is created in the directory corresponding to the node-status.

The network devices are created automatically when the corresponding hardware starts up. For example, the Ethernet driver creates the `eth[0..n]` interfaces sequentially, when the corresponding hardware is located.

The network interface may be configured as of that moment, which requires two steps: assign the network address to the device and boot the network parameters to the system. The command used for this is `ifconfig` (*interface configure*). An example might be:

```
ifconfig eth0 192.168.110.23 netmask 255.255.255.0 up
```

Which indicates that the `eth0` device should be configured with IP address 192.168.110.23 and netmask 255.255.255.0. *Up* indicates that the interface will be activated (to deactivate it, execute `ifconfig eth0 down`). If no values are specified, the command assumes that the default values should be used. In the previous example the kernel will configure this machine as a C-Type machine with IP=192.168.110.23 and the broadcast address=192.168.110.255.

There are commands, such as `ifup` and `ifdown`, that make it possible to configure/unconfigure the network more simply using the `/etc/network/interfaces` file to obtain all the necessary parameters (consult *man interfaces* for syntax).

Note

Consult
`man ifconfig`
for the different command options.

In Debian, there is another simpler method for configuring the network (considered high-level), which uses the abovementioned commands `ifup`, `ifdown` and the `/etc/network/interfaces` file. If we decide to use these commands, we should **not** configure the network at low-level, as these commands are sufficient for configuring/unconfiguring the network.

In order to modify the parameters of the `eth0` interface network, we can (consult *man interfaces* in section 5 of the Unix manual included with the operating system for more information):

<code>ifdown eth0</code>	for all network services over <code>eth0</code>
<code>vi /etc/network/interfaces</code>	edit and modify networks/interfaces parameters
<code>ifup eth0</code>	start up the network services over <code>eth0</code>

Let us suppose that we wish to configure an `eth0` interface in Debian, which has a fixed IP address `192.168.0.123` and has `192.168.0.1` as the gateway. We must edit `/etc/network/interfaces` so that it includes a section such as:

```
iface eth0 inet static
    address 192.168.0.123
    netmask 255.255.255.0
    gateway 192.168.0.1
```

If we have installed the `resolvconf` packet, we can add lines to specify the DNS information. For example:

```
iface eth0 inet static
    address 192.168.0.123
    netmask 255.255.255.0

    gateway 192.168.0.1
    dns-search remix.org
    dns-nameservers 195.238.2.21 195.238.2.22
```

After the interface has been activated, the command line arguments of the options `dns-search` and `dns-nameservers` are available for `resolvconf` for inclusion in `resolv.conf`. The command line argument `remix.org` of the `dns-search` option corresponds to the argument of the `search` option in `resolv.conf` (we will look at this in more detail later) and the arguments `195.238.2.21` and `195.238.2.22` of the `dns-nameservers` option corresponds to the arguments of the `nameserver` options in `resolv.conf` (consult `man resolv.conf`). It is also possible to configure the network at low-level through the `ip` command (which is equivalent to `ifconfig` and `route`). Although this command is much more versatile and powerful (it can be used to establish tunnels, alternate routings etc.), it is more complex and it is recommendable to use the preceding procedures for basic network configurations.

4.1.1. Configuration of network in Fedora style

Red Hat and Fedora use a different file structure for network configuration:

/etc/sysconfig/network. For example, to configure the network statically:

<pre>NETWORKING=yes HOSTNAME=my-hostname FORWARD_IPV4=true GATEWAY="XXX.XXX.XXX.YYY"</pre>	<i>Name of the host defined by the cmd hostname</i> <i>True for NAT firewall gateways and routers.</i> <i>False for any other case</i> <i>Gateway leading out to Internet</i>

To configure using DHCP, it is necessary to delete the GATEWAY line, as it will be assigned by the server. And if NIS is to be incorporated, a line with the server domain must be added: NISDOMAIN=NISProject1

To configure interface eth0 in the file

/etc/sysconfig/network-scripts/ifcfg-eth0:

```
DEVICE=eth0
BOOTPROTO=static
BROADCAST=XXX.XXX.XXX.255
IPADDR=XXX.XXX.XXX.XXX
NETMASK=255.255.255.0
NETWORK=XXX.XXX.XXX.0
ONBOOT=yes Activates the network on boot.
```

From FC3 on, it is also possible to add:

```
TYPE=Ethernet
HWADDR=XX:XX:XX:XX:XX:XX
GATEWAY=XXX.XXX.XXX.XXX
IPV6INIT=no
USERCTL=no
PEERDNS=yes
```

Or else, for configuring using DHCP :

```
DEVICE=eth0
ONBOOT=yes
BOOTPROTO=dhcp
```

To disable DHCP, change BOOTPROTO=dhcp to BOOTPROTO=none. Any change in these files must restart the services with service network restart (or, otherwise, /etc/init.d/network restart).

The following three steps must be taken to change the hostname:

- 1) Command `hostname new-name`.
- 2) Change the network configuration in `/etc/sysconfig/network` editing `HOSTNAME=new-name`.
- 3) Restoring all the services (or *rebooting*):
 - `service network restart` (or executing `/etc/init.d/network restart`)
 - Restarting the desktop by passing into console mode `init 3` and changing to GUI mode `init 5`.

Verifying if the name is not registered in `/etc/hosts`. The hostname may be changed during execution time with `sysctl -w kernel.hostname=newname`.

4.1.2. Configuration of a Wi-Fi (wireless) network

In order to configure Wi-Fi interfaces, we basically use the `wireless-tools` package (as well as `ifconfig` or `ip`). This package uses the `iwconfig` command to configure a wireless interface, but this can also be carried out through `/etc/network/interfaces`.

Example: Configure WiFi in Debian Sarge (Etch) (similar in FC6)

Let's assume that we wish to configure an Intel Pro/Wireless 2200BG wireless network card (very common in many laptops, such as Dell, HP...). The software that controls the cards is usually divided into two parts: the software module that will be loaded in the kernel through the `modprobe` command and the firmware that is the code that will be loaded in the card and which is given to us by the manufacturer (consult the Intel site for this model). As we are discussing modules, it is interesting to use the Debian module-assistant package which allows us to create and install a module easily (another option would be to install the sources and create the corresponding module). We will compile and install the software (which we can find on the manufacturers' website and is called `ipw2200`) using the `m-a` command in the module-assistant package.

```
apt-get install module-assistant (install the package)
m-a -t update
m-a -t -f get ipw2200
m-a -t -build ipw2200
m-a -t install ipw2200
```

We can download the compatible firmware version from the site address provided by the manufacturer (in the product documentation) along with the version of the driver we need, which in our case, would be driver version 1.8 and firmware version 2.0.4, obtained from the following address:

<http://ipw2200.sourceforge.net/firmware.php>

We should then decompress and install the firmware:

```
tar xzvf ipw2200fw2.4.tgz C /tmp/fwr/
cp /tmp/fwr/*.fw /usr/lib/hotplug/firmware/
```

This will copy three packages (`ipw2200-bss.fw`, `ipw2200-ibss.fw` and `ipw2200-sniffer.fw`). The module is then loaded with: `modprobe ipw2200`, the system reboots and then, from the console, we can execute the `dmesg | grep ipw` command, which will show us some lines similar to the ones below and which indicate that the module is loading (this can be checked with `lsmod`):

```
ipw2200: Intel(R) PRO/Wireless 2200/2915 Network Driver, git1.0.8
ipw2200: Detected Intel PRO/Wireless 2200BG Network Connection
...
```

We should then download the wireless tools package that contains `iwconfig` in order to install wireless tools with `aptget`, among others, and if we execute `iwconfig`, something similar to the following will display:

```
eth1 IEEE 802.11b ESSID:"Name-of-the-Wifi"
Mode:Managed Frequency:2.437 GHz
Access Point:00:0E:38:84:C8:72
Bit Rate=11 Mb/s TxPower=20 dBm
Security mode:open
...
```

We must then configure the network file, for example, `gedit /etc/network/interfaces`, and add the `eth1` wifi interface, for example:

```
iface eth1 inet dhcp
    pre-up iwconfig eth1 essid "Name of the Wifi"
    pre-up iwconfig eth1 key open XXXXXXXXXX
```

The pre-up lines execute the `iwconfig` command before activating the interface. This configuration is used if we wish to use the service in DHCP mode (automatic IP assignation, as we shall see). Instead of DHCP, the word `static` should be used and the following lines, as an example, must be entered (as in a cable card):

```
address 192.168.1.132
netmask 255.255.255.0
network 192.168.0.0
broadcast 192.168.0.255
gateway 192.168.1.1
```

Another method for configuring the interface is:

```
iface eth1 inet dhcp
    wireless-essid "Name of the Wifi"
    wireless-key 123456789e
```

We can then start up the network with `ifup eth1` and we will be given information on the connection and the state and quality of reception. In order to scan the available WiFi networks (access points), we can use `iwlist scan`, which will show us information on the available networks, and if we want to connect to a different network, we can use the `iwconfig` command to change the network or Access Point.

4.2. Configuration of Name Resolver

The next step is to configure the name resolver, which changes names like `pirulo.remix.com` to `192.168.110.23`. The `/etc/resolv.conf` file is used for this. The format is very simple (one line of text per sentence). There are three key words for this purpose: *domain* (local domain), *search* (list of alternate domains) and *name server* (IP address of the *domain name server*).

Example of `/etc/resolv.conf`

```
domain remix.com
search remix.com piru.com
name server 192.168.110.1
name server 192.168.110.65
```

This list of name servers often depends on the network environment, which may change depending on where the machine is or where it is connected. The programs for connecting to telephone lines (`pppd`) or obtaining IP addresses automatically (`dhclient`) can modify `resolv.conf` to insert or delete servers; but these characteristics do not always work properly and they can sometimes generate conflicts or incorrect configurations. The `resolvconf` package adequately solves the problem and allows us to configure the name servers easily and dynamically. `resolvconf` is designed to work without the user having to configure anything manually; however, the package is quite new and may require some manual assistance to make it work properly. For more information:

<http://packages.debian.org/unstable/net/resolvconf>

Another important file is `/etc/host.conf`, which can be used to configure the behaviour of the name resolver. This file is very important because it indicates where the node address or name is first resolved. This can be consulted in the DNS server or the local tables within the existing machine (`/etc/hosts`).

Example of `/etc/host.conf`

```
order hosts,bind
multi on
```

This configuration indicates that `/etc/hosts` should be verified first consulting the DNS and it also indicates (2nd line) that all valid addresses found in `/etc/hosts` should be returned. Consequently, the `/etc/hosts` file is where the local addresses are placed and it can also be used to access the nodes without having to consult the DNS.

The consulting process is much faster, but the disadvantage is that, if the node changes, the address will be incorrect. In a system that is properly configured, only the local node and an input for the loopback interface should appear.

Example of /etc/hosts

127.0.0.1	localhost	loopback
192.168.1.2	pirulo.remix.com	pirulo

Aliases may be used for the name of a machine; this means that this machine may have different names for the same IP address. The loopback interface is a special type of interface that makes it possible for a node to connect to itself (for example, to verify that the network subsystem is working without accessing the net). By default, the IP address 127.0.0.1 has specifically been assigned to the loopback (a `telnet 127.0.0.1` command will connect with the same machine). Configuring aliases is very easy (generally, the network startup script configures them).

Example of loopback

```
ifconfig lo 127.0.0.1
route add host 127.0.0.1 lo
```

In version 2 of the GNU library, there is an important replacement with regard to the functions of the `host.conf` file. This improvement includes the centralisation of information on different services for name resolution, which provides many advantages for the network administrator. All the information on name and service consultations has been centralised in the `/etc/nsswitch.conf` file, which allows the administrator to configure the order and the databases in a very simple manner. In this file, each service appears, one per line, with a set of options, such as the node name resolution option. This indicates that the order for consulting the databases for obtaining the node's IP or its name will be first through the DNS service (which uses the `/etc/resolv.conf` file to determine the IP of the DNS node) and then, if it cannot be obtained here, the databases of the local (`/etc/hosts`) will be used. Other options for this could be `nis` or `nisplus`, which are other information services that are explained in subsequent units. The method for each consultation may also be controlled through actions (between `[]`), for example:

```
hosts: xfn nisplus dns [NOTFOUND = return] files
```

This indicates that, when the DNS is consulted, if there is no registry for this consultation, the program that made the consultation will return a zero. The `!` may be used to deny the action, for example:

```
hosts dns [!UNAVAIL = return] files
```

Note

Example of `nsswitch.conf`: ...
hosts: dns files
...
networks: files

4.3. Configuration of routing

Another aspect that has to be configured is the routing. Although the process is considered to be very complex, in general, the routing requirements are very simple. In a node with multiple connections, routing consists of deciding where to send and what to receive. A simple node (one single network connection) also needs routing, given that all the nodes have a loopback and a network connection (for example, Ethernet, PPP, SLIP...). As we have explained, there is a table known as a routing table that contains rows with various fields, three of which are especially important: destination address, interface through which the message will be sent and IP address, which will take the next step in the gateway.

Note

Consultation of routing tables:

```
route -n  
or also  
netstat -r
```

The `route` command can be used to modify this table so as to carry out the appropriate routing tasks. When a message arrives, the destination address is examined, compared with the entries in the table and sent through the interface with the address that most resembles the packet's destination. If a gateway is specified, it is sent to the appropriate interface.

Let us assume, for example, that our node is in a C class network with the address 192.168.110.0 and the address is 192.168.110.23; and the router connected to the Internet is 192.168.110.3. The configuration will be:

- First, the interface:
`ifconfig eth0 192.168.110.23 netmask 255.255.255.0 up`
- Subsequently, indicate that all the datagrams for nodes with 192.168.0.* addresses must be sent to the network device:
`route add -net 192.1 ethernetmask 255.255.255.0 eth0`

`-net` indicates that it is a network route but `-host 192.168.110.3.` may also be used. This configuration will allow it to connect with all the nodes within a network segment (192.1), but, what would happen if we wanted to connect with another node outside this segment? It would be very difficult to have all the appropriate entries for all the machines to which we wish to connect. To simplify this task, we have the *default route*, which is used when the destination address does not match any of the entries in the table. One configuration possibility would be:

`route add default gw 192.168.110.3 eth0`

(the gw is the IP or name of a gateway or router node).

Another method of doing this would be:

```
ifconfig eth0 inet down disable the interface
ifconfig lo
    Link encap:Local Loopback
... (no entries for eth0 will appear)
route
... (no entry in the routing table will appear)
```

Subsequently, the interface is enabled with another IP and a new route:

```
ifconfig eth0 inet up 192.168.0.111 \
    netmask 255.255.0.0 broadcast 192.168.255.255
route add -net 10.0.0.0 netmask 255.0.0.0 \
    gw 192.168.0.1 dev eth0
```

The bar (\) indicates that the command continues on the following line. The result:

```
ifconfig
    eth0 Link encap:Ethernet HWaddr 08:00:46:7A:02:B0
        inet addr:192.168.0.111 Bcast: 192.168.255.255 Mask:255.255.0.0
        UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
        ...
    lo Link encap:Local Loopback
        inet addr:127.0.0.1 Mask:255.0.0.0
        ...

route
    Kernel IP routing table
```

Destination	Gateway	Genmask	Flags	Metric	Ref	Use	Iface
192.168.0.0	*	255.255.0.0	U	0	0	0	eth0
10.0.0.0	192.168.0.1	255.0.0.0	UG	0	0	0	eth0

For more information, see the `ifconfig` (8) and `route` (8) commands.

4.4. Configuration of `inetd`

The next step in the configuration of the network is to configure the servers and services that will allow another user to access the local machine or its services. The server programs will use the ports to listen to the requests from the clients, which will be sent to this service as IP:port. The servers may work in two different ways: standalone (in which the service listens to the assigned port and is always active) or through `inetd`.

The `inetd` is a server that controls and manages the network connections of the services specified in the `/etc/inetd.conf` file, which, when a service request is made, starts up the appropriate server and transfers the request.

Two important files must be configured: `/etc/services` and `/etc/inetd.conf`. In the first file, we associate the services, the ports and the protocol, and in the second, the server programs that will respond to a request to a determined port. The `/etc/services` format is `name port/protocol aliases`, where the first field is the service name, the second is the port where the service is attended and the protocol that it uses, and the next field is an alias of the name. There is a series of default pre-configured services. We will now show an example of `/etc/services` (`#` indicates that what follows is a comment):

<code>tcpmux</code>	<code>1/tcp #</code>	<code># TCP port service multiplexer</code>
<code>echo</code>	<code>7/tcp</code>	
<code>echo</code>	<code>7/udp</code>	
<code>discard</code>	<code>9/tcp</code>	<code>sink null</code>
<code>discard</code>	<code>9/udp sink</code>	<code>sink null</code>
<code>systat</code>	<code>11/tcp</code>	<code>users</code>
<code>...</code>		
<code>ftp</code>	<code>21/tcp</code>	
<code>ssh</code>	<code>22/tcp</code>	<code># SSH Remote Login Protocol</code>
<code>ssh</code>	<code>22/udp</code>	<code># SSH Remote Login Protocol</code>
<code>telnet</code>	<code>23/tcp</code>	
<code># 24 - private</code>		
<code>smtp</code>	<code>25/tcp</code>	<code>mail</code>
<code>...</code>		

The `/etc/inetd.conf` file is the configuration for the master network service (*inetd server daemon*). Each line contains seven fields separated by spaces: *service socket_type proto flags user server_path server_args*, where *service* is the service described in the first column in `/etc/services`, *socket_type* is the type of socket (possible values are *stream*, *dgram*, *raw*, *rdm*, or *seqpacket*), *proto* is the protocol that is valid for this input (it must match that in `/etc/services`), *flags* indicates the action that should be taken when there is a new connection on a service that is attending another connection, (*wait* tells *inetd* not to start up a new server or *nowait* means that *inetd* must start up a new server). *user* will be the local user-name with which the client that has started up the service is identified, *server_path* is the directory where the server is located and *server_args*

are possible arguments that will be passed to the server. An example of some `/etc/inetd.conf` lines is (# is a comment, so if a service has # before the name, it means that it is not available):

```
...
telnet stream tcp nowait root /usr/sbin/tcpd /usr/sbin/in.telnetd
ftp stream tcp nowait root /usr/sbin/tcpd /usr/sbin/in.ftpd
# fsp dgram udp wait root /usr/sbin/tcpd /usr/sbin/in.fspd
shell stream tcp nowait root /usr/sbin/tcpd /usr/sbin/in.rshd
login stream tcp nowait root /usr/sbin/tcpd /usr/sbin/in.rlogind
# exec stream tcp nowait root /usr/sbin/tcpd /usr/sbin/in.rexecd...
...
```

As of Debian Woody 3.0 r1, the `inetd` function has been replaced by `xinetd` (recommendable), which needs the `/etc/xinetd.conf` configuration file (see end of unit). If we wish to start up the *inetd* service, we must execute (and create the appropriate links in the `/etc/rcX.d` directories) `/etc/init.d/inetd.real start` (see the end of this chapter for examples of configurations).

Apart from the `inetd` or `xinetd` configuration, the typical configuration of network services in a desktop or basic server environment might also include (some of these services will be examined in the chapter on servers):

- **ssh:** secure interactive connection to replace telnet that includes two configuration files `/etc/ssh/ssh_config` (for the client) and `/etc/ssh/sshd_config` (for the server)
- **exim:** multi transfer agent (MTA), includes configuration files: `/etc/exim/exim.conf`, `/etc/mailname`, `/etc/aliases`, `/etc/email-addresses`.
- **fetchmail:** daemon for downloading the mail from a POP3 account, `/etc/fetchmailrc`
- **procmail:** program for filtering and distributing local mail, `~/.procmailrc`
- **tcpd:** Filtering services for enabled and disabled machines and domains for connecting to the server (wrappers): `/etc/hosts.allow`, `/etc/hosts.deny`
- **DHCP.** Service for managing (server) or obtaining an IP (client), `/etc/dhcp3/dhclient.conf` (client), `/etc/default/dhcp3-server` (server), `/etc/dhcp3/dhcpd.conf` (server)

- **CVS:** system for managing concurrent versions, `/etc/cvs-cron.conf`, `/etc/cvs-pserver.conf`
- **NFS:** network file system, `/etc/exports`
- **Samba:** network file system and sharing printers in Windows networks, `/etc/samba/smb.conf`
- **lpr:** daemon for the printing system, `/etc/printcap` (for the lpr system -not CUPS-)
- **Apache and Apache2:** Web Server, `/etc/apache/*` and `/etc/apache2/*`
- **squid:** Server proxy-cache, `/etc/squid/*`

4.5. Additional configuration: protocols and networks

There are other configuration files that are hardly ever used, but that can be interesting. The `/etc/protocols` is a file that shows the protocol identifiers with the protocol names; in this way, the programmers can specify the protocols by their names in the programs.

Example of `/etc/protocols`

ip	0	IP	# internet protocol, pseudo protocol number
#hopopt	0	HOPOPT	# IPv6 Hop-by-Hop Option [RFC1883]
icmp	1	ICMP	# internet control message protocol

The `/etc/networks` file has a function similar to `/etc/hosts`, but where the networks are concerned, it shows the network names in relation to its IP address (the route command will show the name of the network and not its address in this case).

Example of `/etc/networks`

```
loopnet 127.0.0.0
localnet 192.168.0.0
amprnet 44.0.0.0 ...
```

4.6. Security aspects

It is important to take into account the security aspects in network connections, as a significant amount of attacks occur through the network. We will discuss this subject in more detail in the unit on security; however, there are

some basic recommendations that should be taken into account in order to minimise the risks immediately before and after configuring the network in our computer:

- Do not activate services in `/etc/inetd.conf` that will not be used, insert an `#` before the name to avoid sources of risk.
- Modify the `/etc/ftpusers` file to deny access to certain users who may have an FTP connection to your machine.
- Modify the `/etc/securetty` file to indicate from which terminals (a name per line), for example: `tty1 tty2 tty3 tty4`, it will be possible for the root superuser to connect. The root superuser will not be able to connect from any of the remaining terminals.
- Use the `tcpd` program. This server is a wrapper that makes it possible to allow/deny a service from a given node and it is placed in `/etc/inetd.conf` as a service intermediary. The `tcpd` verifies certain access rules in two files:
`/etc/hosts.allow` `/etc/hosts.deny`.

If the connection is accepted, it starts up an appropriate service passed as an argument (for example, the FTP service line shown earlier in `inetd.conf`:

```
ftp stream tcp nowait root/usr/sbin/tcpd/usr/sbin/in.ftpd.
```

`tcpd` first search `/etc/hosts.allow` and then inside of `/etc/hosts.deny`. The `hosts.deny` file contains the rules on which nodes do not have access to a service within this machine. A restrictive configuration is `ALL: ALL`, as it will only allow access to the services from the nodes declared in `/etc/hosts.allow`.

The `/etc/hosts.equiv` file permits access to this machine without having to enter the password. Using this mechanism is not recommended; users should be advised not to use the equivalent from the user account, through the `.rhosts` file.

In Debian, it is important to configure `/etc/security/access.conf`, the file that indicates the rules on who and from where it is possible to log in to this machine. This file has a line by command with three fields separated by a ':' of the permission type: Users: origin. The first will be an +o- (allow or deny), the second a user name/user names, group or user@host, and the third will be the name of a device, node, domain, node or networks addresses or ALL.

Example of access.conf

This command does not permit root logins over *tty1*:

```
ALL EXCEPT root:tty1 ...
```

It permits access to *u1*, *u2*, *g1* and all those in the *remix.com* domain:

```
+:u1 u2 g1 .remix.com:ALL
```

4.7. IP Options

There are further options with regard to IP traffic that we should mention. This is configured by starting up the corresponding file in the `/proc/sys/net/ipv4/` directory. The file name is the same as the command and a 1 must be placed in the file to activate them, or a 0 to deactivate them.

Example

For example, if we wish to activate `ip_forward`, we have to execute:

```
echo 1 > /proc/sys/net/ipv4/ip_forward
```

The most widely used are: `ip_forward` used for routing between interfaces or with IP Masquerading; `ip_default_ttl`, which is the lifetime for an IP packet (64 milliseconds, by default) `ip_bootp_agent` logical variable (BOOLEAN) which accepts packets (or not) with the origin address of the 0.b.c.d type and the destination of this node, broadcast or multicast.

4.7.1. Commands for solving problems with the network

If there are problems in the configuration of the network, we can begin by verifying the output of the following commands to obtain an initial idea:

```
ifconfig
cat /proc/pci
cat /proc/interrupts
dmesg | more
```

In order to verify the network connection, we can use the following commands (`netkit-ping`, `traceroute`, `dnsutils`, `iptables` and `net-tools` must be installed):

```
ping uoc.edu           # verifies the Internet connection
traceroute uoc.edu      # scans IP packets
ifconfig               # verifies the host configuration
route -n               # verifies the routing configuration
dig [@dns.uoc.edu] www.uoc.edu # verifies the registries in
                        # on the dns.uoc.edu server.
iptables -L -n |less    # verifies packet filtering (kernel >=2.4)
netstat -a              # shows all the open ports
```

```
netstat -l --inet      # shows all the listening ports
netstat -ln --tcp      # shoos the listening tcp ports (number)
```

5. DHCP Configuration

DHCP stands for dynamic host configuration protocol. It is very simple to configure and it is useful because, instead of having to configure each node in a network individually, this can be done in a centralised manner and administering it is therefore easier. The configuration of a client is very easy, as we only have to install one of the following packages: `dhcp3-client` (version 3, Internet Software Consortium), `dhcpcd` (Yoichi Hariguchi and Sergei Viznyuk), `pump` (Red Hat); we then add the word *dhcp* in the section corresponding to the interface that we wish to work under this dhcp client (e.g./etc/network/interfaces must have `iface eth0 inet dhcp...`).

Configuring the server requires more care, but it is not especially complicated. First, for the server to serve all the DHCP clients (including Windows), we must address some questions concerning the broadcast addresses. In order to do this, first the server must be able to send messages to the 255.255.255.255 address, which is not valid in GNU/Linux. In order to try this, execute:

```
route add -host 255.255.255.255 dev eth0
```

If the following message appears: *255.255.255.255: Unknown host*, then the following entry must be added in `/etc/hosts`: *255.255.255.255 dhcp* and try again:

```
route add -host dhcp dev eth0
```

The configuration of `dhcpcd` can be carried out with the graphic interface of `linuxconf` (not advisable) or by editing `/etc/dhcpd.conf`. An example of this file is:

```
# Example of /etc/dhcpd.conf:
default-lease-time 1200;
max-lease-time 9200;
option domain-name "remix.com";
deny unknown-clients;
deny bootp;
option broadcast-address 192.168.11.255;
option routers 192.168.11.254;
option domain-name-servers 192.168.11.1,192.168.168.11.2;
subnet 192.168.11.0 netmask 255.255.255.0
{ not authoritative;
    range 192.168.11.1 192.168.11.254
    host marte {
        hardware ethernet 00:00:95:C7:06:4C;
```

```
        fixed address 192.168.11.146;
        option host-name "marte";
    }
    host saturno {
        hardware ethernet 00:00:95:C7:06:44;
        fixed address 192.168.11.147;
        option host-name "saturno";
    }
}
```

This will allow the server to assign the address range from 192.168.11.1 to 192.168.11.254, as described for each node. If the corresponding host { ... } segment does not exist, they will be assigned randomly. The IPs are assigned for a minimum time of 1,200 seconds and a maximum of 9,200 (if these parameters do not exist, they will be assigned indefinitely).

Before executing the server, we must verify if the file `/var/state/dhcp/dhcpd.leases` exists (otherwise, it will have to be created with `touch /var/state/dhcp/dhcpd.leases`). To execute the server: `/usr/sbin/dhcpd` (or we can put it in the startup scripts). With `/usr/sbin/dhcpd -d-f`, we can see the activity in the server within the system's console. [Mou01, Rid00, KD00, Dra99]

It is important not to forget the `not authoritative` phrase, as, otherwise, this server may leave other dhcp servers that serve IP for other segments inactive.

6. IP aliasing

There are some applications in which it is useful to configure multiple IP addresses to a single network device. The ISPs (*Internet service providers*) frequently use this characteristic to provide personalised features (such as World Wide Web and FTP) to their users. For this, the kernel must be compiled with the Network Aliasing and IP (aliasing support) options. After installing the new kernel, the configuration is very easy. The aliases are attached to the virtual network devices associated with the new device with a format such as: device: virtual number.

For example: eth0:0, ppp0:8

Let us say that we have an Ethernet network that supports two different IP subnets simultaneously and that our machine wants to have direct access to them. An example of the configuration would be:

```
ifconfig eth0 192.168.110.23 netmask 255.255.255.0 up
route add -net 192.168.110.0 netmask 255.255.255.0 eth0
ifconfig eth0:0 192.168.10.23 netmask 255.255.255.0 up
route add -net 192.168.10.0 netmask 255.255.255.0 eth0:0
```

Which means that we would have two IPs, 192.168.110.23 and 192.168.10.23 for the same NIC. In order to delete an alias, add a '-' at the end of the name (for example, ifconfig eth0:0- 0). [Mou01, Ran05]

A typical case is when we wish to configure a single Ethernet card so that it acts as the interface for different IP subnets. For example, suppose we have a machine that is on a LAN network, LAN 192.168.0.x/24. And we wish to connect the machine to the Internet using a public IP address provided with DHCP using the existing Ethernet card. For example, we can follow the procedure described in the preceding example or edit the /etc/network/interfaces file so that it includes a section similar to the following:

```
iface eth0 inet static
address 192.168.0.1
netmask 255.255.255.0
network 192.168.0.0
broadcast 192.168.0.255

iface eth0:0 inet dhcp
```

The eth0:0 interface is a virtual interface and its parent interface, eth0, will activate when it does.

7. IP Masquerade

The IP Masquerade is a resource used so that a set of machines may use a single IP address. This permits the hidden nodes (in other words, the ones that use a private IP, such as 198.162.10.1) can go out to the Internet; but they cannot directly accept external calls or services; only through the machine that has the real IP.

This means that some services will not work (for example, talk) and others must be configured in PASV (passive) mode for them to work (for example, FTP). However, WWW, telnet or IRC will work properly. The kernel must be configured with the following options: Network firewalls, TCP/IP networking, IP: forwarding/gatewaying, IP: masquerading. Normally, the most common configuration is to have a machine with a SLIP or PPP connection and to have another network device (for example, an Ethernet card) with a reserved net address. As we have seen and as described in RFC 1918, the following address ranges (IP/Mask) can be used as private IPs: 10.0.0.0/255.0.0.0, 172.16.0.0/255.240.0.0, 192.168.0.0/255.255.0.0. The nodes that must be masqueraded will be on this second network. Each of these machines must have the address of the machine that is masquerading such as default gateway or router. On this machine, we can configure:

- Network route for Ethernet considering that the network has a IP = 192.168.1.0/255.255.255.0:

```
route add -net 192.168.1.0 netmask 255.255.255.0 eth0
```
- Default route for the rest of Internet:

```
route add default ppp0
```
- All the nodes over the 192.168.1/24 network will be masqueraded:

```
ipchains -A forward -s 192.168.1.0/24 -j MASQ
```
- If iptables are used over a kernel, version 2.4 or higher:

```
iptables -t nat -A POSTROUTING -o ppp0 -j MASQUERADE
```

Consult the references in the unit covering security for information on *ipchains* and *iptables*. [Ran05, KD00]

8. NAT with kernel 2.2 or higher

The IP *network address translation*, NAT, is a replacement that has made the features of GNU/Linux IP Masquerade obsolete and that provides new features to the service. One of the improvements included in the TCP/IP stack of GNU/Linux 2.2 is that NAT is integrated into the kernel. In order to use it, we have to compile the kernel with:

CONFIG_IP_ADVANCED_ROUTER, CONFIG_IP_MULTIPLE_TABLES and CONFIG_IP_ROUTE_NAT.

And if we need comprehensive control of the NAT rules (for example, to activate the firewall we must also have

CONFIG_IP_FIREWALL and CONFIG_IP_ROUTE_FWMARK.

In order to work with these new features, we need to use the ip program (which can be obtained at ftp://ftp.inr.ac.ru/ip_routing/). Then, to translate the incoming datagram addresses, we can use:

```
ip route add nat <extaddr>[/<masklen>] via <intaddr>
```

This will translate the destination address of an incoming packet addressed to ext-addr (the address that is visible externally from Internet) to int-addr (the address of the internal network through the gateway/firewall). The packet is routed in accordance with the local route table. Single or block addresses can be translated. For example:

```
ip route add nat 240.0.11.34 via 192.109.0.2
ip route add nat 240.0.11.32/27 via 192.109.0.0
```

The first makes the internal address 192.109.0.2 accessible as 240.0.11.34. The second remaps the 192.109.0.0/31 block to 240.0.11.32/63. In this case, we have used, as an example, translations to class D and E addresses, such as 240.0.*.* so as not to use a public address. The user must replace these addresses (240.0.11.34 and 240.0.11.3263) for the corresponding public addresses to which they wish to translate. [Ran05]

9. How to configure a DialUP and PPP connection

Configuring a dial-up connection using PPP in GNU/Linux is very simple. PPP (*point to point protocol*) makes it possible to establish *IP-Links* between two computers with a modem (that it must be a modem supported by GNU/Linux, as not all modems, especially internal ones or those known as Winmodems, can be configured because many of them need additional software in order to establish communication). [Vas00, Law07, Sec00].

To start with, we must have the following information: the modem init-string (this is not normally necessary but if it is and it is not available, we can use ATZ, which works in most modems or we can consult specialised init-string lists).

We also need the ISP data: connection ID (login name), password and telephone number. The DNS addresses would be advisable, but this is optional in the current versions of pppd. Also, we should verify that the modem is connected properly. With an external modem, we must execute `echo > /dev/ttyS0` and check the LEDs on the modem to see if it is active. Otherwise, try with `ttyS1`, if the modem is connected to the 2nd serial port. With an internal modem, check the supported hardware manual to see if this modem can be recognised by GNU/Linux; if this is the case, it may be necessary to reconfigure the kernel in order to use it. We can also use `cat /proc/pci` in case it is in the PCI bus. [PPP00]

The easiest way to configure the modem now is through the kppp package (we must install the `kdenetwork-ppp*` and `ppp*` packages). On a terminal, execute `/usr/bin/kppp`. On the window, fill in the following boxes:

Accounts ⇒ New Connection

Dial ⇒ Authentication 'PAP/CHAP'

Store Password ⇒ yes

IP ⇒ Dynamic IP Address

Autoconfigure hostname ⇒ No

Gateway ⇒ Default Gateway ⇒ Assign the Default Route

DNS ⇒ Configuration Automatic ⇒ Disable existing DNS

Device ⇒ `ttyS1(com1)` o `ttyS2 (com2)`

Modem ⇒ Query Modem to see the results (if you do not obtain the results, change the `ttySx` device).

After entering the login name and password, we will be connected to the Internet (to check that we are connected, execute *ping www.google.com* for example). Here, we have used the *kppp* package, but we could as easily have used *linuxconf* or *gnomeppp* indistinctly).

A quick way of configuring *pppd* in Debian consists of using the *pppconfig* program, which comes with the package. *pppconfig* configures files such as the preceding ones after asking the user some questions through the menu interface. Another option for using *pppd* consists of executing it from *wvdial*, which comes with the *wvdial* package. Instead of making *pppd* execute *chat* to dial and negotiate the connection, *wvdial* dials, carries out the initial negotiation and then starts up *pppd* so that it can do the rest. In most cases, with just the telephone number, username and password, *wvdial* can start the connection.

Once PPP has been configured, for it to work with, for example, *my_isp*, we must edit */etc/network/interfaces* so that it includes a section such as the following (the *ifup*, *ifdown* commands use the *pon* and *poff* commands to configure PPP interfaces):

```
iface ppp0 inet ppp
provider mi_isp
```

with this section, *ifup ppp0* executes:

```
pon my_isp
```

It is not currently possible to use *ifup* down to perform a supporting configuration of the PPP interfaces. As *pon* disappears before *pppd* has finished establishing the connection, *ifup* executes the up scripts before the PPP interface is ready to be used. Until this fault is resolved, it will still be necessary to configure the connection later in */etc/ppp/ip-up* or */etc/ppp/ip-up.d/*.

Many broadband Internet Service Providers (ISP) use PPP to negotiate the connection even when the clients' machines are connected through Ethernet and/or ATM networks. This is achieved through PPP over Ethernet (PPPoE) which is a technique for encapsulating PPP flow within Ethernet frames. Suppose that the ISP is called *my_isp*. First, we must configure PPP and PPPoE for *my_isp*. The easiest way of doing this consists of installing the *pppoeconf* package and executing *pppoeconf* from the console. We then edit */etc/network/interfaces* so that it includes a fragment such as the following:

```
iface eth0 inet ppp
provider mi_isp
```

Sometimes, problems arise with PPPoE that are related to the *maximum transmit unit* (or MTU) in DSL (*digital subscriber line*) lines; you may consult DSL-HOWTO for further details. If the modem has a router, as the modem/router will handle the PPPoE connection on its own and it will appear on the LAN side as a simple Ethernet to Internet gateway.

10. Configuring the network through *hotplug*

The *hotplug* package supports hot swapping when booting (the package in question must have been installed). The network hardware can be hot plugged either at start up, after inserting the card in the machine (a PCMCIA card, for example) or after a utility such as *discover* has been executed and the necessary modules have been loaded. When the kernel detects new hardware, it starts up the driver for the hardware and then executes the *hotplug* program to configure it. If the hardware is subsequently removed, the program executes *hotplug* again, with different parameters. In Debian, when *hotplug* is called, this executes the scripts of `/etc/hotplug/` and `/etc/hotplug.d/`. The network hardware that was recently connected is configured by `/etc/hotplug/net.agent`. Let us assume that the PCMCIA network card has been connected, which would mean that the `eth0` interface would be ready to be used. `/etc/hotplug/net.agent` performs the following:

```
ifup eth0=hotplug
```

Unless a logical interface called *hotplug* has been added in `/etc/network/interfaces`, this command will have no effect. For this command to configure `eth0`, we have to add the following lines to `/etc/network/interfaces`:

```
mapping hotplug
script echo
```

If you only want `eth0` to *hotplug* and not other interfaces, use *grep* instead of *echo* as follows:

```
mapping hotplug
script grep
map eth0
```

ifplugd activates or deactivates an interface depending on whether the underlying hardware is connected to the network or not. The program can detect a cable connected to an Ethernet interface or an access point associated to a Wi-Fi interface. When *ifplugd* sees that the status of the connection has changed, it will execute a script, which, by default, executes *ifup* or *ifdown* for the interface. *ifplugd* works in combination with *hotplug*. When a card is inserted, which means that the interface is ready to be used, `/etc/hotplug.d/net/ifplugd.hotplug` starts up an instance of *ifplugd* for that interface. When *ifplugd* detects that the card is connected to a network, it executes *ifup* for this interface.

In order to associate a Wi-Fi card with an access point, we may have to program it with an appropriate WEP encryption code. If *ifplugd* is being used to control *ifup*, as we have explained, then evidently it will not be able to con-

figure the encryption code using `ifup`, as this is only called once the card has been associated. The simplest solution is to use `waproamd`, which configures the WEP encryption code according to the available access points that are discovered through a WiFi network search. For more information, consult `man waproamd` and the information on the package.

11. Virtual private network (VPN)

A VPN (*virtual private network*) is a network that uses Internet to transport data, but stops any external members from accessing that data.

This means that we have a network with connected VPN nodes tunnelled through another network, through which the traffic passes and with which no one can interact. It is used when remote users wish to access a corporate network to maintain the security and privacy of the data. Various methods can be used to configure a VPN, such as SSH (SSL), CIPE, IPSec, PPTP; they can be consulted in the bibliography (we recommend consulting VPN PPP-SSH HOWTO, by Scott Bronson and VPN-HOWTO by Matthew D. Wilson). [Bro01, Wil02].

In order to perform the configuration tests in this section, we will use OpenVPN, which is a solution based on SSL VPN and can be used for a wide range of solutions, for example, remote access, VPN point to point, secure WiFi networks or distributed corporate networks. OpenVPN implements OSI layer 2 or 3 using SSL/TLS protocols and supports authentication based on certificates, smart cards and other confirmation methods. OpenVPN is not a proxy applications server and does not operate through a web browser.

In order to analyse it, we will use an option in OpenVPN called OpenVPN for Static key configurations, which provides a simple method for configuring a VPN that is ideal for tests or point-to-point connections. The advantages are the simplicity and the fact that it is not necessary to have a X509 public key infrastructure (PKI) certificate to maintain the VPN. The disadvantages are that it only permits one client and one server, as, because the public key and private key are not used, there may be the same keys as in previous sessions and there must be a text-mode key in each peer and the secret key must be previously exchanged for a secure channel.

11.1. Simple example

In this example, we will configure a VPN tunnel on a server with IP=10.8.0.1 and a client with IP=10.8.0.2. The communication will be encrypted between the client and server on a UDP port 1194, which is the default port in OpenVPN. After installing the package (<http://openvpn.net/install.html>), we must generate the static key:

```
openvpn --genkey --secret static.key
```

Then, we must copy the static.key file in the other peer over a secure channel (using ssh or scp, for example). The server configuration file of the openVPN_server for example:

```
dev tun
ifconfig 10.8.0.1 10.8.0.2
secret static.key
```

The client configuration file for example openVPN_client

```
remote myremote.mydomain
dev tun
ifconfig 10.8.0.2 10.8.0.1
secret static.key
```

Before verifying that the VPN works, we must verify the firewall to check that port 1194 UDP is open on a server and that the virtual interface tun0 used by OpenVPN is not blocked either over the client or over the server. Bear in mind that 90% of the connection problems faced by new OpenVPN users are related in some way to the firewall.

In order to verify the OpenVPN between two machines, we must change the IPs for the real ones and the domain for the corresponding one, and then execute the server side.

```
openvpn [server config file]
```

Which will provide an output such as:

```
Sun Feb 6 20:46:38 2005 OpenVPN 2.0_rc12 i686-suse-linux [SSL]
[LZO] [EPOLL] built on Feb 5 2005
Sun Feb 6 20:46:38 2005 Diffie-Hellman initialized with 1024
bit key
Sun Feb 6 20:46:38 2005 TLS-Auth MTU parms [ L:1542 D:138
EF:38 EB:0 ET:0 EL:0 ]
Sun Feb 6 20:46:38 2005 TUN/TAP device tun1 opened
Sun Feb 6 20:46:38 2005 /sbin/ifconfig tun1 10.8.0.1 point-to-
point 10.8.0.2 mtu 1500
Sun Feb 6 20:46:38 2005 /sbin/route add -net 10.8.0.0 netmask
255.255.255.0 gw 10.8.0.2
Sun Feb 6 20:46:38 2005 Data Channel MTU parms [ L:1542
D:1450 EF:42 EB:23 ET:0 EL:0 AF:3/1 ]
Sun Feb 6 20:46:38 2005 UDPv4 link local (bound): [undef]:1194
Sun Feb 6 20:46:38 2005 UDPv4 link remote: [undef]
Sun Feb 6 20:46:38 2005 MULTI: multi_init called, r=256 v=256
Sun Feb 6 20:46:38 2005 IFCONFIG POOL: base=10.8.0.4 size=62
Sun Feb 6 20:46:38 2005 IFCONFIG POOL LIST
```


Sun Feb 6 20:46:38 2005 Initialization Sequence Completed

And the client side:

```
openvpn [client config file]
```

In order to check that it works, we might ping 10.8.0.2 from the server and ping 10.8.0.1 from the client. For more information, please check <http://openvpn.net/howto.html>.

To add compression to the link, we must add the following line to the two configuration files:

```
comp-lzo
```

In order to protect the connection through a NAT router/firewall alive and carry on the IP changes through a DNS, if one of the peers changes, add the following to the two configuration files:

```
keng-timer-rem  
persist-tun  
peepalive 10 60  
persistence-key
```

To execute as a daemon with the privileges of the nobody user/group, add the following to the configuration files:

```
user nobody  
group nobody  
Daemon
```

12. Advanced configurations and tools

There is a set of additional packages (that replace the conventional ones) and tools that either improve the machine's security (recommended in hostile environments) or help to configure the network (and the system in general) in a more user-friendly style.

These packages may be of great help to the network administrator for avoiding intrusions or avoiding local users exceeding their permissions (usually not carried out by the local user but by someone assuming their identity) or for helping new users to configure the services properly.

In this sense, we must examine:

- **Advanced TCP/IP configuration:** the `sysctl` command can be used to modify the parameters of the kernel during execution or at start up, to adjust them to the needs of the system. The parameters that may be modified are the ones in the `/proc/sys/` directory and they can be consulted with `sysctl -a`. The simplest way of modifying these parameters is through the `/etc/sysctl.conf` configuration file. After carrying out the modification, we must restart the network:

`/etc/init.d/networking restart`

In this section, we will examine some modifications for improving the network's performance (improvements depending on conditions) or the system's security (consult the references for more details) [Mou01]:

`net.ipv4.icmp_echo_ignore_all = 1`

- Does not respond to ICMP packages, such as the ping command for example, which could mean that there is a denial-of-service (DoS) attack.

`net.ipv4.icmp_echo_ignore_broadcasts = 1`

- Avoids congestion in the network not responding to the broadcast.

`net.ipv4.conf.all.accept_source_route = 0`

`net.ipv4.conf.lo.accept_source_route = 0`

`net.ipv4.conf.eth0.accept_source_route = 0`

`net.ipv4.conf.default.accept_source_route = 0`

- Inhibits the IP source routing packages, which could represent a security threat (in all the interfaces).

`net.ipv4.tcp_syncookies = 1`

`net.ipv4.conf.all.accept_redirects = 0`

- Permits the rejection of a DoS by SYNC packages, which would consume all the system's resources, forcing the user to reboot the machines.

net.ipv4.conf.lo.accept_redirects = 0

net.ipv4.conf.eth0.accept_redirects = 0

net.ipv4.conf.default.accept_redirects = 0

- Useful for avoiding ICMP redirect acceptance attacks (these packages are used when the routing does not have the appropriate route) in all the interfaces.

net.ipv4.icmp_ignore_bogus_error_responses = 1

- Sends alerts on all the error messages in the network.

net.ipv4.conf.all.rp_filter = 1

net.ipv4.conf.lo.rp_filter = 1

net.ipv4.conf.eth0.rp_filter = 1

net.ipv4.conf.default.rp_filter = 1

- Enables protection against IP spoofing in all the interfaces.

net.ipv4.conf.all.log_martians = 1

net.ipv4.conf.lo.log_martians = 1

net.ipv4.conf.eth0.log_martians = 1

net.ipv4.conf.default.log_martians = 1

Generates a log of all the spoofed packets, source routed packets and redirect packets.

- The following parameters will permit the machine to attend the TCP connections faster and better.

net.ipv4.tcp_fin_timeout = 40, By default, 60.

net.ipv4.tcp_keepalive_time = 3600, By default, 7.200.

net.ipv4.tcp_window_scaling = 0

net.ipv4.tcp_sack = 0

net.ipv4.tcp_timestamps = 0, By default, all at 1 (enabled).

- **Iptables:** the latest versions of GNU/Linux (kernel 2.4 or higher) include a new feature for building package filters called netfilter [Mou01]. This new feature is controlled by a tool called iptables that has better characteristics than its predecessor (ipchains). As we will see in the unit on security, it is extremely easy to build a firewall with this tool for detecting and warding off the most common attacks, such as DoS, IP/MAC spoofing etc. Before it is activated, we have to verify that the kernel is version 2.4 or later, which is the one that is configured to support ipfilter (which means that it is necessary to compile the kernel to activate the option *network packet filtering* [CONFIG_NETFILTER], and all the specific suboptions). The specific rules must be activated when booting (for example, through /etc/init.d and the appropriate link in the appropriate rc directory) and will have a format similar (check the references on capacities and complete syntax) to:

```
iptables -A Type -i Interface -p protocol -s SourceIP --
source-port Port -d DestinationIP --destination-port Port
-j Action
```

- **GnuPG:** this tool makes it possible to encrypt data for subsequent sending (emails, for example) or storage, it can also generate digital signatures (it meets the RFC2440 standard) and it does not use patented algorithms, which means that is open source, but we lose compatibility with other tools (for example, PGP 2.0), which use algorithms such as IDEA and RSA. For compiling and/or installing the tool, follow the instructions of the programmers at <http://www.gnupg.org/>. Firstly, we must create a pair of keys (public and private) by executing, in root, the `gpg --gen-key` command twice and answering the questions that appear. Generally, these keys will be stored in `/root`. Then we export (to a website, for example) the public key so that other users can use it to encrypt the mail/information that may only be seen by the user that generated the public key. For this, we must use `gpg --export -ao UID`, which will generate an ASCII file of the UID user's public key.
In order to import another user's public key, we can use `gpg --import filename`, and to sign a key (which is to tell the system that we are satisfied that the signed key is from who it says it is), we can use `gpg --sign-key UID`. To verify a key, we can use `gpg --verify file/data` and to encrypt/decrypt a key, `gpg -sear UID file g`, `gpg -d file`, respectively. [Gnu]
- **Logcheck:** one of a network administrator's main tasks is to check the log files daily (more than once a day) to detect any possible attacks/intrusions or events that may be evidence of these questions. This tool selects compressed information on problems and potential risks (from the log files) and then sends this information to the corresponding administrator, by email, for example. The package includes utilities for executing in independent mode and remembering the last entry verified for the subsequent executions. For information on the configuration/installation, you may consult the references. [Log]
- **PortSentry** and **Tripwire:** these tools help the network administrator to carry out their security tasks. PortSentry makes it possible to detect and respond to port searching processes (the preliminary step before attacking or spamming) in real time and to make various decisions with regard to the actions that are being performed. Tripwire is a tool that will help administrators by warning them of possible modifications and changes in the files, to avoid possible (severe) damage. This tool compares the differences between the current files and a database previously generated to detect changes (insertions and deletions), which is very useful for detecting possible modifications to vital files such as, for example, configuration files. Consult the references on the installation/configuration of these tools. [Tri]

- **Xinetd:** this tool significantly improves the efficiency and performance of inetd and tcp-wrappers. One of the biggest advantages of xinetd is that it can avoid denial-of-access (DoA) attacks through the control mechanisms for services based on the identification of client addresses, during the accessing time and (logging) time. It should not be assumed that Xinetd is the most appropriate option for all the services (for example, it is better if FTP and SSH execute only as daemons), as many of these processes will overload the system and there are secure access mechanisms that do not create interruptions in the system's security. [Xin]
Compiling and/or installing is simple; we only have to configure two files: `/etc/xinetd.conf` (the configuration file of Xinetd) and `/etc/rc.d/init.d/xinetd` (the Xinetd startup file). The first file contains two sections: *defaults*, which is where we find the parameters that will apply to all the *services*, which will be the ones that activate through Xinetd.

A typical example of the configuration might be:

```
# xinetd.conf
# The default configuration options that are applied to all the
# servers may be modified for each service
defaults
{
    instances = 10
    log_type = FILE /var/log/service.log
    log_on_success = HOST PID
    log_on_failure = HOST RECORD
}
# The name of the service must be located in /etc/services to obtain
# the right port
# If the server/Port is not a standard one, use "port = X"
service ftp
{
    socket_type = stream
    protocol = tcp
    wait = no
    user = root
    server = /usr/sbin/proftpd
}
#service telnet
#{
# socket_type = stream
# protocol = tcp
# wait = no
# user = root
# no_access = 0.0.0.0
# only_from = 127.0.0.1
# banner_fail = /etc/telnet_fail
# server = /usr/sbin/in.telnetd
#}
service ssh
{
    socket_type = stream
    protocol = tcp
    wait = no
    user = root
    port = 22
    server = /usr/sbin/sshd
    server_args = -i
}
service http
{
    socket_type = stream
    protocol = tcp
    wait = no
    user = root
```

```

server = /usr/local/apache/bin/httpd
}
#service finger
#{
# socket_type = stream
# protocol = tcp
# wait = no
# user = root
# no_access = 0.0.0.0
# only_from = 127.0.0.1
# banner_fail = /etc/finger_fail
# server = /usr/sbin/in.fingerd
# server_args = -l
#}
# End of /etc/xinetd.conf

```

The above mentioned services (#) will not be available. In the *defaults* section, we can install parameters such as maximum number of simultaneous service requests, the type of registry (log) that we require, from which nodes the requests will be received by default, the maximum number of IP requests that will be attended or the services that execute as superservers (imapd or popd), such as:

```

default {
instances = 20
log_type = SYSLOG
authpriv log_on_success = HOST
log_on_failure = HOST
only_from = 192.168.0.0/16
per_source = 3
enabled = imaps
}

```

The *service* section, one for each service, such as:

```

service imapd {
socket_type = stream
wait = no
user = root
server = /usr/sbin/imapd
only_from = 0.0.0.0/0 #allows every client
no_access = 192.168.0.1
instances = 30
log_on_success += DURATION USERID
log_on_failure += USERID
nice = 2
redirect = 192.168.1.1 993 #Makes it possible to redirect the traffic of port 993
to node 192.168.1.1
bind = 192.168.10.4
#Makes it possible to indicate the interface to which the service is associated to avoid
service spoofing problems.
}

```

The `/etc/init.d/xinetd` file makes it possible to start up the server (with the appropriate link, according to the selected runlevel, for example, 3, 4 and 5). It is convenient to change the attributes of both files to guarantee that they are not subsequently modified or disabled with: `chmod 700 /etc/init.d/xinetd; chown 0.0 /etc/init.d/xconfig; chmod 400 /etc/xinetd.conf; chattr +i /etc/xinetd.conf`.

- **Linuxconf:** this is a configuration and administration tool of a GNU/Linux system, but it is considered obsolete for most popular distributions, although it can still be found in some distributions. More information at <http://www.solucorp.qc.ca/linuxconf/>.
- **Webmin:** this is another tool (webmin-core, webmin-dhcp, webmin-inetd, webmin-sshd packages etc.) that makes it possible to configure and add aspects related to the network through a web interface (we must have installed the Apache server, for example). Although it is still being developed in many distributions, it is not included by default. For more information, please visit <http://www.webmin.com/>. To execute the tool after it has been installed from a browser, call the URL <https://localhost:10000>, which will ask you to accept the SSL certificate and the username (root user initially) and the corresponding password.
- **System-config-*:** in Fedora, there are a variety of graphic tools that are called system-config-"something" and where "something" is what they have been designed for. In general, if we are in a graphical environment, we can reach each of them using a menu; however, each of these tools means we have to remember the menu. One tool that centralises all the system configs is system-config-control in one single entry in the menu and one single graphical interface from which we can make our selections using a set of icons. For this, we have to go to Applications -> Add/Remove Software and this will start up, in root mode, in the graphical interface of the Pirut software (the Fedora Extras repository must be enabled). In the Pirut interface, the available packages can be searched for using, for example, system-config-*; make the selection for the system-config-control* and click on Apply. Among other options, we can configure almost all of the features of the network and services here.
- **Networkmanager:** it is a tool that makes it possible to manage wireless networks and cable networks easily, simply and without any complications, but it is not the most appropriate for servers (only for desktops). Installing the tool is very easy: `apt-get install network-manager-xx`, where xx is gnome or kde depending on the installed desktop. To configure the tool, we must fill in all the entries in (Debian) `/etc/network/interfaces` except for the loopback interface, for example, by only leaving:

```
auto lo
iface lo inet loopback
```

This step is not obligatory but it does make the process for discovering networks/interfaces quicker. On Debian, there is an extra step that must be taken, as the user must integrate within the netdev group, for reasons related to the permissions. To do this, we must execute (as the root user, or if not, with the sudo command first) `adduser current_user netdev` and reboot the system (or restart the network with `/etc/init.d/net-`

working restart and logging out and back in, so that the current user is included in the netdev group).

- Other tools: (some of these are explained in the unit on security) Nmap(explore and audit for network security purposes), Nessus(evaluate the network security remotely), Wireshark (<http://www.wireshark.org/download.html> (ex-Ethereal) (network protocols analyser), Snort(intrusion detection system, IDS), Netcat(simple and powerful utility for debugging and exploring a network), TCPDump(monitors networks and information acquisition), Hping2(generates and sends ICMP/UDP/TCP packages to analyse how a network works).

Activities

1) Define the following network scenarios:

- a) Isolated machine.
- b) Small local network (4 machines, 1 gateway).
- c) 2 segmented local networks (2 groups of 2 machines and one router each and a general gateway).
- d) 2 interconnected local networks (2 groups of 2 machines + a gateway each).
- e) 2 machines connected through a private virtual network. Indicate the advantages/disadvantages of each configuration, for which types of infrastructure they are appropriate and which important parameters are needed.

2) Configure the network in options a, b and d.

Annex. Controlling the services linked to an FC6 network.

An important aspect for all the services is how they are started up. FC6 includes a series of utilities for managing the service daemons (including the network ones). As we have seen on the chapter on local administration, the runlevel is the operating mode that will specify which daemons will be executed. In FC we can find: runlevel 1 (single user), runlevel 2 (multiuser), runlevel 3 (multiuser with network), runlevel 5 (X11 plus /runlevel 3). Typically, we would execute runlevel 5 or 3 if we do not need any graphical interfaces. In order to determine the level that is being executed, we can use `/sbin/runlevel`, and to know which level will start up by default `cat /etc/inittab | grep :initdefault:` which will give us information such as `id:5:initdefault:` (we can also edit `/etc/inittab` to change the default value.)

To visualise the services that are executing, we can use `/sbin/chkconfig --list` and to manage them, we can use `system-config-services` in the graphic mode or `ntsysv` in the command line. To enable individual services, we can use `chkconfig`; for example the following command enables the `crond` service for levels 3 and 5: `/sbin/chkconfig --level 35 crond on`. Regardless of how the services were started up, we can use `/sbin/service --status-all` or individually `/sbin/service crond status` to see the status of each service. And we can also manage this (start, stop, status, reload, restart), for example `service crond stop` to stop it or `service crond restart` to restart it.

It is important to not disable the following services (unless you know what you are doing): **acpid**, **haldaemon**, **messagebus**, **klogd**, **network**, **syslogd**. The most important services linked to the network (although this is not an exhaustive list and some have been left out, most of the services are listed here) are:

NetworkManager, **NetworkManagerDispatcher**: is a daemon with which we can easily change networks (Wifi and Ethernet basically). If we only have one network, it does not have to be executed.

avahi-daemon, **avahi-dnsmconfd**: is an implementation of zeroconf and it is useful for detecting devices and services on local networks without DNS (it is the same as *mDNS*).

bluetooth, **hcid**, **hidd**, **sdparm**, **dund**, **pand**: Bluetooth wireless network is for portable devices (it is not wifi, 802.11). For example, keyboards, mouse, telephones, speakers/headphones etc.

capi, isdn: network based on ISDN hardware.

Iptables: this is the standard firewall service in Linux. It is essential for security if we have a network connection (cable, DSL, T1).

Ip6tables: the same applies but for the protocol and networks based on Ipv6.

netplugd: Netplugd can monitor the network and execute commands when the status changes.

netfs: it is used to automatically mount the file systems through the network (NFS, Samba etc.) during startup.

nfs, nfslock: these are the standard daemons for sharing file systems through the network in Unix/Linux/BSD-type operating systems.

ntpd: server of time and date through the network.

portmap: is a complementary service for NFS (file sharing) and/or NIS (authentication).

rpcgssd, rpcidmapd, rpcsvcgssd: it is used for NFS v4 (new version of NFS).

sendmail: this service can be used to manage the mails (MTA) or support services such as IMAP or POP3.

smb: this daemon makes it possible to share files over Windows systems.

sshd: SSH allows other users to connect interactively and securely to the local machine.

yum-updatesd: FC network updating service.

xinetd: alternative service of inetd that presents a set of features and improvements, such as, for example, launching multiple services through the same port (this service may not be installed by default).

Server administration

Remo Suppi Boldrito

PID_00148466



Universitat Oberta
de Catalunya

www.uoc.edu

Index

Introduction.....	5
1. Domain name system (DNS).....	7
1.1. Cache names server	7
1.2. Forwarders	10
1.3. Configuration of an own domain	11
2. NIS (YP).....	14
2.1. ¿How do we initiate a local NIS client in Debian?	14
2.2. What resources must be specified in order to use NIS?	15
2.3. How should we execute a master NIS server?	16
2.4. How should we configure a server?	17
3. Remote connection services: telnet and ssh.....	19
3.1. Telnet and telnetd	19
3.2. Secure shell or SSH	20
3.2.1. ssh	20
3.2.2. sshd	21
3.2.3. Tunnel over SSH	22
4. File transfer services: FTP.....	23
4.1. FTP client (conventional)	23
4.2. FTP servers	24
5. Information exchange services at user level.....	26
5.1. The mail transport agent (MTA)	26
5.2. Internet message access protocol (IMAP)	27
5.2.1. Complementary aspects	28
5.3. News	31
5.4. World Wide Web (httpd)	32
5.4.1. Manual (minimum) configuration of httpd.conf	32
5.4.2. Apache 2.2 + SSL + PHP + MySQL	33
6. Proxy Service: Squid.....	38
6.1. Squid as an http accelerator	38
6.2. Squid as proxy-caching	39
7. OpenLdap (Ldap).....	40
7.1. Creating and maintaining the database	42
8. File services (NFS).....	44

8.1. Wiki server	45
Activities	47
Bibliography	48

Introduction

The interconnection between machines and high-speed communications has meant that the resources that are used can be at a different geographical location to that of the user. UNIX (and of course GNU/Linux) is probably the best example of this philosophy, because from its beginning, the focus has always been on the sharing of resources and the independence of the 'devices'. This philosophy has been realized in the creation of something that has now become very common: the services. A service is a resource (which may or not be universal) that makes it possible to obtain information, share data or simply process information remotely, under certain conditions. Our objective is to analyse the services that make it possible for our network. Generally, within a network, there will be a machine (or various machines, depending on the configuration) that will make it possible to exchange information with all the other elements. These machines are called servers and they contain a set of programs that centralise the information and make it easily accessible. These services help to reduce costs and increase the availability of information, but it should be remembered that a centralised service also involves some disadvantages, as it can come offline and leave the users without the service.

The servers should be designed so that all the servers are mirrored to solve these situations.

The services can be classified into two categories: those linking computers to computers or those linking users to computers. In the first case, the services are those needed by other computers, whereas in the second, the services are those required by the users (although there are services that may fall into both categories). In the first category, there are the naming services, such as the domain name system (DNS), the user information service (NISYP), the LDAP information directory or the services for storing in proxies. In the second category, we have interactive connection and remote execution services (SSH, telnet), file transfer (FTP), user-level information exchange such as email (MTA, IMAP, POP), news, World Wide Web, Wiki and files (NFS). To demonstrate all the possibilities of GNU/Linux Debian-FC6, we will describe each of these services with a minimal and operative configuration, but without leaving out the aspects related to security and stability.

1. Domain name system (DNS)

The function of the DNS service (as we explained in the unit on network administration) is to translate the machine names (legible and easy to remember for users) into IP addresses or vice-versa.

Example

When we ask the IP address of `pirulo.remix.com` is, the server will respond `192.168.0.1` (this process is known as mapping); likewise, when we request an IP address, the service will respond with the name of the machine (known as reverse mapping).

The domain name system (DNS) is a tree architecture that avoids duplicating information and makes any searches easier. For this reason, a single DNS makes no sense unless it is part of the architecture.

The application that provides this service is called *named*, it is included in most GNU/Linux distributions (`/usr/sbin/named`) and it is part of the package called *bind* (currently version 9.x) coordinated by the ISC (*Internet software consortium*). The DNS is simply a database, which means that the people that modify it have to be aware of its structure, as, otherwise, the service will be affected. As a precaution, special care must be taken to save copies of the files to avoid any interruption in the service. The package in Debian comes as *bind* and *bind.doc*. [LN01, Deb03c, IET03]. The configurations are similar, as they are FC, but you will have to install *bind*, *bind-utils* and *caching-nameserver* which will be managed by the *yum* for example.

1.1. Cache names server

Firstly, we will configure a DNS server to resolve requests, which will act as a cache for name queries (resolver, caching only server). In other words, the first time, the appropriate server will be consulted because we are starting with a database that contains no information, but all subsequent times, the cache names server will respond, with the corresponding decrease in response times. To configure the cache names server, we need the `/etc/bind/named.conf` file (in Debian), which has the following (the original comments within the file, indicated with `//`, have been respected):

```
options {
    directory "/var/cache/bind";

    // query-source address * port 53;

    // forwarders {
    //     0.0.0.0;
    // }
```

```

    };
    auth-nxdomain no; # conform to RFC1035
    };

// prime the server with knowledge of the root servers}
zone "." {
    type hint;
    file "/etc/bind/db.root"; };
    // be authoritative for the localhost forward and reverse zones, and for
    // broadcast zones as per RFC 1912
}
zone "localhost" {
    type master;
    file "/etc/bind/db.local";
    };
zone "127.in-addr.arpa" {
    type master;
    file "/etc/bind/db.127";
    };
zone "0.in-addr.arpa" {
    type master;
    file "/etc/bind/db.0";
    };
zone "255.in-addr.arpa" {
    type master;
    file "/etc/bind/db.255";
    };

// add entries for other zones below here
}

```

The directory sentence indicates where we will find the remaining configuration files (/var/cache/bind in our case). The /etc/bind/db.root file will contain something similar to the following (only the first lines, which are not comments indicated by a ';', are shown, and care must be taken with the dots [.] at the beginning of some lines –they can be obtained and updated directly from the Internet–):

```

...
; formerly NS.INTERNIC.NET
;
. 3600000 IN NS A.ROOT-SERVERS.NET.
A.ROOT-SERVERS.NET. 3600000 A 198.41.0.4
;
; formerly NS1.ISI.EDU
;
. 3600000 NS B.ROOT-SERVERS.NET.
B.ROOT-SERVERS.NET. 3600000 A 128.9.0.107
;
; formerly C.PSI.NET
;
. 3600000 NS C.ROOT-SERVERS.NET.
C.ROOT-SERVERS.NET. 3600000 A 192.33.4.12
;
...

```

This file describes the root name servers in the world. These servers change, which means that the file must be updated regularly from the Internet. The following sections are the zones; the `localhost` and `127.in-addr.arpa` zones, that link the files to the `/etc/bind/db.local` and `/etc/bind/db.127` directories, refer to the direct and inverse resolution for the local interface. The following zones are for the broadcast zones (see RFC 1912) and the appropriate zones should be added at the end. For example, the `db.local` file could be (';' means 'comment'):

```
; BIND reverse data file for local loopback interface
$TTL 604800
@      IN      SOA      ns.remix.bogus.  root.remix.bogus. (
                        1              ; Serial
                        604800         ; Refresh
                        86400          ; Retry
                        2419200        ; Expire
                        604800)        ; Negative Cache TTL
@      IN      NS       ns.remix.bogus.
1.0.0  IN      PTR      localhost.
```

We will explain how it is used later. The next step is to put the name server in `/etc/resolv.conf`:

```
search subdomain.your-domain.domain your-domain.domain
# for example search remix.bogus bogus
nameserver 127.0.0.1
```

Where we will have to replace the `subdomain.your-domain.domain` with the appropriate values. The search line indicates which domains will be searched for any host that wants to connect (it is possible to replace *search* with *domain*, although they behave differently) and the name server specifies the address of the name server (in this case, your actual machine, which is where the naming process will execute). The search behaves as follows: if a client is searching for the machine called `pirulo`, first, the `pirulo.subdomain.your-domain.domain` will be searched, then `pirulo.your-domain.domain` and finally, `pirulo`. This means that the search will take some time; however, if `pirulo` will be in `subdomain.your-domain.domain`, it is not necessary to enter the rest.

The next step is to start up *named* and look at the results of the execution. To start up the daemon, we can directly use the `/etc/init.d/bind9 start` startup script (if the *named* is already executing, go to `/etc/init.d/bind9 reload`) or, if not, `/usr/sbin/named`. If we look at the system log in `/var/log/daemon.log`, we will see something similar to:

```
Sep 1 20:42:28 remolix named[165]: starting BIND 9.2.1 \\  
Sep 1 20:42:28 remolix named[165]: using 1 CPU \\  
Sep 1 20:42:28 remolix named[167]: loading configuration from '/etc/bind/named.conf'
```

The server's startup and the error messages will appear here (if there were any errors, in which case they must be corrected and the process started again). We can now verify the configuration with commands such as `nslookup` (original and easy but obsolete according to the programmers), `host` or `dig` (recommended). The output of `dig -x 127.0.0.1` will be something like:

```
# dig -x 127.0.0.1
;; <<>> DiG 9.2.1 <<>> -x 127.0.0.1
;; global options: printcmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 31245
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONAL: 0
;; QUESTION SECTION: ;1.0.0.127.in-addr.arpa. IN PTR
;; ANSWER SECTION: 1.0.0.127.in-addr.arpa. 604800 IN PTR localhost.
;; AUTHORITY SECTION: 127.in-addr.arpa. 604800 IN NS ns.remix.bogus.
;; Query time: 1 msec
;; SERVER: 127.0.0.1 #53(127.0.0.1)
;; WHEN: Mon Sep 1 22:23:35 2003
;; MSG SIZE rcvd: 91
```

Where we can see that the query has taken 1 millisecond. If you have an Internet connection, you can search for a machine within your domain and see the server performance and behaviour. In BIND9 there is the `lwresd` (*lightweight resolver daemon*), which is the daemon that provides naming services to clients that use the BIND9 *lightweight resolver* library. It is essentially a cache server (like the one we have configured) that makes the queries using BIND9 *lightweight resolver protocol* instead of the DNS protocol. This server listens through interface 127.0.0.1 (which means it only attends to processes in the local host) in UDP and port 921. The client requests are decrypted and resolved using the DNS protocol. When responses are obtained, the `lwresd` encodes them in the lightweight format and returns them to the client that has requested them.

Finally, as we have mentioned, the kernel uses various sources of information, which, for the network, are obtained from `/etc/nsswitch.conf`. This file indicates from where we obtain the source of information and there is a section, for machine names and IPs, such as:

```
hosts: files dns
```

This line (if it is not there, it should be added) indicates that whoever needs a machine name or IP should first check `/etc/hosts` and then in DNS, in accordance with the domains indicated in `/etc/resolv.conf`.

1.2. Forwarders

In networks with a large workload, it is possible to balance the traffic using the section on forwarders. If your Internet Service Provider (ISP) has one or more stable name servers, it is advisable to use them to decongest the requests on the server. For this, we must delete the comment (`//`) from each line in the

forwarders section of the `/etc/bind/named.conf` file and replace the `0.0.0.0` with the IPs of the name servers of our ISP. This configuration is advisable when the connection is slow, when using a modem, for example.

1.3. Configuration of an own domain

DNS possesses a tree structure and the origin is known as `'.'` (see `/etc/bind/db.root`). Beneath the `'.'` there are the TLDs (*top level domains*) such as `org`, `com`, `edu`, `net` etc. When searching in a server, if the server does not know the answer, the tree will be searched recursively until it is found. Each `'.'` in an address (for example, `pirulo.remix.com`) indicates a different branch of the DNS tree and a different scope for requesting (or responsibility) that will be followed recursively from left to right.

Another important aspect, apart from the domain, is `in-addr.arpa` (*inverse mapping*), which is also nested as the domains and serves to obtain names when requesting by IP address. In this case, the addresses are written the other way round, in accordance with the domain. If `pirulo.remix.com` is `192.168.0.1`, it will be written as `1.0.168.192`, in accordance with `pirulo.remix.com`.

We must then configure the actual `remix.bogus` domain in file `/etc/bind/db.127` [LN01]:

```
; BIND reverse data file for local loopback interface
$TTL 604800
@ IN SOA ns.remix.bogus. root.remix.bogus. (
    1                ; Serial
    604800           ; Refresh
    86400            ; Retry
    2419200          ; Expire
    604800 )          ; Negative Cache TTL
@      IN      NS      ns.remix.bogus.
1.0.0 IN      PTR      localhost.
```

The `'.'` must be taken into account at the end of the domain names. The origin of a zone's hierarchy is specified by the identification of the zone, which in our case, is `127.in-addr.arpa`. This file (`db.127`) contains 3 registries: SOA, NS, PTR. The SOA (*start of authority*) must be in all of the zone files, at the beginning, after TTL and the `@` signifies the origin of the domain; NS, the name server for the domain and PTR (*domain name pointer*), which is host 1 in the subnet (`127.0.0.1`) and is called local host. This is the series 1 file and `root@remix.bogus` (last space in the SOA line) is in charge of it. We could now restart the *named* as shown above and, using `dig -x 127.0.0.1`, we could see how it works (identically to that shown previously).

We would then have to add a new zone in `named.conf`:

```
zone "remix.bogus" {
    type master;
    notify no;
    file "/etc/bind/remix.bogus";
};
```

We must remember that in `named.conf`, the domains appear without the `'.'` at the end. In the file `remix.bogus` we will put the hosts of which we will be in charge:

```
; Zone file for remix.bogus
$TTL      604800
@         IN      SOA      ns.remix.bogus.  root.remix.bogus. (
        199802151      ; serial, todays date + todays serial
        604800          ; Refresh
        86400           ; Retry
        2419200         ; Expire
        604800 )        ; Negative Cache TTL
@         NS       ns       ; Inet Address of name server
         MX        10      mail.remix.bogus. ; Primary Mail Exchanger
localhost A        127.0.0.1
ns        A        192.168.1.2
mail      A        192.168.1.4
         TXT       "Mail Server"
ftp       A        192.168.1.5
         MX        10      mail
www       CNAME     ftp
```

A new MX registry, the Mail exchanger, appears here. This is the place to which the emails that arrive will be sent, `someone@remix.bogus`, and they will be sent to `mail.remix.bogus` (the number indicates the priority if we have more than one MX). Always remember the `'.'` that is necessary in the zone files at the end of the domain (if these are not entered, the system will add the SOA domain at the end, which would transform `mail.remix.bogus`, for example, into `mail.remix.bogus.remix.bogus`, which would be incorrect). CNAME (*canonical name*) is used to give a machine one alias or various aliases. As of this moment, we would be able (after the `/etc/init.d/bind9 reload`) to test, for example, `dig www.remix.bogus`.

The last step is to configure the inverse zone, in other words, so that IP addresses can be changed into names, for example, adding a new zone:

```
zone "192.168.1.in-addr.arpa" {
    type master;
    notify no;
    file "/etc/bind/192.168.1";
};
```

And the file `/etc/bind/192.168.1` similar to the preceding one:

```
$TTL 604800
```



```
@      IN      SOA      ns.remix.bogus. root.remix.bogus. (
      199802151      ; serial, todays date + todays serial
      604800         ; Refresh
      86400          ; Retry
      2419200        ; Expire
      604800 )       ; Negative Cache TTL
@      NS              ns.remix.bogus.
2      PTR             ns.remix.bogus
4      PTR             mail.remix.bogus
5      PTR             ftp.remix.bogus
```

This can be tested again with `dig -x 192.168.1.4`. We must remember that these examples are on private IPs, in other words, not on Internet IPs. Another important point is that we must not forget the `notify no`, as otherwise, our experiments with the DNS will spread to the servers through the DNS tree (possibly even modifying the DNS of our provider or institution). These should only be modified when we are sure that it works and we are certain about the changes we want to make. To look at a real example, please see DNS-HOWTO at <http://tldp.org/HOWTO/DNS-HOWTO-7.html>.

Once we have created a master server, we must create a slave server for security, which is identical to the master, except in that the zone in the place of the type master must have a slave and the IP of the master. For example:

```
zone "remix.bogus" {
    type slave;
    notify no;
    masters {192.168.1.2; }
};
```

2. NIS (YP)

In order to facilitate the administration and make the system more user-friendly, in networks of different sizes that execute GNU/Linux (or Sun's Solaris or any other operating system that supports this service), there are Network Information Services, NIS (or yellow pages, YP, Sun's original name). GNU/Linux can provide support as an NIS client/server and can act as a client ("beta" version) of NIS+, which is a safer and more optimised version of NIS. The information that can be distributed in NIS is: users (login names), passwords (`/etc/passwd`), user directories (home directories), group information (`/etc/group`), which has the advantage that, from any client machine or from the server itself, the user may connect with the same account and password and to the same directory (although the directory must have been previously mounted on all the machines with NFS or using the automount service). [Miq01, Kuk03]

The NIS architecture is of the client-server type, in other words, there is a server that will have all the databases and some clients that will consult these data in a transparent manner for the user. For this reason, we must consider configuring the 'reinforcement' servers (called secondary servers) so that users will not be blocked because the primary server is unavailable. This architecture is called multiple server architecture (master+mirrors-clients).

2.1. ¿How do we initiate a local NIS client in Debian?

Having a local client means adding the computer to an existing NIS domain:

- First, we must verify that the *netbase* (basic TCP/IP network), *portmap* (server that turns RPC numbers into DARPA ports and that is necessary for programs that execute RPC, including NFS and NIS) and *nis* (specific) packages have been installed. We recommend using the `kpackage` command or directly with `apt-get` (we can check if this is installed with `apt-cache pkgnames`) in text mode. When installing the NIS package, you will be prompted for a domain (NIS *domainname*). This is a name that will describe the set of machines that will use NIS (it is not a host name). Bear in mind that NISpirulo is different to Nispirulo as a domain name. To configure this domain, you may use the command `nisdomainname`, a domain which will be saved in `/proc/sys/kernel/domainname`.
- Firstly, we must start up the portmap service with:
 - `/etc/init.d/portmap start`

- We can check whether these are active using `rpcinfo -p`.
- If the NIS server is not local, we must use the `ypbind` command. The `ypbind` command is used to find a server for the specified domain, whether it is through a broadcast (not recommended, as it is not secure) or through searching the server indicated in the configuration file `/etc/yp.conf` (recommended). The `/etc/yp.conf` file has the following syntax:
 - *domain nisdomain server hostname*: indicates that the *hostname* should be used for the *nisdomain*. It is possible to have more than one entry of this type for the same domain.
 - *domain nisdomain broadcast*: indicates that *broadcast* should be used on the local network to discover a server with a *nisdomain*.
 - *ypserver hostname*: indicates that *hostname* should be used as a server. It is advisable to use this line (*ypserver*) where we must enter the IP address of the NIS server. If the name is specified, make sure that the IP can be found by DNS or that it appears in the `/etc/hosts` file, as, otherwise, the client will be blocked.
- Start up the service by executing:
 - `/etc/init.d/nis stop`and then:
 - `/etc/init.d/nis start`
- This steps must start up, the NIS client will be working (this can be confirmed with `rpcinfo` or `localhost ypbind`, which will show the two versions of the active protocol) or we can use the `ypcat mapname` command (for example, `ypcat passwd`, which will show the NIS users defined in the server) where the relationship of the *mapnames* to the tables in the NIS database are defined in `/var/yp/nicknames`.

2.2. What resources must be specified in order to use NIS?

Let us assume that we have installed one of Debian's latest distributions (for example, 3.0 Woody or 3.1 Sarge), which supports Libc6 (the same for FC4 or higher) and that we want to configure the system so that the users of one client machine may access the information in the server. In this case, we must send the *login* request to the appropriate databases by:

1) Verify that the `/etc/nsswitch.conf` and ensure that the `passwd`, `group`, `shadow` and `netgroup` entries are similar to:

```
passwd: compat
group: compat
```

```
shadow: compat ...  
netgroup: nis
```

See *man nsswitch.conf* for the syntax of this file.

2) Add the following line in the NIS client machines, in the `/etc/passwd` file at the end of the file (this will indicate that if the user is not local, the NIS server will be asked):

```
+:::::: (one '+' and six ':')
```

3) It should be remembered that in `/etc/passwd` we can use the `+` and the `?` in front of each user name in `/etc/passwd`, to include/override the login of these users. If we are using passwords with shadows (more secure, as it will not allow a normal user to see the encrypted passwords of other users), the following line must be added at the end of the `/etc/shadow` file:

```
+:::::::: (one '+' and eight ':')
```

4) The following line must also be added at the end of `/etc/group`:

```
+::: (one '+' and three ':')
```

5) The searches for hosts (host lookups) will be carried out through DNS (and not NIS), which means that, for Libc6 applications, in file `/etc/nsswitch.conf` we will have to change the hosts entry for the following line: `hosts: files dns`. Or, if we prefer to do this using NIS, `hosts: files nis`. For Libc5 applications, we must modify the `/host.conf` file by entering order hosts, DNS or order hosts, NIS, as required.

With this configuration, it will be possible to establish a local connection (over the NIS client) with a user that is not defined in the `/etc/passwd` file, in other words, a user defined in another machine (ypserver).

For example, we may execute `ssh -l user localhost`, where the user is a user defined in ypserver.

2.3. How should we execute a master NIS server?

Let us assume that we have installed the `nis` package and `portmap` (and `portmap` is working) and that the data bases of the NIS have been created (see the following section):

- We must make sure that `/etc/hosts` contains all the machines that will form part of the domain in the FQDN (*fully qualified domain name*)

format, which is where the IP, the name and domain and the name without the domain of each machine appears (for example, 192.168.0.1 pirulo.remix.com pirulo). This is only necessary in the server, as the NIS does not use DNS.

- In addition, it exists in the `/etc/defaultdomain` file with the chosen domain name. Do not use your DNS domain so that you do not incur in any security risks, unless you appropriately configure the files `/etc/ypserv.securenets`, which indicates the sites from which the clients will be able to connect with a netmask/network pair, and `/etc/ypserv.conf`, which carried out a more detailed control because it indicates which hosts can access which maps, for example: `passwd.byname o shadow.byname`.
- Verify that `NISSERVER = master` exists in `/etc/default/nis`.
- For security reasons, it is possible to add the local network number to the `/etc/ypserv.securenets` file.
- Start up the server executing the `/etc/init.d/nis stop` command and then the `/etc/init.d/nis start` command. This command will start up the server (ypserv) and the password daemon (yppasswd), which may be consulted if it is active with `ypwhich -d domain`.

2.4. How should we configure a server?

The server is configured with the command `/usr/lib/yp/ypinit -m`; however, it is necessary to verify that the `/etc/networks` file exists, as it is essential for this script.

If this file does not exist, create an empty one with `touch /etc/networks`. It is also possible to make the ypbind client execute on the server; in this way, all the users entered by NIS, as mentioned above, modifying the `/etc/passwd` file, where all the normal entries before the line `+:::::::` will be ignored by the NIS (they may only access locally), whereas the subsequent ones may access through the NIS from any client. [Miq01]

Consider that as of this moment, the commands for changing the passwd or user information such as `passwd`, `chfn`, `adduser` are no longer valid. Instead, we will have to use commands such as `yppasswd`, `ypchsh` and `ypchfn`. If we change the users or modify the abovementioned files, we will have to rebuild the NIS tables by executing the `make` command in the `/var/yp` directory to update the tables.

Bear in mind that the Libc5 does not support shadow passwords, which means that shadow should not be used with NIS, if we have applications with Libc5. There will not be a problem if we have Libc6, which accepts NIS with shadow support.

Configuring a slave server is similar to configuring a master server, except in that *NISSERVER* = *slave* in */etc/default/nis*. On the master, we must indicate that it has to distribute the tables automatically to the slaves by entering *NO-PUSH* = "*false*" in the */var/yp/Makefile* file.

Now we must tell the master who its slave is, by executing:

```
/usr/lib/yp/ypinit -m
```

and entering the names of the slave servers. This will rebuild the maps but it will not send the files to the slaves. For this, on the slave, execute:

```
/etc/init.d/nis stop
```

```
/etc/init.d/nis start
```

and, finally:

```
/usr/lib/yp/ypinit -s name_master_server.
```

In this way, the slave will load the tables from the master.

It would also be possible to place the *nis* file in the */etc/cron.d* directory with a content similar to (remember to perform an *chmod 755 /etc/cron.d/nis*):

```
20 *** root /usr/lib/yp/ypxfr_1perhour >/dev/null 2>&1
40 6 *** root /usr/lib/yp/ypxfr_1perday >/dev/null 2>&1
55 6,18 *** root /usr/lib/yp/ypxfr_2perday >/dev/null 2>&1
```

With which we will ensure that all the changes in the master will be transferred to the slave NIS server.

Recommendation: After using *adduser* to add a new user on the server execute *make -C /var/yp* to update the NIS tables (and do this whenever any user characteristic changes, for example, the password with the *passwd* command, which will only change the local password and not the NIS password). To check that the system is working and that the user is registered in the NIS, you may execute *ypmatch userid passwd* where *userid* is the user previously registered with *adduser* and after performing the *make*. To verify that the NIS system is working, you may use the script of <http://tldp.org/HOWTO/NIS-HOWTO/verification.html>, which permits a more detailed verification on the NIS.

3. Remote connection services: telnet and ssh

3.1. Telnet and telnetd

Telnet is a (client) command used to communicate interactively with another host that executes the daemon telnetd. The telnet command may be executed as telnet host or interactively as telnet, which will enter the "telnet>" prompt, and then, for example: *open host*. Once communication has been established, we must enter the user and the password with which we wish to connect to the remote system. There are various commands (in the interactive mode), such as open, logout, mode (defines the visualisation characteristics), close, encrypt, quit, set, unset, or you may execute external commands with '!'. You may use the /etc/telnetrc file for default definitions or .telnetrc the definitions of a particular user (these must be in the user's home directory).

The telnetd daemon is the telnet protocol server for the interactive connection. Telnetd is generally started up by the inetd daemon and it is recommended that a tcpd wrapper (which uses the access rules in host.allow and host.deny) be included in the telnetd call within the /etc/inetd.conf file (for example), include a line such as:

```
telnet stream tcp nowait telnetd.telenetd /usr/sbin/tcpd /usr/bin/in.telnetd)
```

To increase the system's security, please see the unit on security. In some distributions (Debian 3.0 or higher), inetd's functions can be replaced by xinetd, which means that the /etc/xinetd.conf file must be configured (see the unit on the network administration). Likewise, if you wish to start up inetd in test mode, you can use the sentence `/etc/init.d/inetd.real start`. If the /etc/uissue.net file is present, telnetd will show its contents when logging in. It is also possible to use /etc/security/access.conf to enable/disable user logins, host logins or user group logins, as they connect.

It should be remembered that, although the *telnet-telnetd* pair may function in encrypt mode in the latest versions (transfer of encrypted data, although they must be compiled with the corresponding option), it is an obsolete command (deprecated), mainly due to the lack of security, although it can still be used in secure networks or in controlled situations.

If it has not been installed, we can use (Debian) `apt-get install telnetd` and then verify that it has been registered either in `/etc/inetd.conf` or in `/etc/xinetd.conf` (or in the directory in the files are defined, for example, `/etc/xinetd.d` as indicated in the previous file with the sentence `includes /etc/xinetd.d`). `xinetd.conf` or `/etc/xinetd.d/telnetd` should include a section such as (any modification in `xinetd.conf` must reboot the service with `service xinetd restart`):

```
service telnet
{
    disable = no
    flags = REUSE
    socket_type = stream
    wait = no
    user = root
    server = /usr/sbin/in.telnetd
    log_on_failure += USERID
}
```

Instead of using `telnetd`, we recommend using `SSLtelnet(d)` which replaces `telnet(d)` using encryption and authentication through SSL or using SSH (next section). `SSLtelnet(d)` may work with `telnet(d)` normally in both directions, as, when beginning communication, it verifies whether the other peer supports SSL and if not, it continues with the normal `telnet` protocol. The advantages compared to `telnet(d)` are that the passwords and data do not pass through the network in the plain text mode and anyone using, for example, `tcpdump` will be able to see the contents of the communication. Also, `SSLtelnet` may be used to connect, for example, to a secure web server (for example `https://servidor.web.org`) by simply executing: `telnet server.web.org 443`.

3.2. Secure shell or SSH

An advisable change is to use `ssh` instead of `telnet`, `rlogin` or `rsh`. These latter commands are insecure (except for `SSLtelnet`) for various reasons: the most important is that all that is transmitted through the network, including the user names and passwords, is in plain text (although there are encrypted versions of `telnet-telnetd`, they must coincide in that both of them are encrypted), anyone that has access to that network or any segment of that network will be able to obtain all that information and then assume the identity of the user. The second is that these ports (`telnet`, `rsh`,...) are the first place at which a cracker will try to connect. The `ssh` protocol (in version `OpenSSH`) provides an encrypted and compressed connection that is much more secure than, for example, `telnet` (it is advisable to use version 2 of the protocol). All current distributions incorporate the `ssh` client and the `sshd` server by default.

3.2.1. ssh

To execute the command, proceed as follows:

```
ssh -l login name host o ssh user@hostname
```


Through SSH we can encapsulate other connections such as X11 or any other TCP/IP. If we omit the parameter `-l`, the user will connect to the same local user and in both cases the server will ask for the password to authenticate the user's identity. SSH supports different authentication modes (see `ssh man` pages) based on the RSA algorithm and the public password.

It is possible to create the user identification passwords using the command `ssh-keygen -t rsa|dsa`. The command creates in the user `.ssh` directory the file `id_rsa` and `id_rsa.pub`, the private and public key respectively (for example, for RSA encryption algorithm). The user could copy the public key (`id_rsa.pub`) on the remote machine in the `.ssh` directory of the remote user, in the `authorized_keys` file. This file will be able to contain as many public keys as sites from which a remote connection to the machine will be wanted. The syntax is of one key per line and is equivalent to the `.rhosts` file (although the lines will have a considerable size). After entering the public keys of the user-machine into this file, this user will be able to connect from that machine without needing a password.

In normal mode (without creating the keys), the user will be prompted for a password, but the communication will always be encrypted and will never be accessible to other users who could be listening in on the network. For further information, see `man ssh`. In order to execute a command remotely, simply:

```
ssh -l login name host_remote_command
```

For example:

```
ssh -l user localhost ls -al
```

3.2.2. **sshd**

The `sshd` is the server (daemon) for `ssh` (if not installed, it can be installed using `apt-get install ssh` which will install the server and the client). In combination, they replace `rlogin`, `telnet`, and `rsh` and provide secure and encrypted communication between two insecure hosts in the network.

This will generally start up with the initialization files (`/etc/init.d` or `/etc/rc`) and wait for connections from clients. The `sshd` of most current distributions supports versions 1 and 2 of the SSH protocol. When the package is installed, it creates a specific RSA key of the host, and when the daemon boots, it creates another, the RSA for the session, which is not stored on disk and changes every hour. When a client initiates communication, the client generates a random number of 256 bits which is encrypted together with the two keys of the server and sent. This number will be used during the communication as the session key to encrypt the communication using a standard encryption algorithm. The user may select any of the available ones offered by the server. There are some (more secure) differences when using version 2 of the protocol. Then,

some of the user authentication methods described in the client are initiated or it will ask for the password, but always with the communication encrypted. For further information, see the `sshd` man pages.

3.2.3. Tunnel over SSH

Often we have access to an `sshd` server, but for security reasons not to other non-encrypted services (for example a POP3 mail service or X11 windows server) or simply we wish to connect to a service that can only be accessed from the company environment. To do so, it is possible to establish an encrypted tunnel between the client machine (for example with Windows, running a free software `ssh` client called `putty`) and the server with `sshd`. In this case, when we connect the tunnel to the service, the service will see the request as if it were coming from the same machine. For example, if we want to establish a POP3 connection on port 110 of the remote machine (which also has an `sshd` server) we will execute:

```
ssh -C -L 1100:localhost:110 user-id@host
```

This command will ask for the password of the `user-id` over the host and, once connected, the tunnel will have been created. Every package sent to the local machine over port 1100 will be sent to the remote machine `localhost` over port 110, which is where the POP3 service listens (option `-C` compresses the traffic through the tunnel).

Making tunnels over other ports is very easy. For example, let's suppose that we only have access to a remote proxy server from a remote machine (remote login) – not from the local machine –, we can make a tunnel to connect the navigator through the tunnel in the local machine. Let's suppose that we have a login on a gateway machine, which can access the machine called `proxy` that runs the Squid proxy server over port 3128. We run:

```
ssh -C -L 8080:proxy:3128 user@gateway
```

Once we have connected we will have the tunnel listening over local port 8080, which will redirect traffic from the gateway to the proxy to 3128. To navigate securely, all we will need to do is `http://localhost:8080/`

4. File transfer services: FTP

The file transfer protocol (FTP) is a client/server protocol (under TCP) which allows files to be transferred to and from a remote system. An FTP server is a computer that runs the `ftpd` daemon.

Some sites that allow an anonymous connection under anonymous user are generally software repositories. On a private site, we will need a username and password in order to obtain access. It is also possible to access an FTP server via a navigator and nowadays software repositories are usually replaced by web servers (e.g. Apache) or other technologies such as Bittorrent (which uses peer to peer (P2P) networks). Nonetheless, in some cases and with Debian, for example, access continues to use the username or password with the possibility of uploading files to the server (although this is also possible with web services). The file transfer protocol (FTP) (and servers/clients that implement it) by definition are not encrypted (the data, usernames and passwords are transmitted in clear text by the network) with its ensuing risk. But there are a number of servers/clients that support SSL and therefore, encryption.

4.1. FTP client (conventional)

An FTP client allows access to FTP servers and there are a large number of clients available. Using FTP is extremely simple; from the command line, run:

```
ftp server-name
```

Or also FTP, and then interactively:

```
open server-name
```

The server will prompt for a username and a password (if it accepts anonymous users, anonymous will be entered as the username and our e-mail address as the password) and from the command prompt (following several messages) we will be able to start transferring files.

The protocol allows the transfer in ASCII or binary modes. It is important to decide what type of file has to be transferred because transferring a binary in ASCII mode will destroy the file. To change between modes, we will need to execute the `ascii` or `binary` command. Useful commands of the FTP client are the `ls` (navigation in the remote directory), `get file_name` (to download files) or `mget` (which admits *), `put file_name` (to send files to the server) or `mput` (which admits *); in these last two cases we need to be authorised to write on the server's directory. We can run local commands by entering a `!`

before the command. For example `lcd /tmp` will mean that the files downloaded to the local machine will be downloaded to `/tmp`. In order to view the status and functioning of the transfer, the client will be able to print marks, or ticks, which are activated by the `hash` and `tick` commands. There are other commands that can be consulted on the page of the manual (`FTP man`) or by running `help` from within the client.

We have numerous alternatives for clients, for example in text mode: `ncftp`, `lukemftp`, `lftp`, `cftp`, `yafc`, or in graphic mode: `gFTP`, `WXftp`, `LLNL XFTP`, `guiftp`. [Bor00]

4.2. FTP servers

The traditional UNIX server is run through port 21 and is booted by the `inetd` daemon (or `xinetd` depending on which one is installed). In `inetd.conf` it is advisable to include the `tcpd` wrapper with the access rules in `host.allow` and `host.deny` in the call to `ftpd` by `inetd` to increase the system's security (refer to the chapter on security). When it receives a connection, it verifies the user and password and allows entry if authentication is correct. An anonymous FTP works differently, since the user will only be able to access an established directory in the configuration file and its subjacent tree, but not upwards, for security reasons. This directory generally contains `pub/`, `bin/`, `etc/`, and `lib/` directories so that the FTP daemon can run external commands for `ls` requests. The `ftpd` daemon supports the following files for its configuration:

- `/etc/ftpusers`: list of users that are not accepted on the system, one user per line.
- `/etc/ftpchroot`: list of users whose base chroot directory will be changed when they connect. Necessary when we want to configure an anonymous server.
- `/etc/ftpwelcome`: welcome announcement.
- `/etc/motd`: news after login.
- `/etc/nologin`: message shown after denying the connection.
- `/var/log/ftpd`: log of transfers.

If at some point we wish to inhibit the FTP connection, we can do so by including the `/etc/nologin` file. The `ftpd` will show its content and finish. If there is a `.message` file in a directory, the `ftpd` will show this when accessed.

A user's connection passes through five different levels:

- 1) Having a valid password.
- 2) Not appearing on the list `/etc/ftpusers`.
- 3) Having a valid standard shell.
- 4) If it appears in `/etc/ftpchroot`, it will be changed to the home directory (included if anonymous or FTP).
- 5) If the user is anonymous or FTP, it should have an entry in the `/etc/passwd` with FTP user, but will be able to connect by giving any password (conventionally the e-mail address is used).

It is important to bear in mind that the users that are only enabled to use the FTP service do not have a shell to the corresponding entry user in `/etc/passwd` to prevent this user having a connection through ssh or telnet, for example. Therefore, when the user is created, we will have to indicate, for example:

```
useradd -d/home/nteum -s /bin/false nteum
```

And then:

```
passwd nteum
```

Which will mean that the user `nteum` will not have a shell for an interactive connection (if the user already exists, we can edit the `/etc/passwd` file and change the last field for `/bin/false`). Then we will have to add as a last line `/bin/false` in `/etc/shells`. [Mou01] describes step by step how to create both a secure FTP server with registered users and an anonymous FTP server for non-registered users. Two of the most common non-standard servers are WUFTP-D (<http://www.wuftp.org>) and ProFTPD (<http://www.proftpd.org>). [Bor00, Mou01]

To install Proftpd on Debian, execute: `apt-get install proftpd`. After it is downloaded, `debconf` will ask if we want to run it by `inetd` or in manual mode (it is advisable to select the latter). If we wish to stop the service (for example, in order to change the configuration), we can use `/etc/init.d/proftpd stop` and to modify the file we can use `/etc/proftpd.conf`.

Consult <http://www.debian-administration.org/articles/228> in order to configure it in encrypted mode (SSL) or to have anonymous access.

A Debian server that is very interesting is PureFtpd (`pure-ftpd`) which is very secure, it allows virtual users, quotas, SSL/TSL, and a set of very interesting features. We can check its installation/configuration at <http://www.debian-administration.org/articles/383>.

5. Information exchange services at user level

5.1. The mail transport agent (MTA)

An mail transport agent (MTA) is responsible for sending/receiving mails from an e-mail server to/from Internet, implementing the simple mail transfer protocol (SMTP). By default, Debian uses *exim*, because it is easier to configure than other MTA packages, such as *smtpd* or *sendmail* (the latter is one of the precursors). *exim* offers advanced features such as rejecting known SPAM site connections, it has defences against junk mail or mail bombing and is extremely efficient at processing large amounts of mail. It is run through *inetd* on a line in the configuration file */etc/inetd.conf* with parameters for normal configurations (or *xinetd*).

exim uses a configuration file in */etc/exim/exim.conf*, which can be modified manually, but it is advisable to do so using a shell script called *eximconfig*, in order to be able to configure *exim* interactively. The configuration values will depend on the machine's situation; however, its connection is extremely easy, since the script itself suggests the default values. Nonetheless, in */usr/doc/exim* we can find examples of typical configurations.

We can test whether the configuration is valid with *exim-bV* and, if there are errors in the configuration file, the program will show them on screen or, if everything is correct, it will simply indicate the version and date. To test if it can recognise a local mailbox, use:

```
exim -v -bt local_user
```

Which will show the layers of transport used and the user's local address. We can also do the following test with a remote user by replacing local user with a remote address to see how it behaves. Then try sending a local mail message and remotely, passing the messages directly to *exim* (without using an agent, for example, *mailx*), by keying in for example (all together):

```
exim postmaster@OurDomain
From: user@domain
To: postmaster@OurDomain
Subject: Test Exim
Test message
^D
```

Next, we can analyse the *mainlog* and *paniclog* track files in */var/log/exim/* to see its behaviour and see what error messages have been generated. Obviously, we can also connect to the system as the *postmaster* user (or as the user to which the mail has been sent) and read the mail messages to see if everything

is correct. The other way consists of running it in debug mode using `-dNro` as a parameter, where `Nro` is the debug level (1-9). The normal parameter with which we should boot it is `exim -bs`, whether by `inetd` or `xinetd`. It is also possible to run it as a daemon through `/etc/init.d/exim start` in systems that require a high mail processing capacity. See the documentation (included in Debian in the `exim-doc-html` package) in order to configure filters, verification of hosts, senders etc. It is also interesting to install the `eximon` package, which is an `exim` monitor that allows the administrator to see the queue of mail messages and logs and to act on messages in the queue in order to distribute them (freezing, bouncing, thawing...).

The latest version of `exim` is `exim4` (it can be installed with `apt-get install exim4-daemon-heavy` (and also `install exim4-config` which will help to configure `exim4`) – bear in mind that there are different packages with different possibilities but `exim4-daemon-heavy` is the most complete. We recommend reading `/usr/share/doc/exim/README.Debian.gz` and `update-exim4.conf(8)`. For further information, see the `HowTo` section <http://www.exim.org/docs.html>. One of the small differences to consider in the configuration is that instead of having a single configuration `exim.conf` (the default option if we install `exim` from the sources) the package `exim4-config` (it is advisable to install it) uses small configuration files instead of a single one and that these will be in `/etc/exim4/conf.d/*` and will be chained into a single file (`/var/lib/exim4/config.autogenerated` by default) by `update-exim4.conf`.

5.2. Internet message access protocol (IMAP)

This service allows access to mail messages stored in a single server through a mail client such as `Thunderbird` or the `Seamoney` mail client (both in `mozilla.org`). This service supported by the `imapd` daemon (the current ones support the `IMAP4rev1` protocol) allows an electronic mail file that is on a remote machine. The `imapd` service is offered through the 143 (`imap2`) or 993 (when SSL encryption is supported) (`imaps`) ports. If we use `inetd`, this server is booted through a line in `/etc/inetd.conf` as:

```
imap2 stream tcp nowait root /usr/sbin/tcpd /usr/sbin/imapd
imap3 stream tcp nowait root /usr/sbin/tcpd /usr/sbin/imapd
```

In this example, the `tcpd` wrapper is called, which functions with `hosts.allow` and `hosts.deny` in order to increase security. The most popular applications are `uw-imapd` (University of Washington and installed by default in Debian) or its secure version `uw-imapd-ssl`, but also `cyrus-imap` or `courier-imap`. To test that the `imap` server functions, we could use a client, such as `seamoney-mail` and create an account for a local user and configure it appropriately so that it connects over the local machine, verifying that `imap` works correctly.

On Debian, the `imap` version has been compiled to support MD5 as the method for authenticating remote users, for encrypting connection passwords and to avoid replaced identities by sniffing on the network (the client used

to connect to the imap server must also support the MD5 authentication method). The method is very simple and secure, but the server must know the passwords in plain text of the mail users, meaning that it is advisable to use the version of imapd over SSL which functions over port 993. Like ssh, the imaps protocol is based on encrypting the communication through a host certificate (the client used for connecting to the server must also support this connection method, for example thunderbird or seamonkey -mail). To configure the imaps server, install the Debian package uw-imap-dssl which is the imap server with SSL support.

The installation will generate an auto-signed certificate valid for one year and stored in `/etc/ssl/certs/imapd.pem`. This certificate can be replaced by one signed by a certifying company or can generate its own one using OpenSSL. It is advisable to leave just the imaps entry in the file `/etc/inetd.conf` and to remove the imap2 and imap3 entries if we want the access to imap to be only by SSL.

Another protocol with similar characteristics which has been very popular in the past but that has been overtaken now by IMAP, is the post office protocol (POP) version 2 and 3. It is installed and booted in the same way as IMAP. There are numerous POP servers, but the most common ones are courier-pop, cyrus-pop3d, ipopd (University of Washington), qpopper, solid-pop3d.

5.2.1. Complementary aspects

Let's suppose that as users we have 4 email accounts on different servers and that we would like all email messages that are sent to these accounts to be gathered into a single one; to access that account externally and for it also to have an anti-spam filter.

First, we will have to install `exim + Imap` and check that they work. We need to take into account that if we install `courier-imap` (which according to some authors is better than `uw-imapd`) it functions over a mail format called Maildir, that `exim` will also have to be configured to run over maildir with the following configuration in `/etc/exim/exim.conf` (or the corresponding one if we have `exim4`), changing the option `mail_dir format = true` (the mails will be saved in the local user account in a directory called Maildir). Then we will have to reinitiate the `exim` server with `/etc/init.d/exim restart`, repeat the operational test by sending us an email message and read it with a client that supports maildir (for example `mutt -mailx` does not support it – see <http://www.mutt.org>).

To fetch the mail from the different accounts we will use `fetchmail`, (which is installed with `apt-get install fetchmail`). Next, we will have to create the `.fetchmailrc` file in our `$HOME` (we can also use the `fetchmailconf` tool) which will have to contain something like:


```
set postmaster "pirulo"
set bouncemail
set no spambounce
set flush
```

```
poll pop.domain.com proto pop3
user 'user1' there with password 'secret' is pirulo here
```

```
poll mail.domain2.com
user 'user5' there with password 'secret2' is 'pirulo' here
user 'user7' there with password 'secret3' is 'pirulo' here
```

The action set tells Fetchmail that this line contains a global option (error sending, delete mail from servers...). Next, we will specify the mail servers: one for checking if there is mail with the POP3 protocol and another for testing the use of several protocols to find one that works. We check the mail of two users with the second server option, but all mail found is sent to pirulo's mail spool. This allows us to check several mailboxes of different servers as if they were a single MUA mailbox. The specific information of each user starts with the action user. The fetchmail can be put in the cron (for example in /var/spool/cron/crontabs/pirulo adding 1 * * * * /usr/bin/fetchmail -s), so that it runs automatically or can be run in daemon mode (put set daemon 60 in .fetchmailrc and run it once for example in Autostart of Gnome/KDE or in .bashrc – it will run every 60 seconds).

To remove junk mail we will use SpamAssassin (apt-get install spamassassin) and we can configure Kmail or Evolution (check the bibliography to see how to configure it) for them to run it. In this configuration we will use Procmail, which is a very powerful tool (it allows mail distribution, filtering, automatic resending...). Once installed (apt-get install procmail), we need to create a file called .procmailrc in each user's home which will call the Spamassassin:

- Set *yes* for functioning or debugging messages
VERBOSE=no
- We suppose that the mails are in "~/Maildir", change if it is another
PATH=/usr/bin:/bin:/usr/local/bin:
MAILDIR=\$HOME/Maildir
DEFAULT=\$MAILDIR/
Directory for storing the files
PMDIR=\$HOME/.procmail
Comment if we do not want a log of Procmail
LOGFILE=\$PMDIR/log
Spam filter
INCLUDERC=\$PMDIR/spam.rc

The file ~/.procmail/spam.rc contains:

```
# If the spamassassin is not on the PATH
# add the directory to the PATH variable:
```

```
# Ofw: spamassassin.lock|
| spamassassin - a

# The three following lines will move
# Spam mail to a directory called
# "spam-folder". If we want to save it in the Inbox, so that
# it can be filtered later with the client, comment the three lines.

:O:
* ^X-Spam-Status: Yes
spam-folder
```

The file `~/spamassassin/user_prefs` contains some useful configurations for spamassassin (see the bibliography):

```
#User preferences file. Ver man
#Mail::SpamAssassin::Conf
#Threshold for recognising a Spam: #Default 5, but with 4 it works a bit better
required_hits 4
# Sites we will never consider Spam to
#come from
whitelist_from root @debian.org
whitelist_from *@uoc.edu
#Sites SPAM always comes from
#(separated by commas)
blacklist_from viagra@domain.com
#Addresses on Whitelist and blacklist are
#global patterns such
#as:"friend@place.com", "*@isp.net", or
#"*.domain.com".
#Insert the word "[SPAM]" in the subject
#(to make filtering easier).
#If we do not wish to comment the line.
subject_tag [SPAM]
```

This will generate a X-Spam-Status tag: *Yes* in the message heading if it believes that the message is Spam. Then we will have to filter these and put them in another file or to delete them directly. We can use procmail to filter mails from domains, users etc. For further information, visit <http://www.debian-administration.org/articles/242>. Finally, we can install a mail client and configure filters so that it selects all email messages with X-Spam-Status: *Yes* and deletes them or sends them to a directory where we will later verify false positives (mails identified as junk but that are not). A complementary aspect of this installation is if we wish to have a mail server through webmail (in other words, to be able to check the mails from a server through a navigator without having to install or configure a client – like consulting a gmail or hot-

mail account) it is possible to install Squirrelmail (`apt-get install squirrelmail`) in order to offer this service. For Debian visit <http://www.debian-administration.org/articles/200>.

There are other possibilities as discussed at <http://www.debian-administration.org/articles/364> installing MailDrop instead of Procmail, Postfix instead of Exim, or including Clamav/Amavisd as an antivirus (Amavisd allows postfix to be linked with spamassassin and clamav).

5.3. News

The news or discussion groups are supported through the Network News Transfer Protocol (NNTP). Installing a news server is necessary if we wish to read news offline, if we wish to have a repeater of the central servers or if we wish to have our own news master server. The most common servers are INN or CNEWS, but they are complex packages designed for large servers. Leafnode is a USENET package that implements a TNP server, especially suited for sites with small groups of users but from which we wish to access a large number of news groups. This server is installed in the basic Debian configuration and can be reconfigured with `dpkg-reconfigure leafnode` for all parameters such as central servers, type of connection etc. This daemon starts up from `inetd` in a similar way as `imap` (or with `xinetd`). Leafnode supports filters through regular indicated expressions (of the type `^Newsgroups:. * [,] alt.flame$`) in `/etc/news/leafnode/filters`, where for each message the heading is compared to the regular expression and if there is a match, the message is rejected.

This server is simple to configure and all the files must be the property of a news user with authorisation to write (check that this owner exists in `/etc/passwd`). All control, news and configuration files are found in `/var/spool/news` except for the configuration of the server itself which is in the `/etc/news/leafnode/config` file. The configuration has some obligatory parameters that must be configured (for example, so that the server can connect to the master servers). They are *server* (news server from which the news will be obtained and sent) and *expire* (number of days that a thread or session has been read and will be deleted). Likewise, we have a set of optional parameters of a general or specific nature to the server that can be configured. For further information, see the documentation (*leafnode man* or `/usr/doc/leafnode/README.Debian`).

To check the server performance, we can run:

```
telnet localhost nntp
```

and if everything works correctly, it will show the server identification and will wait for a command, as a test, we can enter *help* [to abort, Ctrl+ (and then Quit)].

5.4. World Wide Web (httpd)

Apache is one of the most popular servers with the best capabilities in terms of hypertext transfer protocol (HTTP). Apache has a modular design and supports dynamic module extensions during its execution. It is highly configurable in the number of servers and available modules and supports various mechanisms of authentication, access control, metafiles, proxy caching, virtual servers etc. With modules (included in Debian) it is possible to have PHP3, Perl, Java Servlets, SSL and other extensions (see the documentation in <http://www.apache.org>).

Apache is designed to be executed as a daemon standalone process. This way it creates a set of subsidiary processes that will handle entry requests. It can also be executed as an Internet daemon through *inetd*, meaning that it will start up every time it receives a request. The server's configuration can be extremely complex depending on the requirements (check the documentation), however, here we can see a minimum acceptable configuration. The configuration files are in `/etc/apache` and are `httpd.conf` (main configuration file), `srm.conf`, `access.conf` (these last two are maintained for compatibility), `mime.conf` (MIME formats) and `magic` (file identification number). The log files are in `/var/log/apache` and are `error.log` (registers the errors in the server requests), `access.log` (register of who has accessed what) and `apache.pid` (process identifier).

Apache boots from the start up script `/etc/init.d/apache` and `/etc/rcX.d`, but can be controlled manually through the `apachectl` command. The `apacheconfig` command can also be used in order to configure the server. The default directories (in Debian) are:

- `/var/www`: directory of HTML documents.
- `/usr/lib/cgi-bin`: directory of executables (*cgi*) by the server.
- `http://server.domain/ user`: users' personal pages.
- `/home/~user/public.html`: directory of personal pages.

The default file that is read from each directory is `index.html`. After installing the *apache* and *apache-common* packages, Debian basically configures the server and initiates it. We can check that it functions by opening a browser (for example, the Konqueror, and typing "`http://localhost`" in the URL bar, which will load the page `/var/www/index.html`).

5.4.1. Manual (minimum) configuration of `httpd.conf`

Let's look at some of the most important parameters to be configured in Apache (the example is taken from Apache version 1.X and there are some minor changes if we use version 2).

ServerType standalone	Recommended, more efficient
ServerRoot /etc/apache	Where the configuration files are found
Port 80	Where the server will listen to requests
User www-data	User and group with which the server will be executed (important for security) must be valid users (they can be <i>locked</i>)
Group www-data ServerAdmin webmaster@pirulo.remix.com	User address that will attend to errors
ServerName pirulo.remix.com	Name of the server sent to users – must be a valid name in /etc/host or DNS –
DocumentRoot /var/www	Directory where the documents will be
Alias /icons/ /usr/share/apache/icons/	Where the icons are
ScriptAlias /cgibin/ /usr/lib/cgibin/	Where the CGI scripts are

5.4.2. Apache 2.2 + SSL + PHP + MySQL

An important aspect of dynamic web servers is making the most of the advantages of Apache in secure mode (SSL), PHP (is programming language generally used to create web site content) and MySQL+PHPAdmin (database that will be discussed in later chapters and graphic interface for managing it) all working in combination. We will start by installing it on a Debian Sarge, but not through the deb packages but rather from the software downloaded from the relevant sites, this way we can repeat the experience with other distributions. Obviously, afterwards it will not be possible to control these packages using apt or another package manager. We need to take care with the versions, which can change, and not to install the package over already installed packages.

a) Download the necessary files (for example within the directory /root
-> cd /root):

1) Apache: from <http://httpd.apache.org/download.cgi>: httpd-2.2.4.tar.bz2

2) PHP: from <http://www.php.net/downloads.php> PHP 5.2.1 (tar.bz2)

3) MySQL from <http://mysql.org/get/Downloads/MySQL-4.1/mysql-standard-4.1.21-pc-linux-gnu-i686.tar.gz>/from/pick

4) PHPAdmin from <http://prdownloads.sourceforge.net/phpmyadmin/phpMyAdmin-2.9.1-all-languages.tar.bz2?download>

b) Utilities: bzip2 libssl-dev openssl gcc g++ cpp make (verify that they are not installed or otherwise, run `apt-get install bzip2 libssl-dev openssl gcc g++ cpp make`).

c) Apache:

```
cd /root
tar jxvf httpd-2.2.4.tar.bz2
cd httpd-2.2.4
```

With prefix, we indicate that we will install for example `/usr/local/apache2`

```
./configure --prefix=/usr/local/apache2 \
-with ssl=/usr/include/openssl \
--enable-ssl
make
make install
```

We modify the configuration file `/usr/local/apache2/conf/httpd.conf` and change the user and workgroup for `www-data`:

```
User www-data
Group www-data
```

We change the owner and group of the data directory to

```
www-data:chown -R www-data:www-data /usr/local/apache2/htdocs
```

We modify the user `www-data` to change its home directory in `/etc/passwd`:

```
www-data:x:33:33:www-data:/usr/local/apache2/htdocs:/bin/sh
```

Apache server installed. To initiate it (to stop it, change *start* for *stop*):

```
/usr/local/apache2/bin/apachectl start
```

We can place a script to start up the apache server upon booting.

```
In -s /usr/local/apache2/bin/apachectl /etc/rcS.d/S99apache chmod 755 /etc/rcS.d/S99apache
```

d) SSL:

In `/usr/local/apache2/conf/httpd.conf` we remove the comment from the line

```
Include conf/extra/httpd-ssl.conf
```

The files are generated with the keys for the secure server, in /root we run (adjust the versions to the ones that have been downloaded) – the first openssl command is a long line and ends with 1024:

```
openssl genrsa -rand ../httpd-2.2.4.tar.bz2:.../php-5.2.1.tar.bz2:.../phpMyAdmin-2.9.1-  
all-languages.tar.bz2 -out server.key 1024  
openssl rsa -in server.key -out server.pem  
openssl req -new -key server.key -out server.csr  
openssl x509 -req -days 720 -in server.csr -signkey server.key -out server.crt
```

We copy the files...

```
cp server.crt /usr/local/apache2/conf/  
cp server.key /usr/local/apache2/conf/
```

We restart the server...

```
/usr/local/apache2/bin/apachectl restart
```

We can check how to add the SSL module to a server that does not have it installed at <http://www.debian-administration.org/articles/349>.

e) MySQL (for more information see module 8):

We create a group and a user for MySQL if it does not exist.

```
groupadd mysql  
useradd -g mysql mysql
```

In the directory where we will install MySQL (/usr/local/) we type:

```
cd /usr/local/  
gunzip < /root/mysql-standard-4.1.21-pc-linux-gnu-  
i686.tar.gz | tar xvf - ln -s mysql-standard-4.1.21-pc-  
linux-gnu-i686 mysql cd mysql
```

We create a database and change the permissions

```
scripts/mysql_install_db --user=mysql  
chown -R root.  
chown -R mysql data  
chgrp -R mysql.
```

We can place a script for initiating the mySQL server.

```
ln -s /usr/local/mysql/support-files/mysql.server /etc/  
rcS.d/S99mysql.server  
chmod 755 /etc/rcS.d/S99mysql.server
```

We start the server

```
/etc/rcS.d/S99mysql.server start
```

We can enter the database and change the password of the root user for security (consult <http://dev.mysql.com/doc/refman/5.0/en/index.html> for the syntax)

```
/usr/local/mysql/bin/mysql
```

Inside, we can type:

```
USE mysql
```

We place the password pirulo on the user root

```
UPDATE user SET Password=PASSWORD('pirulo') WHERE User='root';  
FLUSH privileges;
```

To enter MySQL we will have to type

```
/usr/local/mysql/bin/mysql -u root -ppirulo
```

f) PHP (replace with the appropriate versions):

Necessary utilities:

```
apt-get install libxml2-dev curl \  
libcurl3-dev libjpeg-mmx-dev zlib1g-dev \  
libpng12-dev
```

With the Apache server stopped we can type:

```
cd /root  
tar jxvf php-5.2.0.tar.bz2  
cd php-5.2.0
```

With the prefix we can indicate where we want to install it (all on one line):

```
./configure --prefix=/usr/local/php5 --enable-mbstring  
--with-apxs2=/usr/local/apache2/bin/apxs --with-mysql=  
usr/local/mysql --with-curl=/usr/include/curl --with-  
jpeg-dir=/usr/include --with-zlib-dir=/usr/include --  
with-gd --with-xml --enable-ftp --enable-bcmath
```

```
make
```

```
make
```

```
install cp php.ini-dist /usr/local/php5/lib/php.ini
```

We modify Apache (/usr/local/apache2/conf/httpd.conf) in the indicated part:

```
<IfModule mime_module>  
AddType application/x-httpd-php .php .phtml  
AddType application/x-httpd-php-source .phps
```


And also:

DirectoryIndex index.php index.html

We restart the server.

g) PHPAdmin

```
cd /usr/local/apache2/
```

The phpmyadmin is decompressed in the apache2 directory (be careful with the versions).

```
tar jxvf /root/phpMyAdmin-2.9.1-all-languages.tar.bz2
mv phpMyAdmin-2.9.1-all-languages phpmyadmin
cd phpmyadmin
cp config.sample.inc.php config.inc.php
```

We need to modify the configuration file (config.inc.php):

```
$cfg['blowfish_secret'] = 'pirulo';
```

We remove the user and user password by default two quotation marks (') one after the other:

```
$cfg['Servers'][$i]['controluser'] = "";
$cfg['Servers'][$i]['controlpass'] = "";
```

We change apache (/usr/local/apache2/conf/httpd.conf) adding in <IfModule alias_module>

```
<IfModule alias_module>
    Alias /phpmyadmin "/usr/local/apache2/phpmyadmin/"
<Directory "/usr/local/apache2/phpmyadmin/">
    Order allow, deny
    Allow from all
</Directory>
```

We reinitiate the server and we can call it with <http://localhost/phpadmin>

Further information can be obtained from the respective websites of each application and in LWP.

6. Proxy Service: Squid

A Proxy server (PS) is used to save connection bandwidth, to improve security and to increase web-surfing speed.

Squid is one of the main PS, since it is OpenSource, it accepts ICP (characteristics that allow the exchange of hints with other PS), SSL (for secure connections between proxies) and supports FTP objects, Gopher, HTTP and HTTPS (secure). Its functioning is simple, it stores the most frequently requested objects in the RAM and the least requested objects in a database on the disk. Squid servers can also be configured hierarchically to form a tree of proxies according to requirements. There are two possible configurations:

- 1) As an httpd accelerator to achieve improved performance of the web service.
- 2) As a proxy-caching server to allow the users of a corporation to use the PS to exit towards the Internet.

In the first mode, it acts as an inverse proxy in other words, it accepts a client's request, serves the object if it has it, and if not, asks for it and passes it onto the client when it does, storing it for the next time. In the second option it can be used as a control to restrict the sites where a connection to the Internet can be obtained or to authorise access at specific times of day. Once installed (squid package in Debian, squid-cgi, squidguard or squidtailed can also be installed) three files are generated: /etc/squid.conf (configuration), /etc/init.d/squid (initialisation) and /etc/logrotate.d/squid (for log control).

6.1. Squid as an http accelerator

In this mode, if the web server is on the same machine as the PS, it will have to be reconfigured to attend to the requests of port 81 (in Apache, change Port 80 for Port 81 in httpd.conf). The configuration file (/etc/squid.conf) contains a large number of entries, but here we will only see the essential ones [Mou01]:

http_port 80 icp_port 0 hierarchy_stoplist cgi-bin \? acl QUERY urlpath_regexp cgi-bin \? no_cache deny QUERY	Where it listens for httpd Where it listens for ICP
cache_mem 100 MB redirect_rewrites_host_header off cache_replacement_policy lru memory_replacement_policy lru	Memory for objects in progress

cache_dir ufs /var/spool/squid 100 16 256 Database emulate_httpd_log on	Type and place where we can find the disk cache
acl all src 0.0.0.0/0.0.0.0 http_access allow all cache_mgr root cache_effective_user proxy cache_effective_group proxy httpd_accel_host 192.168.1.1 httpd_accel_port 81 logfile_rotate 0 log_icp_queries off buffered_logs on	Access for all And for everything Mail responsible UID GID Real web server Port

In this way, the option `httpd_accel_host` deactivates the possibility of it being executed as proxy-caching. For further information visit <http://www.squid-cache.org/>.

6.2. Squid as proxy-caching

This way, squid is enabled to control Internet access, when access will be given, the object that can be accessed. In this case, the configuration file will have to include the following modifications added in `/etc/squid.conf`:

```
acl localnet src 192.168.1.0/255.255.255.0
acl localhost src 127.0.0.1/255.255.255.255
acl Safe_ports port 80 443 210 70 21 102565535
acl CONNECT method CONNECT
acl all src 0.0.0.0/0.0.0.0
http_access allow localnet
http_access allow localhost
http_access deny
http_access deny CONNECT
http_access deny all
cache_emulate_httpd_log on
```

The main difference with the other mode are the `acl` lines, in which case C class clients C 192.168.1.0 will be allowed access to the PS, also the localhost IP and other ports that will be able to access the Internet 80(http), 443(https), 210(whais), 70(gopher), and 21(ftp), also, the `connect` method is denied to avoid a connection from the outside to the PS and then all IP and ports over the PS are denied. [Mou01] More information at <http://www.squid-cache.org/> and for a transparent-proxy at <http://tldp.org/HOWTO/TransparentProxy-1.html>.

7. OpenLdap (Ldap)

LDAP means lightweight directory access protocol and is a protocol for accessing data based on an X.500 service. It is run on TCP/IP and the directory is similar to that of a database that contains information based on attributes. The system allows this information to be organised in a secure manner and uses replicas to maintain its availability, ensuring its coherence and verification of accessed-modified data.

The service is based on the client-server model, where there is one or more servers that contain the data; when a client connects and asks for information, the server replies with the data or a pointer to another server where more information can be extracted, but the client will only see a directory of global information. [Mou01, Mal07]

To import and export information between ldap servers or to describe a number of changes that will be applied to the directory, we use a format called LDIF (*LDAP data interchange format*). LDIF stores the information in hierarchies oriented at objects that will then be converted to the internal format of the database. An LDIF file has a similar format to:

```
dn: o = UOC, c = SP
or: UOC
objectclass: organization
dn: cn = Pirulo Nteum, o = UOC, c = SP
cn: Pirulo Nteum
sn: Nteum
mail: nteum@uoc.edu
objectclass: person
```

Each entry is identified by a name indicated as a distinguished name (DN). The DN consists of the entry name plus a series of names that relate it to the directory's hierarchy and where there is an *objectclass*, that defines the attributes that can be used in this entry. LDAP offers a basic set of object classes: groups (including disorganised lists of individual objects or groups of objects), locations (such as countries and their description), organisations and people. An entry can also belong to more than one object class, for example, an individual is defined by the class of person, but can also be defined by attributes of the classes inetOrgPerson, groupOfNames, and organisation. The structure of the server's objects (called *schema*) determines what the permitted attributes are for an object of a class (which are defined in /etc/ldap/schema as opeldap.schema, corba.schema, nis.schema, inetorgperson.schema etc.).

All the data are represented as a pair *attribute = value* where attribute is the description of the information it contains, for example, the attribute used to store the name of a person is commonName, or cn, in other words for a person called Pirulo Nteum, it will be represented by cn: Pirulo Nteum and will have associated other attributes of the person class such as givenname: Pirulo surname: Nteum mail: pirulo@uoc.edu. The classes have obligatory and optional attributes and every attribute has an associated syntax which indicates what type of information the attribute contains, for example, bin (*binary*), ces (*case exact string*, case must match in search), cis (*case ignore string*, case can be ignored during the search), tel (*telephone number string*, ignores spaces and '-'), dn (*distinguished name*). An example of a file in LDIF format could be:

```
dn: dc = UOC, dc = com
objectclass: top
objectclass: organizationalUnit
dn: ou = groups, dc = UOC, dc = com
objectclass: top
objectclass: organizationalUnit
ou: groups
dn: ou = people, dc = UOC, dc = com
objectclass: top
objectclass: organizationalUnit
ou: people
dn: cn = Pirulo Nteum, ou = people, dc = UOC, dc = com
cn: Pirulo Nteum
sn: Nteum
objectclass: top
objectclass: person
objectclass: posixAccount
objectclass: shadowAccount
uid:pirulo
userpassword:[crypt]p1pss2ii(0pgbs*do&@ = )eksd
uidnumber:104
gidnumber:100
gecos:Pirulo Nteum
loginShell:/bin/bash
homeDirectory: /home/pirulo
shadowLastChange:10877
shadowMin: 0
shadowMax: 999999
shadowWarning: 7
shadowInactive: -1
shadowExpire: -1
shadowFlag: 0
dn: cn = unixgroup, ou = groups, dc = UOC, dc = com
objectclass: top
objectclass: posixGroup
cn: unixgroup
gidnumber: 200
memberuid: pirulo other-user
memberuid:
```

The long lines can be continued underneath starting with a space or a tab (LDIF format). In this case, the DN base has been defined for the institution dc = UOC, dc = com, which contains two sub-units: *people* and *groups*. Then it has described a user that belongs to *people* and to *group*. Having prepared the file with the data, we need to import it to the server so that it is available for LDAP clients. There are tools for converting the data of different databases to the LDIF format. [Mal07]

In Debian, we need to install the slapd package which is the OpenLdap server. During the installation, it will ask a number of questions such as: *Method of installing the directory*: auto; *extensions to the directory [domain-host,site,institution]*: host, domain, password of the Adm; *replicate local changes to other servers*: no. This installation will generate a configuration file in /etc/ldap/slapd.conf and the database on /var/lib/ldap. There is also another file /etc/ldap/ldap.conf (or ~/.ldaprc), which is the configuration file used for initialising default values when ldap clients are executed. Here it indicates which is the database, which is the ldap server, security parameters, size of the search etc.

The /etc/ldap/slapd.conf server configuration file (see `man slap.conf`) consists of different sections, each indicated by one of the following guidelines: global, backend specific and database specific, and in that order. The *global* guideline is of a general nature and applies to all the *backends* (databases) and defines general questions such as access permissions, attributes, waiting times, *schemas* etc. The *backend specific* guideline defines the attributes to the specific *backend* that it defines (bdb, dnssrv, ldbm...), and the *database specific* guideline defines the specific attributes for the database it defines. To boot the server, we need to run:

```
/etc/init.d/slapd start (or stop to stop it)
```

During the installation, the system will have created the right links for running it after start up.

7.1. Creating and maintaining the database

There are two methods for entering data in the LDAP database. The first one is easy and suitable for small amounts of data, it is interactive and we need to use tools such as ldapadd (or any other such as Ldap Browser <http://www.iit.edu/~gawojar/ldap/>) to make new entries. The second needs to be worked on offline and is suitable for large databases and uses the slapadd command included with slapd. Because it is more general, we will briefly describe the second method, where we must first verify that it contains the following attributes in slapd.conf: suffix (top of the directory, for example "o = UOC, c = SP"); directory /var/lib/ldap (directory where the indexes will be created and that can write slapd). We must also verify that the database contains the definitions of the indexes we wish to use:

```
index cn,sn,uid
index objectClass pres,eq
```

Having defined the slapd.conf, we must execute the command:

```
slapadd -l entry-f configuration [-d level] [-n whole] -b suffix]
```

The arguments are:

- l*: file in LDIF format.
- f*: server configuration file where it indicates how to create the indexes.
- d*: level of debugging.
- n*: Nro of the database, if we have more than one.
- b*: specifies what database need to be modified.

There are other commands with slapd such as slapindex, which allows the indexes to be regenerated, and slapcat, which allows dumping the database to a file in LDIF format.

8. File services (NFS)

The NFS system allows a server to export a file system so that it can be used interactively from a client. The service consists of an `nfsd` server and a client (*mountd*) which can share a file system (or part of it) through the network.

In Debian, install `apt-get install nfs-common portmap` for the client, while the server needs:

```
apt-get install nfs-kernel-server nfs-common portmap.
```

The server (in Debian) starts through the `nfscommon` and `nfs-kernel-server` scripts in `/etc/init.d` (and the appropriate links in `/etc/rcX.d`).

The server uses a file (`/etc/exports`) to manage the access and control of the file systems that will be accessed remotely. On the client, the root (or other user through *sudo*) can mount the remote system using the command:

```
mount IPserver:remote-directory local_directory
```

and as of that moment, the remote-directory will be seen within the local directory (which must exist before executing the *mount*). This task in the client can be automated using the automatic *mount* file (`/etc/fstab`) including a line; for example:

```
pirulo.remix.com:/usr/local /pub nfs rsize=8192,wzise=8192,timeo=14
```

This sentence indicates that the directory `/usr/local` of the host `pirulo.remix.com` will be mounted in the `/pub` local directory. The parameters `rsize`, `wzise` are the size of the reading and writing blocks, `timeo` is the RPC timeout (if these three values are not specified, the default values are taken).

The `/etc/exports` file serves as ACL (access control list) of the file systems that can be exported to the clients. Every line contains a file system to be exported followed by the clients that can mount it, separated by blank spaces. Each client can have a set of options associated to it in order to modify the behaviour (see the *exports man* for a detailed list of the options). An example of this could be:

```
# Example of /etc/exports
/ /master(rw) trusty(rw,no_root_squash)
/projects proj*.local.domain(rw)
/usr *.local.domain(ro) @trusted(rw)
/pub (ro,insecure,all_squash)
/home 195.12.32.2(rw,no_root_squash) www.first.com(ro)
```



```
/user 195.12.32.2/24(ro,insecure)
```

The first line exports the entire file system (/) to master and trusty in read/write mode. Plus, for trusty there is no *uid squashing* (the root of the client will access as root the root files of the server, in other words, the two root are equivalent despite being from different machines; it is suited for machines without a disk). The second and third lines show examples of '*' and *netgroups* (indicated by @). The fourth line exports the /pub directory to any machine in the world, read-only, allows access to NFS clients that do not use a port reserved for NFS (option *insecure*) and everything is executed under the user *nobody* (option *all squash*). The fifth line specifies one client for its IP and the sixth the same but with a network mask (/24) and with options between brackets () and without any spaces. There can only be spaces between the enabled clients. It is important to bear in mind that there are 3 versions of NFS (V2, V3 and recently V4). The most common ones are V3 and in some installations V2. If from a V3 client we connect to a V2 server, this situation must be indicated with a parameter.

8.1. Wiki server

A wiki (from Hawaiian *wiki wiki*, "fast") is a collaborative website that can be edited by various users who can create, edit, delete or modify the content of a web page, in an easy, fast and interactive manner; these capabilities make wiki an effective tool for collaborative writing. Wiki technology allows web pages stored in a public server (the wiki pages) to be written in a collaborative fashion through a navigator, using simple notation for giving format, creating links etc., saving a log of changes that makes it possible to recover easily any prior status of the page. When someone edits a wiki page, its changes appear immediately on the web, without passing through any type of prior revision. Wiki can also refer to pages of hypertext, which can be visited and edited by anyone (definition of Wikipedia). Debian has its wiki in <http://wiki.debian.org/> and FC in <http://fedoraproject.org/wiki/> and both are based on Moin Moin (<http://moinmoin.wikiwikiweb.de/>). MoinMoin is a Python WikiClone that can rapidly initiate its own wiki; it just needs a web server and the installed Python language.

In <http://moinmoin.wikiwikiweb.de/MoinMoinPackages/DebianLinux> we can find detailed instructions for installing Moin Moin on Debian, but, basically, it comes down to: 1) Installing apache2 and mod_python, 2) configuring Apache to note the code of MoinMoin, 3) installing the moinmoin package, 4) configuring MoinMoin and 5) restarting Apache. A configuration example:

```
apt-get install python-moinmoin
mkdir /var/www/mywiki
cp -r /usr/share/moin/data /usr/share/moin/underlay \
/usr/share/moin/server/moin.cgi /var/www/mywiki
chown -R www-data:www-data /var/www/mywiki
```

```
chmod -R g+w /var/www/mywiki
```

- Configure apache2 by adding /etc/apache2/conf.d/wiki (or wherever the configuration file is):

```
Alias /wiki/ "/usr/share/moin/htdocs/"
<Location /mywiki>
    SetHandler python-program
    PythonPath ["'/var/www/mywiki','/etc/moin/'"+sys.path"]
    PythonHandler MoinMoin.request::RequestModPy.run
    PythonDebug On
</Location>
```

- Restart apache2:

```
/etc/init.d/apache2 reload
```

- Configure MoinMoin: Edit /etc/moin/farmconfig.py (multiple wikis)

```
wikis = [
("mywiki", r"^yoursite.com/mywiki/*$"),
]
```

- we can also use (just one wiki):

```
wikis = [
("mywiki", r".*"),
]
```

- Also in /etc/moin/farmconfig.py remove the comment data_dir and data_underlay_dir (one for each wiki) and copy the file.

```
cp /etc/moin/moinmaster.py /etc/moin/mywiki.py
```

- Then edit /etc/moin/mywiki.py and change:

```
sitename = u'MyWiki'
data_dir = '/var/www/mywiki/data'
data_underlay_dir = '/var/www/mywiki/underlay'
```

The Wiki will be installed on <http://yoursite.com/mywiki/>

Activities

- 1) Configure a DNS server as cache and with its own domain.
- 2) Configure a NIS server/client with two machines exporting the server's user directories by NFS.
- 3) Configure an SSH server to access from another machine without a password.
- 4) Configure an Apache server + SSL+ PHP+ MySQL+ PHPAdmin in order to visualise users' personal pages.
- 5) Create and configure an electronic mail system through Exim, fetchmail, Spam-Assassin and an IMAP server for receiving mail from the outside and being able to read them from a remote machine with the Mozilla client (Thunderbird).
- 6) Install the MoinMoin Wiki and create a set of pages to verify that it works.

Bibliography

Other sources of reference and information

[Debc, LPD03b, Ibi]

<http://tldp.org/HOWTO/DNS-HOWTO-7.html>

<http://tldp.org/HOWTO/NIS-HOWTO/verification.html>

Squid proxy server

Proxy Cache: <http://www.squid-cache.org/>

Transparent Proxy: <http://tldp.org/HOWTO/TransparentProxy-1.html>

Proftpd: <http://www.debian-administration.org/articles/228>

PureFtpd: <http://www.debian-administration.org/articles/383>

Exim: <http://www.exim.org/docs.html>

Mutt: <http://www.mutt.org>

ProcMail: <http://www.debian-administration.org/articles/242>

LWP: http://www.lawebdelprogramador.com/temas/tema_stablephpapachemysql.php

Moin Moin: (<http://moinmoin.wikiwikiweb.de/>)

Moin Moin + Debian:

<http://moinmoin.wikiwikiweb.de/MoinMoinPackages/DebianLinux>

Apache2 + SSL: <http://www.debian-administration.org/articles/349>

Data administration

Remo Suppi Boldrito

PID_00148469



Universitat Oberta
de Catalunya

www.uoc.edu

Index

Introduction	5
1. PostgreSQL	7
1.1. How should we create a DB?	7
1.2. How can we access a DB?	8
1.3. SQL language	8
1.4. Installing PostgreSQL	10
1.4.1. Post-installation	11
1.4.2. DB users	12
1.5. Maintenance	14
1.6. Pgaccess	15
2. Mysql	17
2.1. Installation	17
2.2. Post-installation and verification	18
2.3. The MySQL monitor program (client)	19
2.4. Administration	21
2.5. Graphic interfaces	22
3. Source Code management systems	24
3.1. Revision control system (RCS)	25
3.2. Concurrent versions system (CVS)	25
3.2.1. Example of a session	29
3.2.2. Multiple users	30
3.3. Graphic interfaces	30
4. Subversion	32
Activities	37
Bibliography	38

Introduction

An important aspect of an operating system is where and how the data is saved. When the availability of data needs to be efficient, it is necessary to use databases (DB).

A database is a structured set of data that can be organised in a simple and efficient manner by the database handler. Current databases are known as relational, since the data can be stored in different tables for ease of management and administration. For this purpose and with a view to standardising database access, a language known as structured query language (SQL) is used. This language allows a flexible and rapid interaction irrespective of the database applications.

At present, the most commonly used way consists of accessing a database from an application that runs SQL code. For example, it is very common to access a DB through a web page that contains PHP or Perl code (the most common ones). When a page is requested by a client, the PHP or Perl code embedded in the page is executed, the DB is accessed and the page is generated with its static content and the content extracted from the DB that is then sent to the client. Two of the most relevant current examples of databases are those provided by PostgreSQL and MySQL, which are the ones we will analyse.

However, when we work on software development, there are other data-related aspects to consider, regarding their validity and environment (especially if there is a group of users that work on the same data). There are several packages for version control (revisions), but the purpose of all of them is to facilitate the administration of the different versions of each developed product together with the potential specialisations made for any particular client.

Versions control is provided to control the different versions of the source code. However, the same concepts apply to other spheres and not only for source code but also for documents, images etc. Although a version control system can be implemented manually, it is highly advisable to have tools that facilitate this management (cvs, Subversion, SourceSafe, Clear Case, Darcs, Plastic SCM, RCS etc.).

In this chapter, we will describe cvs (*version control system*) and Subversion for controlling and administering multiple file revisions, automating storage, reading, identifying and merging different revisions. These programs are useful when a text is revised frequently and includes source code, executables, libraries, documentation, graphs, articles and other files. [Pos03e, Mys, Ced]

The reasoning behind using cvs and Subversion is that cvs is one of the most commonly used traditional packages and Subversion is a version control system software designed specifically to replace the popular cvs and to resolve several of its deficiencies. Subversion is also known as svn since this is the name of the command line tool. An important feature of Subversion is that, unlike CVS, the files with versions do not each have an independent revision number. Instead, the entire repository has a single version number that identifies a shared status of all the repository's files at a certain point in time.

1. PostgreSQL

The PostgreSQL database (DB) language uses a client server model [Posa]. A PostgreSQL session consists of a series of programs that cooperate:

- A server process that handles the DB files accepts connections from clients and performs the actions required by the clients on the DB. The server program in PostgreSQL is called postmaster.
- The client application (frontend) is what requests the operations to be performed on the DB, which can be extremely varied; for example: tools in text mode, graphic, web servers etc.

Generally, the client and the server are on different hosts and communicate through a TCP/IP connection. The server can accept multiple requests from different clients, activating a process that will attend to the user's request exclusively and transparently for each new connection. There is a set of tasks that can be performed by the user or by the administrator, as appropriate, and that we describe as follows.

1.1. How should we create a DB?

The first action for checking whether the DB server can be accessed is to create a database. The PostgreSQL server can handle many DBs and it is recommended to use a different one for each project. To create a database, we use the `createdb` command from the operating system's command line. This command will generate a *CREATE DATABASE* message if everything is correct. It is important to take into account that for this action we will need to have a user enabled to create a database. In the section on installation (1.4) we will see that there is a user, the one that installs the database, who will have permissions for creating databases and creating new users who in turn can create databases. Generally (and in Debian) the default user is `postgres`. Therefore, before running `createdb`, we need to run `postgres` (if we are the root user, we do not need a password, but any other user will need the `postgres` password) and then we will be able to run `createdb`. To create a DB named `nteumdb`:

```
createdb nteumdb
```

If we cannot find the command, it may be that the path is not properly configured or that the DB is not properly installed. We can try with the full path (`/usr/local/pgsql/bin/createdb nteumdb`), which will depend on our specific installation, or check references for problem-solving. Other messages

would be *could not connect to server* when the server has not initiated or *CREATE DATABASE: permission denied* when we do not have authorisation to create the DB. To eliminate the DB, we can use `dropdb nteumdb`.

1.2. How can we access a DB?

After we have created the DB, we can access it in various ways:

- By running an interactive command called `psql`, which allows us to edit and execute SQL commands (e.g. `nqoj l r skb` o`).
- Executing a graphic interface such as PgAccess or a suite with ODBC support for creating and manipulating DBs.
- Writing an application using one of the supported languages, for example PHP, Perl, Java... (see PostgreSQL 7.3 Programmer's Guide).

See also

In order to access the DB, the database server must be running. When we install PostgreSQL the appropriate links are created so that the server initiates when the computer boots. For more details, consult the section on installation (1.4).

For reasons of simplicity, we will use `psql` to access the DB, meaning that we will have to enter `psql nteumdb`: some messages will appear with the version and information and a similar prompt to `nteumdb =>`. We can run some of the following SQL commands:

```
SELECT version();
```

or also

```
SELECT current date;
```

`psql` also has commands that are not SQL and that start with `\`, for example `\h` (list of all available commands) or `\q` to finish.

Example

```
Access the DB nteumdb:
psql nteumdb [enter]
nteumdb =>
```

1.3. SQL language

The purpose of this section is not to provide a tutorial on SQL, but we will analyse some examples to see this language's capabilities. They are examples that come with the PostgreSQL distribution in the `InstallationDirectory/src/tutorial` directory; in order to access them, change to the PostgreSQL directory (`cd InstallationDirectory/ src/tutorial`) and run `psql -s nteumdb` and once inside `\i basics.sql`. The parameter `\i` reads the commands of the specified file (`basic.sql` in this case).

PostgreSQL is a relational database management system (RDBMS), which means that it manages data stored in tables. Each table has a specific number of rows and columns and every column contains a specific type of data. The tables are grouped into one DB and a single server handles this collection of DB (the full set is called a database cluster).

To create, for example, a table with *psql*, run:

```
CREATE TABLE weather (
    city          varchar(80),
    min_temp      int,
    max_temp      int,
    real          rain,
    day           date
);
```

Example

Create table. Inside *psql*:

```
CREATE TABLE NameTB (var1 type, var2 type,...);
```

The command ends when we type ';' and we can use blank spaces and tabs freely. *varchar(80)* specifies a data structure that can store up to 80 characters (in this case). The point is a specific type of PostgreSQL.

To delete the table:

```
DROP TABLE table_name ;
```

We can enter data in two ways, the first is to enter all the table's data and the second is to specify the variables and values that we wish to modify:

```
INSERT INTO weather VALUES ('Barcelona', 16, 37, 0.25, '2007-03-19');
INSERT INTO weather (city, min_temp, max_temp, rain, day)
```

VALUES ('Barcelona', 16, 37, 0.25, '2007-03-19'); This method can be simple for small amounts of data, but when a large amount of data has to be entered, it can be copied from a file with the sentence:

COPY weather FROM '/home/user/time.txt'; (this file must be on the server, not on the client).

To look at a table, we could type:

```
SELECT * FROM weather;
```

where * means all the columns.

Example

A second example could be:

```
CREATE TABLE city(
name varchar(80),
place
point
);
```

Recommend Reading

We recommend studying chapter 3 of PostgreSQL on advanced characteristics (Views, Foreign Keys, Transactions, <http://www.postgresql.org/docs/8.2/static/tutorial-advanced.html> [Pos03d])

Examples

- Enter the data into the table. Inside *psql*:

```
INSERT INTO TBName (valueVar1, ValueVar2,...);
```

- Data from a file. Inside *psql*:

```
COPY TBName FROM 'FileName';
```

- Visualising data. Inside *psql*:

```
SELECT * FROM TBName;
```

Examples of more complex commands (within *psql*) would be:

- Visualises the column city after typing:

```
SELECT city, (max_temp+min_temp)/2 AS average_temp, date FROM weather;
```

- Visualises everything where the logical operation is fulfilled:

```
SELECT * FROM weather WHERE city = 'Barcelona'
AND rain > 0.0;
```

- Joining tables:

```
SELECT * FROM weather, city WHERE city = name;
```

- Functions, in this case maximum:

```
SELECT max(min_temp) FROM weather;
```

- Nested functions:

```
SELECT city FROM weather WHERE min_temp = (SELECT max(min_temp) FROM weath-
er);
```

- Selective modification:

```
UPDATE weather SET max_temp = max_temp 2, min_temp = min_temp 2 WHERE day
> '19990128';
```

- Deleting the register:

```
DELETE FROM weather WHERE city = 'Sabadell';
```

1.4. Installing PostgreSQL

This step is necessary for DB administrators [Posa]. The DB administrator's functions include software installation, initialisation and configuration, administration of users, DBs and DB maintenance tasks.

The database can be installed in two ways: through the distribution's binaries, which is not difficult, since the distribution scripts carry out all the necessary steps for making the DB operative, or through the source code, which will have to be compiled and installed. In the first case, we can use the *kpackage* (Debian) or the *apt-get*. In the second case, we recommend always going to the source (or to a mirror repository of the original distribution). It is important to bear in mind that the installation from the source code will then be left outside the DB of installed software and that the benefits of software administration offered, for example, by *apt-cache* or *apt-get* will be lost.

Installation from source code step by step:

- First we need to obtain the software from the site (x.x is the available version) <http://www.postgresql.org/download/> and decompress it (x.x.x is version number 8.2.3 at the time of this revision):

```
gunzip postgresql-x.x.x.tar.gz
tar xf postgresql-7.3.tar
```
- Change to the postgresql directory and configure it with `./configure`.
- Compile it with `gmake`, verify the compilation with `gmake check` and install it with `gmake install` (by default, it will install it in `/usr/local/pgsql`).

1.4.1. Post-installation

Initialise the variables, in *bash*, *sh*, *ksh*:

```
LD_LIBRARY_PATH = /usr/local/pgsql/lib;
PATH = /usr/local/pgsql/bin:$PATH;
export LD_LIBRARY_PATH PATH;
```

or, in *csh*:

```
setenv LD_LIBRARY_PATH /usr/local/pgsql/lib;
set path = (/usr/local/pgsql/bin $path)
```

We recommend locating this initialisation in the user configuration scripts, for example `/etc/profile` or `.bashrc` for *bash*. To have access to the manuals, we need to initialise the `MANPATH` variable in the same way:

```
MANPATH = /usr/local/pgsql/man:$MANPATH;
export MANPATH
```

Once the DB is installed, we will need to create a user that will handle the databases (it is advisable to create a different user from the root user so that there is no connection with other services of the machine), for example, the postgres user using the *useradd*, command for example.

Next, we will need to create a storage area for the databases (single space) on the disk, which will be a directory, for example `/usr/local/pgsql/data`. For this purpose, execute `initdb -D /usr/local/pgsql/data`, connected as the user created in the preceding point. We may receive a message that the directory cannot be created due to no privileges, meaning that we will first have to create the directory and then tell the DB which it is; as root, we have to type, for example:

```
mkdir /usr/local/pgsql/data
```

```
chown postgres /usr/local/pgsql/data
su postgres
initdb -D /usr/local/pgsql/data
```

Initiate the server (which is called *postmaster*), to do so, use:

```
postmaster -D /usr/local/pgsql/data
```

to run it in active mode (in the foreground); and to run it in passive mode (in the background) use:

```
postmaster -D /usr/local/pgsql/data < logfile 2 >&1 &.
```

Reroutings are done in order to store the server's errors. The package also includes a script (*pg_ctl*) so as not to have to know all the *postmaster* syntax in order to run it:

```
/usr/local/pgsql/bin/pg_ctl start -l logfile \
-D /usr/local/pgsql/data
```

We can abort the server's execution in different ways, with *pg-ctl*, for example, or directly using:

```
kill -INT 'head -1 /usr/local/pgsql/data/postmaster.pid'
```

1.4.2. DB users

DB users are completely different to the users of the operating system. In some cases, it could be interesting for them to maintain correspondence, but it is not necessary. The users are for all the DBs that the server controls, not for each DB. To create a user, execute the SQL sentence:

```
CREATE USER name
```

To delete users:

```
DROP USER name
```

We can also call on the *createuser* and *dropuser* programs from the command line. There is a default user called *postgres* (within the DB), which is what will allow us to create the rest (to create new users from *psql -U postgres* if the user of the operating system used for administrating the DB is not *postgres*).

A DB user can have a set of attributes according to what the user is allowed to do:

Note

Creating, deleting users:
createuser [options] name
dropuser [options] name

- Superuser: this user has no restrictions. For example, it can create new users; to do this, run:
`CREATE USER name CREATEUSER`
- DB creator: is authorised to create a DB. To create a user with these characteristics, use the command:
`CREATE USER name CREATEDB`
- Password: only necessary if we wish to control users' access when they connect to a DB for security reasons. To create a user with a password, we can use:
`CREATE USER name PASSWORD 'password'`
where password will be the password for that user.
- We can change a user's attributes by using the command `ALTER USER`. We can also make user groups that share the same privileges with:
`CREATE GROUP GroupName`
And to insert the users in this group:
`ALTER GROUP GroupName ADD USER Name1`
Or to delete it :
`ALTER GROUP GroupName DROP USER Name1`

Example

Group operations inside psql:

```
CREATE GROUP GroupName;  
ALTER GROUP GroupName ADD USER Name1...; ALTER GROUP GroupName  
DROP USER Name1...;
```

When we create a DB, the privileges are for the user that creates it (and for the superuser). To allow another user to use this DB or part of it, we need to grant it privileges. There are different types of privileges such as `SELECT`, `INSERT`, `UPDATE`, `DELETE`, `RULE`, `REFERENCES`, `TRIGGER`, `CREATE`, `TEMPORARY`, `EXECUTE`, `USAGE`, and `ALL PRIVILEGES` (consult the references for their meaning). To assign privileges, we can use:

```
GRANT UPDATE ON object TO user
```

where user must be a valid PostgreSQL user and object, a table, for example. This command must be executed by the superuser or table owner. The `PUBLIC` user can be used as a synonym for all users and `ALL`, as a synonym for all privileges. For example, to remove all of the privileges from all of the users of an object, we can execute:

```
REVOKE ALL ON object FROM PUBLIC;
```

1.5. Maintenance

There is a set of tasks that the DB administrator is responsible for and that must be performed periodically:

1) Recovering the space: periodically we must execute the VACUUM command, which will recover the disk space of deleted or modified rows, update the statistics used by the PostgreSQL scheduler and improve access conditions.

2) Reindexing: In some cases, PostgreSQL can give problems with the reuse of indexes, therefore it is advisable to use REINDEX periodically to eliminate pages and rows. We can also use contrib/reindexdb in order to reindex an entire DB (we need to take into account that, depending on the size of the DBs, these commands can take a while).

3) Change of log files: we need to prevent the log files from becoming too large and difficult to handle. This can be done easily when the server is initiated with:

```
pg_ctl start | logrotate
```

logrotate renames and opens a new log file and it can be configured with /etc/logrotate.conf.

4) Backup copy and recovery: there are two ways of saving data, with the sentence SQL Dump or by saving the DB file. The first will be:

```
pg_dump DBFile> BackupFile
```

For recovery, we can use: `psql DBFile< BackupFile`

In order to save all of the server's DBs, we can execute:

```
pg_dumpall > TotalBackupFile
```

Another strategy is to save the database files at the level of the operating system, for example using:

```
tar -cf backup.tar /usr/local/pgsql/data
```

There are two restrictions that can make this method unpractical:

- The server has to be stopped before saving and recovering the data.
- We need to know very well all of the implications at the level of the file where all the tables are, transactions etc., since otherwise, we could render a DB useless. Also (in general), the size that will be saved will be greater

than if done using the previous methods, since for example, with the `pg_dump` the indexes are not saved, but rather the command in order to recreate them is saved.

Summary of the installation of PostgreSQL:

```
./configure
gmake
su
gmake install
adduser postgres
mkdir /usr/local/pgsql/data
chown postgres /usr/local/pgsql/data
su - postgres
/usr/local/pgsql/bin/initdb -D /usr/local/pgsql/data
/usr/local/pgsql/bin/postgres -D /usr/local/pgsql/data <
logfile 2>&1 &
/usr/local/pgsql/bin/createdb test
/usr/local/pgsql/bin/psql test
```

1.6. Pgaccess

The application `pgaccess [DBName]` (<http://www.pgaccess.org/>) allows us to access and administer a database with a graphic interface. The easiest way of accessing (from KDE for example) is from a terminal, the DB administrator will have to do, (if not the *postgres* user) `xhost+` which will allow other applications to connect to the current user's display

```
su postgres
pgaccess [DBName] &
```

If configured in 'Preferences' it will always open the last DB. Figure 15 shows the `pgaccess` interface.

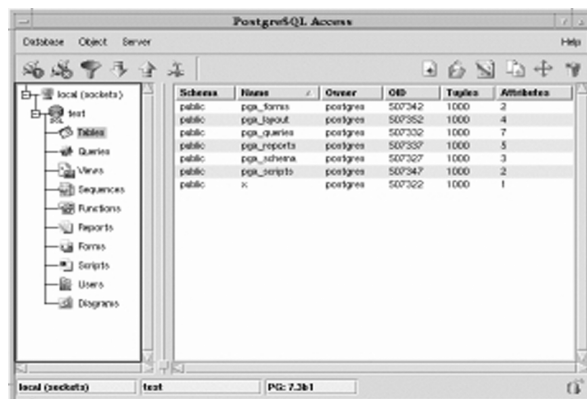


Figure 1. PgAccess

In a typical session the administrator /user could, firstly, Open DataBase, indicating here for example, Port = 5432, DataBase = nteum (the other parameters are not necessary if the database is local) and then Open. As of this moment, the user will be able to work with the bidimensional space selecting what it wants to do in the Y axis (tables, consultations, views etc.) and with that el-

ement highlighted, and selecting one of that type within the window, using the *X* axis above for New (add), Open or Design. For example, if we select in *Y* Users and in *X*, New, the application will ask for the username, password (with verification), timeout and characteristics (for example, Create DB, Create other users). In DataBase we could also select Preferences, so as to change the type of font, for example, and select the possibility of seeing the system's tables.

Users' personal configurations will be registered in the file `~/.pgaccessrc`. The interface helps to perform/facilitate a large amount of the user/administrator's work and it is recommended for users who have just started in PostgreSQL, since they will not need to know the syntax of the command line as in `psql` (the application itself will request all of a command's options through several windows).

A simpler tool is through the corresponding webmin module (we need to install the packages `webmin-core` and required modules, for example, in this case `webmin-postgresql`), but in many distributions it is not included by default (for more information visit <http://www.webmin.com/>). During the installation, webmin will warn that the main user will be the root and will use the same password as the root of the operating system. To connect, we can do so from a navigator for example, `https://localhost:10000`, which will ask to accept (or deny) the use of the SSL certificate for the SSL communication, and next it will show all of the services that can be administered, among them the PostgreSQL Data Base Server.

2. Mysql

MySQL [Mys] is (according to its authors) the most popular open SQL (DB), in other words free software (Open Source), and is developed and distributed by MySQL AB (a commercial enterprise that makes profit from the services it offers over the DB). MySQL is a database management system (DBMS). A DBMS is what can add and process the data stored inside the DB. Like PostgreSQL, MySQL is a relational database, which means that it stores data in tables instead of in a single location, which offers greater speed and flexibility. As it is free software, anyone can obtain the code, study it and modify it according to their requirements, without having to pay anything, since MySQL uses the GPL license. On its webpage, MySQL offers a set of statistics and features compared to other DBs to show users how fast, reliable and easy it is to use. The choice of a DB should be made carefully according to users' needs and the environment in which the DB will be used.

2.1. Installation

- Obtain it from <http://www.mysql.com/> or any of the software repositories. The binaries and source files can be obtained for compilation and installation.
- In the case of the binaries, use the Debian distribution, and select the packages `mysql-*` (client, server, common are required). Following a number of questions, the installation will create a `mysql` user and an entry in `/etc/init.d/mysql` in order to start/stop the server during boot. It can also be done manually using:
`/etc/init.d/mysql start|stop`
- In order to access the database, we can use the `mysql` monitor from the command line. If we obtain the binaries (not Debian or RPM, with this simply use the common `-apt-get, rpm-`), for example `gz` from the MySQL website, we will have to execute the following commands in order to install the DB:

```
groupadd mysql
useradd -g mysql mysql
cd /usr/local
gunzip < /path/to/mysql-VERSION-OS.tar.gz | tar xvf -
ln -s full-path-to-mysql-VERSION-OS mysql
cd mysql
scripts/mysql_install_db --user=mysql
chown -R root .
chown -R mysql data
chgrp -R mysql .
```

```
bin/mysqld_safe --user=mysql &
```

This creates the user/group/directory, decompresses and installs the DB in /usr/local/mysql.

- In the case of obtaining the source code, the steps are similar:

```
groupadd mysql
useradd -g mysql mysql
gunzip < mysql-VERSION.tar.gz | tar -xvf -
cd mysql-VERSION
./configure --prefix=/usr/local/mysql
make
make install
cp support-files/my-medium.cnf /etc/my.cnf
cd /usr/local/mysql
bin/mysql_install_db --user=mysql
chown -R root .
chown -R mysql var
chgrp -R mysql .
bin/mysqld_safe --user=mysql &
```

It is important to pay attention when configuring, since `prefix= /usr/local/mysql` is the directory where the DB will be installed and it can be changed to locate the DB in any directory we wish.

2.2. Post-installation and verification

Once installed (whether from the binaries or the source code), we will have to verify if the server works properly. In Debian this can be done directly:

<code>/etc/init.d/mysql start</code>	starts the server
<code>mysqladmin version</code>	Generates version information
<code>mysqladmin variables</code>	Shows the values of the variables
<code>mysqladmin -u root shutdown</code>	Shuts down the server
<code>mysqlshow</code>	Will show the predefined DBs
<code>mysqlshow mysql</code>	Will show the tables of the mySQL DB

If installed from the source code, before making these checks we will have to execute the following commands in order to create the databases (from the distribution's directory):

```
./scripts/mysql_install_db
cd InstallationDirectoryMysql
```

```
./bin/mysqld_safe --user = mysql &
```

If we install from the binaries (RPM, Pkg,...), we must do the following:

```
cd InstallationDirectoryMysql
./scripts/mysql_install_db
./bin/mysqld_safe user = mysql &
```

The script `mysql_install_db` creates the `mysql` DB and `mysqld_safe` starts up the `mysqld` server. Next, we can check all of the commands given above for Debian, except the first one which is the one that starts up the server. Plus, if the tests have been installed, these can be run using `cd sql-bench` and then `run-all-tests`. The results will appear in the directory `sql-bench/results` for comparison with other DBs.

2.3. The MySQL monitor program (client)

The MySQL client can be used to create and use simple DBs, it is interactive and can connect to the server, run searches and visualise results. It also works in batch mode (as a script) where the commands are passed onto it through a file). To see all the command options, we can run `mysql --help`. We will be able to make a connection (local or remote) using the `mysql` command, for example, for a connection via the web interface but from the same machine:

```
mysql -h localhost -u mysql -p DBName
```

If we do not enter the last parameter, no DB is selected.

Once inside, `mysql` will show a prompt (`mysql>`) and wait for us to insert a command (own and SQL), for example `help`. Next, we will give a series of commands in order to test the server (remember always to type the `'` to end the command):

```
mysql> SELECT VERSION(), CURRENT_DATE;
```

We can use capital letters or small caps.

```
mysql> SELECT SIN(PI()/4), (4+1)*5; Calculator.
mysql> SELECT VERSION(); SELECT NOW();
```

Multiple commands on the same line.

```
mysql> SELECT
-> USER()
-> ,
-> CURRENT_DATE;
```

Or on multiple lines.

```
mysql> SHOW DATABASES;
```

Note

MySQL client (frontend):
`mysql [DBName]`

Web site

For further information, see the documentation, commands and options. [Mys07]
<http://dev.mysql.com/doc/refman/5.0/es/>

Shows the available DBs.

```
mysql> USE test
```

Changes the DB.

```
mysql> CREATE DATABASE nteum; USE nteum;
```

Creates and selects a DB called nteum.

```
mysql> CREATE TABLE pet (name VARCHAR(20), owner VARCHAR(20),  
-> species VARCHAR(20), sex CHAR(1), birth DATE, death DATE);
```

Creates a table inside nteum.

```
mysql> SHOW TABLES;
```

Shows the tables.

```
mysql> DESCRIBE pet;
```

Shows the table's definition.

```
mysql> LOAD DATA LOCAL INFILE "pet.txt" INTO TABLE pet;
```

Loads data from pet.txt in pet. The pet.txt file must have one register per line separated by data tabs according to the table's definition (date in YYYY-MM-DD format).

```
mysql> INSERT INTO pet  
-> VALUES ('Marciano','Estela','gato','f','1999-03-30',NULL);
```

Loads data inline.

```
mysql> SELECT * FROM pet;Shows table data.
```

```
mysql> UPDATE pet SET birth = "1989-08-31" WHERE name = "Browser";
```

Modifies table data.

```
mysql> SELECT * FROM pet WHERE name = "Browser";
```

Selective sample.

```
mysql> SELECT name, birth FROM pet ORDER BY birth;
```

Ordered sample.


```
mysql> SELECT name, birth FROM pet WHERE MONTH(birth) = 5;
```

Selective sample with functions.

```
mysql> GRANT ALL PRIVILEGES ON *.* TO  
martian@localhost -> IDENTIFIED BY 'passwd'  
WITH GRANT OPTION;
```

Create user marciano in the DB. This has to be executed by the DB root user.
Or it can also be done directly by using.

```
mysql> INSERT INTO user (Host,User,Password) ->  
  
VALUES('localhost','marciano','passwd');
```

2.4. Administration

Mysql has a configuration file in /etc/mysql/my.cnf (in Debian), where the DB default options can be changed, for example, the connection port, user, password of remote users, log files, data files, whether it accepts external connections etc. In relation to security, we need to take certain precautions:

- 1) Not to give anyone (except the root user of MySQL) access to the user table within the MySQL DB, since this is where the user passwords are, which could be used for other purposes.
- 2) Verify `mysql -u root`. If we can access, it means that the root user does not have a password. To change this, we can type:

```
mysql -u root mysql  
mysql> UPDATE user SET Password =  
PASSWORD('new_password')  
-> WHERE user = 'root';  
mysql> FLUSH PRIVILEGES;
```

Now, to connect as root:

```
mysql -u root -p mysql
```

- 3) Check the documentation concerning the security conditions and the network environment to avoid problems with attacks and/or intrusions.

- 4) To make copies of the database, we can use the following command:

```
mysqldump --tab = /DestinationDirectory \  
--opt DBName
```

or also:

```
mysqlhotcopy DBName /DestinationDirectory
```

Likewise, we can copy the files *.frm', *.MYD', and *.MYI with the server stopped. To recover, execute:

```
REPAIR TABLE o myisamchk -r
```

which will work in 99% of cases. Otherwise, we could copy the saved files and start up the server. There are other alternative methods depending on what we want to recover, such as the possibility of saving/recovering part of the DB (see point 4.4 of the documentation). [Mys]

2.5. Graphic interfaces

There are a large number of graphic interfaces for MySQL, among which we should mention MySQL Administrator (it can be obtained from <http://www.mysql.com/products/tools/administrator/>). Also as tools we can have Mysql-Navigator (<http://sourceforge.net/projects/mysqlnavigator/>), or Webmin with the module for working with MySQL (packages webmin-core and webmin-mysql) although the latter is no longer included with some distributions. Similarly to PostgreSQL, Webmin also permits working with MySQL (we will need to install the webmin-mysql packages as well as webmin-core). During the installation, webmin will warn that the main user will be the root and will use the same password as the root of the operating system. To connect, we can type, for example, <https://localhost:10000> on the URL bar of a navigator which will request acceptance (or denial) of the use of a certificate for the SSL communication and next it will show all the services that can be administered, among them the MySQL Data Base Server.

MySQL Administrator is a powerful application for administering and controlling databases based on MySQL. This application integrates DB management, control and maintenance in a simple fashion and in the same environment. Its main characteristics are: advanced administration of large DBs, fewer errors through "visual administration", greater productivity and a safe management environment. The following figure shows a view of MySQL Administrator (in <http://dev.mysql.com/doc/administrator/en/index.html> we can find all of the documentation for installing it and starting it up).

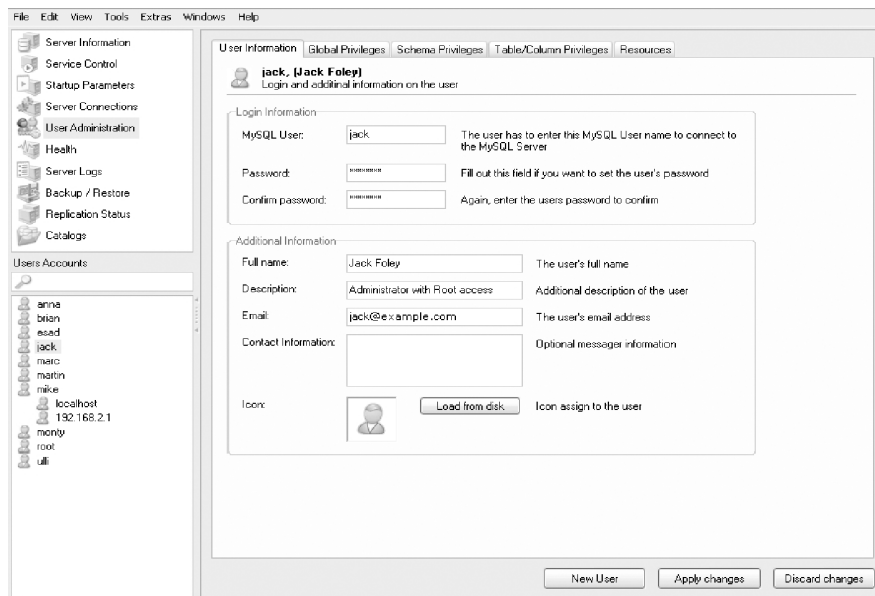


Figure 2. MySQL Administrator

3. Source Code management systems

The concurrent versions system (CVS) is a version control system that allows old version of files to be maintained (generally source code), saving a log of who made any changes, when and why. Unlike other systems, CVS does not work with a file/directory per occasion, but rather acts on hierarchical groups of the directories it controls.

The purpose of CVS is to help to manage software versions and to control the concurrent editing of source files by multiple authors. CVS uses another package called RCS (*revision control system*) internally as a low level layer. Although RCS can be used independently, it is not recommended, because in addition to its own functionality CVS offers all the capabilities of RCS but with notable improvements in terms of stability, functioning and maintenance. Among which we would highlight: decentralised operation (every user can have their own code tree), concurrent editing, adaptable behaviour through shell scripts etc. [Ced, CVS, Vasa, Kie]

As already explained in the introduction, Subversion (<http://subversion.tigris.org/>) is a version control system software specifically designed to replace the popular CVS, and to extend its capabilities. It is free software under an Apache/BSD type license and is also known as svn for the name on the command line. An important feature of Subversion is that unlike CVS the versioned files do not each have an independent revision number and instead the entire repository has a single version number which identifies a common status of all the repository's files at the time that it was "versioned". Among the main features we would mention:

- File and directory history can be followed through backups and renamings.
- Atomic and secure modifications (including changes to several files).
- Efficient and simple creation of branches and labels.
- Only the differences are sent in both directions (in CVS, complete files are always sent to the server).
- It can be served by Apache, over WebDAV/DeltaV.
- It handles binary files efficiently (unlike CVS, which treats them internally as if they were text).

There is an interesting free book that explains everything related to Subversion <http://svnbook.red-bean.com/index.es.html> and the translation is fairly advanced (<http://svnbook.red-bean.com/nightly/es/index.html>).

3.1. Revision control system (RCS)

Given that CVS is based on RCS and is still used on some systems, we will offer a few brief explanations of this software. RCS consists of a set of programs for its different RCS activities: `rcs` (program that controls file attributes under RCS), `ci` and `co` (which verify the entry and exit of files under RCS control), `ident` (searches RCS for files using key words/attributes), `rcsclean` (cleans files that are not used or that have not changed), `rcsdiff` (runs the `diff` command to compare versions), `rcsmerge` (joins two branches [files] into a single file), and `rlog` (prints log messages).

The format of the files stored by RCS can be text or another format, like binary for example. An RCS file consists of an initial revision file called 1.1 and a series of files of changes, one for each revision. Every time a copy of the repository is made to the work directory with the `co` (which obtains a revision of every RCS file and puts it in the work file) or `ci` (which stores new revisions in the RCS) commands is used, the version number is increased (for example, 1.2, 1.3,...). The files are (generally) in the `/RCS` directory and the operating system needs to have the `diff` and `diff3` commands installed in order for it to function properly. In Debian, it does not have to be compiled since it is included in the distribution.

With the `rcs` command we will create and modify file attributes (consult `rcs man`). The easiest way to create a repository is to create a directory with `mkdir rcs` in the directory of originals and include the originals in the repository using: `ci name_files_sources`.

We can use the `*` and should always have a backup copy to avoid problems. This will create the versions of the files with the name `./RCS/file_name` and request a descriptive text for the file. Then, using `co RCS/file_name`, we will obtain a work copy from the repository. This file can be blocked or unblocked to prevent modifications, respectively, using:

```
rcs -L workfile_name
rcs -U workfile_name
```

With `rlog file_name` we will be able to see the information on the different versions. [Kie]

3.2. Concurrent versions system (CVS)

First we need to install the concurrent versions system (CVS) from the distribution bearing in mind that we must have RCS installed and that we should also install OpenSSH if we wish to use it in conjunction with CVS for remote access. The environment variables `EDITOR` `CVSROOT` must also be initiated for example in `/etc/profile` (or in `.bash profile`):

```
export EDITOR = /bin/vi
export CVSROOT = /usr/local/cvsroot
```

Obviously, users can modify these definitions using `/.bash` profile. We need to create the directory for the repository and to configure the permissions; as root, we have to type, for example:

```
export CVSROOT = /usr/local/cvsroot
groupadd cvs
useradd -g cvs -d $CVSROOT cvs
mkdir $CVSROOT
chgrp -R cvs $CVSROOT
chmod o-rwx $CVSROOT
chmod ug+rwx $CVSROOT
```

To start up the repository and save the code file in it:

```
cvs -d /usr/local/cvsroot init
```

`cvs init` will take into account never overwriting an already created repository to avoid the loss of other repositories. Next, we will need to add the users that will work with CVS to the `cvs` group; for example, to add a user called *nteum*:

```
usermod -G cvs,nteum
```

Now user *nteum* will have to save his or her files in the repository directory (`/usr/local/cvsroot` in our case) by typing:

```
export EDITOR = /bin/vi
export CVSROOT = /usr/local/cvsroot
export CVSREAD = yes
cd directory_of_originals
cvs import RepositoryName vendor_1_0 rev_1_0
```

The name of the repository can be a single identifier or also `user/project/xxxx` if the user wishes to have all their repositories organised. This will create a tree of directories in `CVSROOT` with that structure.

This adds a directory (`/usr/local/cvsroot/RepositoryName`) in the repository with the files that will be in the repository as of that moment. A test to know whether everything has been stored correctly is to save a copy in the repository and then create a copy from there and check the difference. For example, with the originals in `user_directory/dir_org` if we want to create a repository `first_cvs/proj`, we will have to execute the following commands:

```
cd dir_org    Change to the original source code directory.
```

```
cvs import -m \
"Original sources"
```

```
\primer_cvs/proj userX vers0
```

Creates the repository in first_cvs/proj with userX and vers0.

```
cd..
```

Change to the superior directory dir_org.

```
cvs checkout primer_cvs/proj
```

Generating a copy of the repository. The variable CVSROOT must be initiated, otherwise the full path will have to be shown.

```
diff -r dir_org primer_cvs/proj
```

Shows the differences between one and the other; there should not be any except for the directory first_cvs/proj/CVS created by CVS.

```
rm -r dir_org
```

Deletes the originals (always make a backup copy for safety and to have a reference of where the work with CVS started).

The following figure shows the organisation and how the files are distributed between versions and branches.

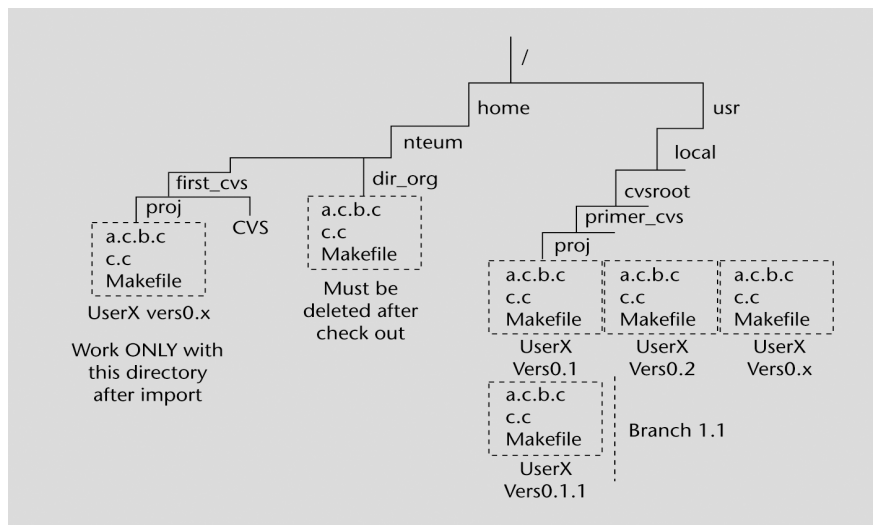


Figure 3

Deleting the originals is not always a good idea; only in this case, after verifying that they are in the repository, so that they will not be worked on by mistake and the changes will not be reflected on the CVS. On machines where users will want to access a remote CVS server (by ssh), we must type:

```
export CVSROOT = ":ext:user@CVS.server.com:/home/cvsroot"
```

```
export CVS_RSH = "ssh"
```

Where user is the user login and cvs.server.com the name of the server with CVS. CVS offers a series of commands (named as cvs command options...) to work with the revision system, including: checkout, update, add, remove, commit and diff.

The initial cvs checkout command creates its own private copy of the source code so as to later work with it without interfering with the work of other users (at minimum it creates a subdirectory where the files will be located).

- `cvs update` must be executed from the private tree when copies of source files have to be updated with the changes made by other programmers to the repository's files.
- `cvs add file...` this command is necessary when we need to add new files to the work directory on a module that has already previously run a checkout. These files will be sent to the CVS repository when we execute the `cvs commit` command.
- `cvs import` can be used for introducing new files to the repository.
- `cvs remove file...` this command will be used to delete files from the repository (once these have been deleted from the private file). This command has to be accompanied by a `cvs commit` command for the changes to become effective, since this is the command that converts all of the users requests over the repository.
- `cvs diff file...` it can be used without affecting any of the files involved if we need to verify differences between repository and work directory or between two versions.
- `cvs tag -R "version"` can be used for introducing a version number in project files and then typing `cvs commit` and a `cvs checkout -r 'version' project` in order to register a new version.

An interesting characteristic of cvs is that it is able to isolate the changes to files isolated on a separate line of work called a branch. When we change a file on a branch, these changes do not appear on the main files or on other branches. Later, these changes can be incorporated to other branches or to the main file (merging). To create a new branch, use `cvs tag -b rel-1-0-patches` within the work directory, which will assign the name rel-1-0-patches to the branch. To join the branches to the work directory involves using the `cvs update -j` command. Consult references for merging or accessing different branches.

3.2.1. Example of a session

Following the example of the documentation provided in the references, we will show a work session (in a general form) with cvs. Since cvs saves all the files in a centralised repository, we will assume that it has already been initiated.

Let's suppose that we are working with a set of files in C and a *makefile*, for example. The compiler we use is gcc and the repository is initialised to gccrep.

In the first place, we will need to obtain a copy of the repository files as our own private copy with:

```
cvs checkout gccrep
```

This will create a new directory called gccrep with the source files. If we execute `cd gccrep` and `ls`, we will see, for example, cvs makefile a.c b.c c.c, where there is a cvs directory that is created to control the private copy that normally we do not need to touch.

We could use an editor to modify a.c and introduce substantial changes in the file (see the documentation on multiple concurrent users if we need to work with more than one user on the same file), compile, change again etc.

When we decide that we have a new version with all the changes made in a.c (or in the necessary files), it is time to make a new version by saving a.c (or all those that have been touched) in the repository and making this version available to the rest of the users: `cvs commit a.c`.

Using the editor defined in the variable CVSEEDITOR (or EDITOR if it is not initialised) we will be able to enter a comment that discusses what changes have been made to help other users or to remind what characterised this version so that a log can be made.

If we decide to eliminate the files (because the project was completed or because it will not be worked on any more), one way of doing this is at the level of the operating system (`rm -r gccrep`), but it is better to use the cvs itself outside of the work directory (level immediately above): `cvs release -d gccrep`. The command will detect whether there is any file that has not been sent to the repository, and if there is and it is erased, it means that all the changes will be lost, which is why it will ask us if we wish to continue or not.

To look at the differences for example, b.c has been changed and we do not remember what changes were made, within the work directory, we can use: `cvs diff b.c`. This will use the operating system's diff command to compare

version b.c with the version in the repository (we must always remember to type `cvs commit b.c` if we want these differences to be transferred to the repository as a new version).

3.2.2. Multiple users

When more than one person works on a software project with different revisions, it is extremely complicated, because more than one user will sometimes want to edit the same file simultaneously. A potential solution is to block the file or to use verification points (reserved checkouts), which will only allow one user to edit the same file simultaneously. To do so, we must execute the command `cvs admin -l command` (see man for the options).

cvs uses a default model of unreserved checkouts, which allows users to edit a file in their work directory simultaneously. The first one to transfer their changes to the repository will be able to do so without any problems, but the rest will receive an error message when they wish to perform the same task, meaning that they must use cvs commands in order to transfer firstly the changes to the work directory from the repository and then update the repository with their own changes.

Consult the references to see an example of its application and other ways of working concurrently with communication between users. [Vasa].

3.3. Graphic interfaces

We have a set of graphic interfaces available such as tkcvs (<http://www.twobarleycorns.net/tkcvs.html>) [gcus] developed in Tcl/Tk and that supports subversion, or an also very popular one, cervisia [Cerc].

In the cvs wiki (http://ximbiot.com/cvs/wiki/index.php?title=CVS_Clients) we can also find a set of clients, plugins for cvs. Next, we will look at two of the mentioned graphic interfaces (tkcvs and gcvs):

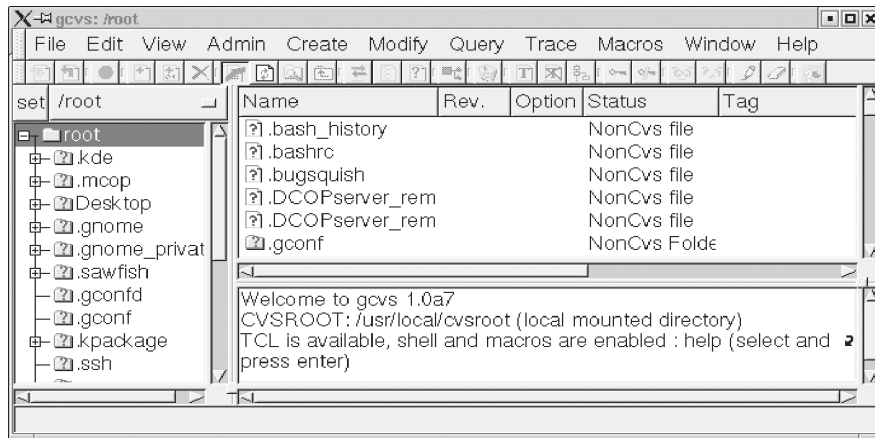


Figure 4. TkCVS (TkSVN)

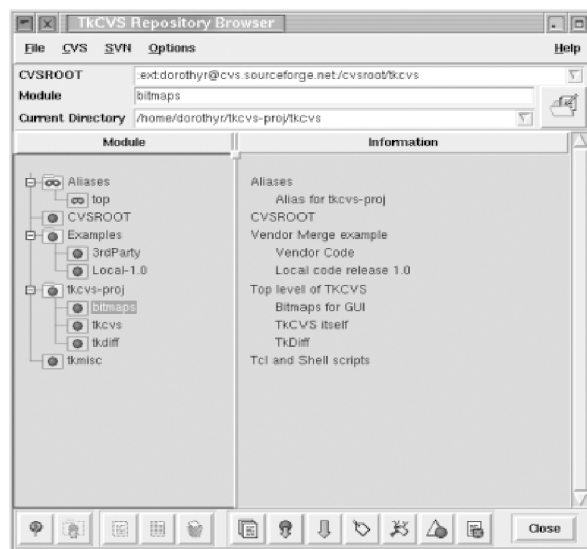


Figure 5. gCVS.

4. Subversion

As an initial idea subversion serves to manage a set of files (repository) and its different versions. It is relevant to note that we do not care how the files are saved, but rather how we can access them, for which it is common to use a database. The idea of a repository is like a directory from which we want to recover a file of one week or 10 months ago based on the database status, to recover the latest versions and add new ones. Unlike cvs, subversion makes global revisions of the repository, which means that a change in a file does not generate a leap in version of that file only, but also of the entire repository, which adds one to the revision. In addition to the book we have mentioned (<http://svnbook.red-bean.com/nightly/es/index.html>), consult the documentation at <http://subversion.tigris.org/servlets/ProjectDocumentList>. Figure 5. gCVS

In Debian, we will have to type `apt-get install subversion`, if we wish to publish the repositories in Apache2 `apt-get install Apache2-common` and the specific module `apt-get install libApache2-subversion`.

- First step: create our repository, user (we assume the user is svuser), group (svgroup) as *root*...

```
mkdir -p /usr/local/svn  
addgroup svgroup  
chown -R root.svgroup /usr/local/svn  
chmod 2775 /usr/local/svn
```
- `addgroup svuser svgroup` Adds the svuser user to the svgroup group.

- We connect as svuser and verify that we are in the svgroup group (with the group command).
- `svnadmin create /usr/local/svn/tests`
This command will create a series of files and directories for version management and control. If we are not permitted in /usr/local/svn, we can do so in the local directory: `mkdir -p $HOME/svndir` and next `svnadmin create $HOME/svndir/tests`.
- Next we create a temporary directory `mkdir -p $HOME/svntmp/tests` we move to the directory `cd $HOME/svntmp/tests` and create a file like: `echo First File Svn 'date' > file1.txt`.
- We transfer it to the repository: inside the directory we type `svn import file:///home/svuser/svndir/tests-m "View. Initial"`. If we have created it in /usr/local/svn/tests we should type the full path after `file:///`. The import command copies the directory tree and the -m option allows the version message to be shown. If we do not add the -m option, an editor will open to do so (we need to enter a message in order to avoid problems). The subdirectory \$HOME/svntmp/tests is a copy of the work in the repository and deleting it is recommended so as not to be tempted to commit the error of working with it and not with the repository (`rm -rf $HOME/svntmp/tests`).
- Once in the repository, we can obtain the local copy where we can work and then upload the copies to the repositories, by typing:
`mkdir $HOME/svn-work`
`cd $HOME/svn-work`
`svn checkout file:///home/svuser/svndir/tests`
Where we will see that we have the tests directory. We can copy with another name adding the name we want at the end. To add a new file to it:
`cd /home/kikov/svn-work/tests`
`echo Second File Svn 'date' > file2.txt`
`svn add file2.txt`
`svn commit -m"New file"`
It is important to note that once in the local copy (svn-work) we must not specify the path. `svn add` marks to add the file to the repository and that really it is added when we run `svn commit`. It will give us some messages indicating that it is the second version.

If we add another line file1.txt with `echo 'date'>>file1.txt`, then we will be able to upload the changes with: `svn commit -m"New line"`.

It is possible to compare the local file with the repository file, for example we add a third line to file1.txt with `echo 'date'>>file1.txt`, but we do not upload it and if we want to see the differences we can run: `svn diff`.

This command will highlight what the differences are between the local file and those of the repository. If we load it with `svn commit -m"New line2"` (which will generate another version) then the `svn diff` will not give us any differences.

We can also use the command `svn update` within the directory to update the local copy. If there are two or more users working at the same time and each one has made a local copy of the repository and modifies it (by doing commit), when the second user goes to commit their copy with their modifications, they will receive a conflict error, since the copy in the repository has a more recent modification date than this user's original copy (in other words, there have been changes in between), meaning that if the second user runs commit, we could lose the modifications of the first one. To do this, we must run `svn update` which will tell us the file that creates a conflict with a C and will show us the files where the parts in conflict have been placed. The user must decide what version to keep and whether they can run commit.

An interesting command is the `svn log file1.txt`, which will show all the changes that have been made to the file and its corresponding versions.

An interesting feature is that subversion can run in conjunction with Apache2 (and also over SSL) to be accessed from another machine (consult the clients in <http://svnbook.red-bean.com/>) or simply look at the repository. In Debian Administration they explain how to configure Apache2 and SSL for Sarge, or as we already indicated in the part on servers. For this, we need to activate the WebDAV modules (see <http://www.debian-administration.org/articles/285> or in their absence <http://www.debian-administration.org/articles/208>).

As root user we type:

```
mkdir /subversión chmod www-data:www-data
```

So that Apache can access the directory

```
svnadmin create /subversion
```

we create the repository

```
ls -s /subversion
```

```
-rw-r--r-- 1 www-data www-data 376 May 11 20:27 README.txt
drwxr-xr-x 2 www-data www-data 4096 May 11 20:27 conf
drwxr-xr-x 2 www-data www-data 4096 May 11 20:27 dav
drwxr-xr-x 2 www-data www-data 4096 May 11 20:28 db
-rw-r--r-- 1 www-data www-data 2 May 11 20:27 format
drwxr-xr-x 2 www-data www-data 4096 May 11 20:27 hooks
drwxr-xr-x 2 www-data www-data 4096 May 11 20:27 locks
```

For authentication we use `htpasswd` (for example with

htpasswd2 -c -m /subversion/.dav_svn.passwd user created as www-data. We only have to type the -c the first time that we execute the command to create the file. This tells us that in order to access this directory we need a password (which is the one we have entered for user).

Then we will need to change the httpd.conf so that it is something like:

```
<location /svn>
    DAV svn
    SVNPath /subversion
    AuthType Basic
    AuthName "Subversion Repository"
    AuthUserFile /subversion/.dav_svn.passwd
    Require valid-user
</location>
```

We reinitiate Apache and now we are ready to import some files, such as:

```
svn import file1.txt http://url-server.org/svn \
-m "Import Initial"
```

We will be asked for authentication (user/password) and told that the file file1.txt has been added to the repository.

Activities

1) Define in PostgreSQL a DB that has at least 3 tables with 5 columns (of which 3 must be numerical) in each table.

Generate an ordered list for each table/column. Generate a list ordered by the highest value of the *X* column of all tables. Change the numerical value in the *Y* column with the numerical value of column *Z* + the value of column *W*/2.

2) The same exercise as above, but with MySQL.

3) Configure the cvs to make three revisions of a directory where there are 4 .c files and a makefile. Make a branch of the file and then merge it with the main one.

4) Simulate the concurrent use of a file with two Linux terminals and indicate the sequence of steps to be done so that the two alternating modifications of each user are reflected in the cvs repository.

5) Same exercise as above, but one of the users must connect to the repository from another machine.

6) Idem 3, 4 and 5 in Subversion.

Bibliography

Other sources of reference and information

[Debc, Ibi, Mou01]

PgAccess: <http://www.pgaccess.org/>

WebMin: <http://www.webmin.com/>

Mysql Administrator <http://www.mysql.com/products/tools/administrator/>

Graphic interfaces for CVS: <http://www.twobarleycorns.net/tkcv.html>

Or in the CVS wiki: http://ximbiot.com/cvs/wiki/index.php?title=CVS_Clients

Subversion: <http://subversion.tigris.org>

Free Book about Subversion: <http://svnbook.red-bean.com/index.es.html>

Apache2 and SSL: <http://www.debian-administration.org/articles/349>

Apache2 and WebDav: <http://www.debian-administration.org/articles/285>

There is a large amount of documentation about Apache and SSL + Subversion in Debian as well as <http://www.debian-administration.org>, in Google, enter "Apache2 SSL and Subversion in Debian" to obtain some interesting documents.

Security administration

Josep Jorba Esteve

PID_00148474



Universitat Oberta
de Catalunya

www.uoc.edu

Index

Introduction.....	5
1. Types and methods of attack.....	7
1.1. Techniques used in the attacks	10
1.2. Countermeasures	16
2. System security.....	20
3. Local security.....	21
3.1. Bootloaders	21
3.2. Passwords and shadows	22
3.3. Suid and sticky bits	23
3.4. Enabling hosts	24
3.5. PAM modules	24
3.6. System alterations	26
4. SELinux.....	28
4.1. Architecture	31
4.2. Criticism	34
5. Network security.....	36
5.1. Service client	36
5.2. Server: inetd and xinetd	36
6. Intrusion detection.....	39
7. Filter protection through wrappers and firewalls.....	40
7.1. Firewalls	41
7.2. Netfilter: IPtables	42
7.3. Packets of firewalls in the distributions	45
7.4. Final considerations	46
8. Security tools.....	48
9. Logs analysis.....	51
10. Tutorial: Tools for security analysis	53
Activities.....	59
Bibliography.....	60

Introduction

The technological leap from isolated desktop systems to current systems integrated into local networks and Internet has added a new difficulty to the administrator's usual tasks: controlling system security.

Security is a complex field, which combines analysis techniques with techniques for detecting or preventing potential attacks. Such as the analysis of "psychological" factors, in relation to the behaviour of system users or attackers' possible intentions.

The attacks can come from many sources and affect from a single application or service or user to all of them or even the entire computer system.

Potential attacks can change the systems' behaviour and even make them crash (disabling them), or give a false impression of security, which can be difficult to detect. We can come across authentication attacks (obtaining access through previously disabled programs or users), interceptions (redirecting or intercepting communication channels and the data circulating within them) or substitution (replacing programs, machines or users for others, without the changes being noticed).

Note

Absolute security does not exist. A false impression of security can be as damaging as not having any security. Security is a very dynamic field on which we need to keep our knowledge constantly updated.

We must bear in mind that it is impossible to achieve 100% security.

Security techniques are a double-edged sword that can easily give us a false impression of controlling the problem. Currently, security is an extensive and complex problem and, more importantly, it is also dynamic. We can never expect or say that security is guaranteed, but rather it will probably be one of the areas that the administrator will have to spend most time on and on which knowledge will have to be kept updated.

In this unit we will examine some of the types of attacks we can encounter, how we can verify and prevent parts of local security and network environments from being attacked. Furthermore, we will examine techniques for detecting intrusions and some basic tools that can help us to control security.

We should also mention that in this unit we can only introduce some of the aspects related to security nowadays. For any real thorough learning, we advise consulting the available bibliography, as well as the manuals for the products and tools we have covered.

1. Types and methods of attack

Computer security in administration terms can be understood as the process that allows the system's administrator to prevent and detect unauthorised use of the system. Preventive measures help to prevent attempts by unauthorised users (known as intruders) to access any part of the system. Detection helps to discover when these attempts were made or, if they are effective, to establish barriers so that intrusions are not repeated and so that the system can be recovered if breached.

Intruders (known also colloquially as hackers, crackers, 'attackers' or 'pirates') normally wish to obtain control over the system, whether to cause its malfunctioning, to corrupt the system or its data, to make use of the machine's resources or simply to use it to launch attacks on other systems, thus helping them to protect their own identity and hide the real source of the attacks. It is also possible that they wish to examine (or steal) the system's information, straightforward espionage of the system's actions or to cause physical damage to the machine, by formatting the disk, changing data, deleting or modifying critical software etc.

With regard to intruders, we need to establish some differences that are not very clear in colloquial terms. Normally, we refer to a hacker [Him01], as a person with detailed knowledge of computing, more or less passionate about programming and security issues and that normally, for no malevolent purpose uses their knowledge to protect themselves or third parties by entering networks to detect security failures and, in some cases, to test their abilities.

An example would be the GNU/Linux community, which owes a lot to its hackers, since the term hacker has to be understood as an expert in certain issues (rather than an intruder on security).

At the same time, we have crackers. This is where the term is used more or less negatively, towards those who use their knowledge in order to corrupt (or destroy) systems, whether for their own fame, for financial reasons, with the intention of causing damage or simply inconvenience; for reasons of technological espionage, acts of cyber-terrorism etc. Likewise, we talk of hacking or cracking, when we refer to techniques for studying, detecting and protecting security, or, on the contrary, techniques designed to cause damage by breaching systems' security.

Unfortunately, obtaining access to a system (whether it is unprotected or partially safe) is much easier than it would seem. Intruders constantly discover new vulnerabilities (sometimes known as 'holes' or exploits), that allow them to enter different layers of software. The ever-increasing complexity of soft-

ware (and hardware) makes it more and more difficult to test the security of computer systems in a reasonable manner. The common use of GNU/Linux on networks, whether via the Internet or private networks with TCP/IP technology such as intranets, makes us expose our systems, as victims, to security attacks. [Bur02][Fen02][Line]

The first thing we have to do is to break the myth of computer security: it simply does not exist. What we can achieve is a certain level of security that makes us feel safe within certain parameters. But as such, it is merely a perception of security and, like all perceptions, can be false so that we may only become aware at the last minute once our systems have already been affected. The logical conclusion is that computer security requires an important effort in terms of consistency, realism and learning on a practically daily basis.

We need to be capable of establishing security policies for our systems that allow us to prevent, identify and react against potential attacks. And to be aware that the feeling of security that we may have, is precisely no more than that: a feeling. Therefore, we must not neglect any implemented policies and we need to keep them up to date, as well as our knowledge of the issue.

Possible attacks are a constant threat to our systems and can compromise their functioning, as well as the data that we handle; We will always have to define a certain policy of security requirements for our systems and data. The threats we may suffer could affect the following aspects:

Note

Threats affect confidentiality, or the integrity or accessibility of our systems.

- **Confidentiality:** the information must only be accessible to authorised persons; we are answering the question: who will be able to access it?
- **Integrity:** the information must only be modified by authorised persons: what can be done with it?
- **Accessibility:** the information must be available for those who need it when they need it, on condition that they are authorised: how and when can it be accessed?

Let's move on to a certain (non-exhaustive) classification of the usual types of attacks that we can suffer:

- **Authentication:** attacks that falsify the identity of the participant so that access is obtained to programs or services that were initially out of bounds.
- **Interception** (or tapping): mechanism whereby data is intercepted by third parties to whom the data was not directed.
- **Falsification** (or replacement): replacement of some participants – whether machines, software or data – by other false ones.

- **Theft** of resources: unauthorised use of our resources.
- Or, simply, **vandalism**: after all, the presence of mechanisms that allow interference with the correct functioning of the system or services to cause partial inconvenience or the shutdown or cancellation of resources is fairly common.

The methods and precise techniques employed can vary enormously (moreover, innovations arise everyday), obliging us, as administrators to be in constant contact with the field of security to know what we may have to face on a daily basis.

For each of these types attacks, normally one or more methods of attack may be used, which in turn can provoke one or more types of attack.

With regards to where an attack occurs, we need to be clear what can be done or what the objective of the methods will be:

- **Hardware**: in this respect, the threat is directly on accessibility, what will someone who has access to the hardware be able to do? In this case, we will normally need "physical" measures, such as security controls for access to the premises where the machines are located in order to prevent problems of theft or damage to the equipment designed to erase their service. Confidentiality and integrity may also be compromised if physical access to the machines allows some of their devices, such as disk drives, to be used, or if it allows booting of the machines or access to user accounts that may be open.
- **Software**: if accessibility is compromised during an attack, programs may be deleted or disabled, denying access. In the case of confidentiality, it can give rise to unauthorised copies of the software. In the case of integrity, the default functioning of the program could be altered, so that it fails in certain situations or so that it performs tasks in the interest of the attacker, or may simply compromise the integrity of program data: making them public, altering them or simply stealing them.
- **Data**: whether structured, such as in database services, or version management (such as cvs) or simple files. Attacks that threaten accessibility can destroy or eliminate them, thus denying access to them. In the case of confidentiality, we could be allowing unauthorised reading and the integrity would be affected when modifications are made or new data is created.
- **Communication channel** (on the network, for example): for the methods that affect accessibility, it can cause the destruction or elimination of messages and prevent access to the network. In confidentiality, reading and observation of the traffic of messages to or from the machine. And

Note

Attacks may have the purpose of destroying, disabling or spying our components, whether hardware, software or communication systems.

with regards to integrity, any modification, delay, reordering, duplication or falsification of the incoming and/or outgoing messages.

1.1. Techniques used in the attacks

The methods used are various and can depend on an element (hardware or software) or the version of the element. Therefore, we need to maintain the software updated for security corrections that arise and to follow the instructions of the manufacturer or distributor in order to protect the element.

Despite this, there are normally always "fashionable" techniques or methods at any particular time. Some brief notes on today's attack techniques are:

- **Bug exploits:** or exploitation of errors or exploits [CERb] [Ins][San], whether of a hardware, software, service, protocol or of the operating system itself (for example, in the kernel), and normally in a specific version of these. Normally, any computer element is more or less prone to errors in its design, or simply to things that have not been foreseen or taken into account. Periodically, holes are discovered (sometimes known as exploits, or simply bugs), which may be taken advantage of for breaching system security. Normally either generic attack techniques are used, such as the one explained as follows, or particular techniques for the affected element. Every affected element will have someone responsible – whether the manufacturer, developer, distributor or the GNU/Linux community – for producing new versions or patches to handle these problems. As administrators, we are responsible for being informed and maintaining a responsible policy of updates to avoid potential risks of attack. If there are no solutions available, we can also study the possibility of using alternatives for the element or disabling it until we find a solution.
- **Virus:** program normally annexed to others and that uses mechanisms of autocopy and transmission. It is common to annex viruses to executable programs, electronic mails, or to incorporate them into documents or programs that allow macros (not verified). They are perhaps the greatest security plague of the moment.

GNU/Linux systems are protected almost completely against these mechanisms for several reasons: in executable programs, they have very limited access to the system, in particular to the user account. With the exception of the root user, where we have to be very careful with what it executes. Mail does not tend to use non-verified macros (contrary to Outlook and Visual Basic Script in Windows, which is an exploit for the entry of viruses), and in the case of the documents, we are in a similar situation, since they do not support non-verified macros or scripting languages (such as Visual Basic for Applications (VBA) in Microsoft Office).

In any case, we will have to pay attention to what may happen in the future, since specific viruses for GNU/Linux could be created taking advan-

Note

The methods used by attackers are extremely varied and evolve constantly in terms of the technological details that they use.

tage of some bugs or exploits. We must also take a look at mail systems, since although we may not generate viruses, we can transmit them; for example, if our system functions as a mail router, messages with a virus could come in and could then be sent on to others. Here we can implement virus detection and filtering policies. Another plague that could enter the category of viruses are spam messages, which although not usually used as attacking elements, can be considered problematic due to the virulence with which they appear, and the financial cost that they can entail (in loss of time and resources).

- **Worm:** normally this is a type of program that takes advantage of a system bug in order to execute code without a permission. They tend to be used to take advantage of the machine's resources, such as the use of the CPU, when it detects that the system is not functioning or is not in use or, with malicious intent, with the objective of stealing resources or to use them to stop or block the system. Transmission and copying techniques are also commonly used.
- **Trojan horse** (or 'Trojans'): useful programs that incorporate some functionality but hide other functionalities, which are the ones used to obtain information from the system or in order to compromise it. A particular case could be the one of the mobile type codes of web applications such as Java, JavaScript or ActiveX; these normally ask for consent to be executed (ActiveX in Windows), or have limited models of what they can do (Java, JavaScript). But like all software, they also have bugs and are an ideal method for transmitting Trojans.
- **Back door** (or trap door): method for accessing a hidden program that could be used to give access to the system or processed data without our knowledge. Other effects could be changing the system's configuration, or allowing viruses to be introduced. The mechanism employed could come included in some type of common software or in a Trojan.
- **Logic bombs:** program embedded in another program which checks when specific conditions occur (temporary, user actions etc.) to activate itself and perform unauthorised activities.
- **Keyloggers:** special program dedicated to hijacking the interactions with the user's keyboard and/or mouse. They may be individual programs or Trojans incorporated into other programs.
Normally, they would need to be introduced in an open system to which there was access (although more and more frequently they can come incorporated in Trojans that are installed). The idea is to capture any introduction of keys, in such a way as to capture passwords (for example, for bank accounts), interaction with applications, visited websites, completed forms etc.

- **Scanner** (port scanning): rather than an attack, it represents a prior step consisting of gathering potential targets. Basically, it consists of using tools that allow the network to be examined in order to find machines with open ports, whether TCP, UDP or other protocols, which indicate the presence of certain services. For example, scanning machines looking for port 80 TCP, indicates the presence of web servers, from which we can obtain information about the server and the version used in order to take advantage of its known vulnerabilities.
- **Sniffers**: allows to capture packages circulating on a network. With the right tools we can analyse machines' behaviours: which are servers, clients, what protocols are used, and in many case obtaining passwords for insecure services. Initially, they were used a lot for capturing passwords of telnet, rsh, rcp, ftp... insecure services that should not be used (use the secure versions instead: ssh, scp, sftp). Sniffers (and scanners) are not necessarily an attack tool, since they can also serve for analysing our networks and detecting failures, or simply for analysing our own traffic. Normally, the techniques of both scanners and sniffers tend to be used by an intruder looking for the system's vulnerabilities whether to learn the data of an unknown system (scanners), or to analyse its internal interaction (sniffer).
- **Hijacking**: these are techniques that try to place a machine in such a way that it intercepts or reproduces the functioning of a service in another machine from which it has intercepted the communication. They tend to be common in cases of electronic mail, file or web transfers. For example, in the web case, a session may be captured and it will be possible to reproduce what the user is doing, pages visited, interaction with forms etc.
- **Buffer overflows**: fairly complex technique that takes advantage of the programming errors in the applications. The basic idea is to take advantage of overflows in application buffers, whether queues, arrays etc. If the limits are not controlled, an attacking program can generate a bigger message or data than expected and cause failures. For example, many C applications with poorly written buffers, in arrays, if we surpass the limit we can cause the program's code to be overwritten causing a malfunctioning or breakdown of the service or machine. Moreover, a more complex variant allows parts of program to be incorporated in the attack (C compiled or shell scripts), that may allow the execution of any code that the attacker wishes to introduce.

- **Denial of Service** ('DoS attack'): this type of attack causes the machine to crash or overloads one or more services, rendering them unusable. Another technique is DDoS (Distributed DoS), which is based on using a set of distributed machines in order to produce the attack or service overload. This type of attack tends to be solved with software updates, since normally all of the services that were not designed for a specific workload are affected and saturation is not controlled. DoS and DDoS attacks are commonly used in attacks on websites or DNS servers, which are affected by server vulnerabilities, for example, specific versions of Apache or BIND. Another aspect that is worth taking into account is that our system could also be used for DDoS type attacks, through control from a backdoor or a Trojan.

A fairly simple example of this attack (DoS) is known as the SYN flood, which tries to generate TCP packages that open a connection, but then do nothing else with it, simply leaving it open; this spends system resources on data structures of the kernel, and network connection resources. If this attack is repeated hundreds or thousands of times, all of the resources can become occupied without being used, in such a way that when users wish to make use of the service, it is denied because the resources are occupied. Another case is known as mail bombing, or simply resending (normally with a false sender) until mail accounts are saturated, causing the mail system to crash or to become so slow that it is unusable. To some extent these attacks are fairly simple to carry out with the right tools and have no easy solution, since they take advantage of the internal functioning of protocols and services; in these cases we need to take measures of detection and subsequent control.

- **Spoofing**: the techniques of spoofing encompass various methods (normally, very complex) of falsifying both information or the participants in a transmission (origin and/or destination). Some spoofing examples include:
 - IP spoofing, falsification of a machine, allowing false traffic to be generated or intercepting traffic that was directed to another machine. In combination with other attacks, it can even breach firewall protection.
 - ARP spoofing, complex technique (uses a DDoS), which tries to falsify source addresses and network recipients by means of attacking the machines' ARP caches, in such a way that the real addresses are replaced by others in various points of a network. This technique can breach all type of protections, including firewalls, but is not a simple technique.
 - E-mail is perhaps the simplest. It consists of generating false emails, in terms of both content and source address. For this type, techniques of the type known as social engineering are fairly common; these basically trick the user in a reasonable manner, a classical example are false emails from the system administrator or, for example, from the bank where we have our current account, stating that there have been prob-

Web sites

SYN flood, see: <http://www.cert.org/advisories/CA-1996-21.html>

Problems associated to e-mail bombing
amb spamming: http://www.cert.org/tech_tips/email_bombing_spamming.html

Web site

See the case of Microsoft in: <http://www.computerworld.com/softwaretopics/os/windows/story/0,10801,59099,00.html>

lems with the accounts and that we have to send confidential information or the previous password in order to solve them, or asking the password to be changed for a specific one. Surprisingly, this technique (also known as phishing) manages to deceive a considerable number of users. Even with (social engineering of) simple methods: a famous cracker commented that his preferred method was by telephone. As an example, we describe the case of a certification company (*Verisign*), for which the crackers obtained the Microsoft private software signature by just making a call on behalf of a company that said a problem had arisen and that they needed their key again. In summary, high levels of computer security can be overcome by a simple telephone call or by an email badly interpreted by a user.

- **SQL injection:** it is a technique aimed at databases and web servers in particular, which generally takes advantage of the incorrect programming of web forms, where the information provided has not been correctly controlled. It does not determine that the input information is of the correct type (strongly typified in relation to what is expected) or the type or literal characters that are introduced are not controlled. The technique takes advantage of the fact that the literals obtained by the forms (for example web, although the attacks can be sustained from any API that allows access to a database, for example php or perl) are used directly for making consultations (in SQL), which will attack a specific database (to which in principle there is no direct access). Normally, if there are vulnerabilities and poor form control, SQL code can be injected into the form, in such a way that it can make SQL consultations which provide the searched information. In drastic cases, security information could be obtained (database users and passwords), or even entire database tables, or else loss of information or intentional deletion of data. This technique in web environments in particular can be serious, due to the laws on the protection of the privacy of personal data which an attack of this nature can threaten. In this case, rather than an issue of system security, we are dealing with a problem of programming and control with strong typing of the data expected by the application, in addition to the appropriate control of knowledge of vulnerabilities present in the used software (database, web server, API like php, perl...).
- **Cross-side scripting** (or XSS): another problem associated to web environments and, in particular, to alterations of html code and/or scripts that a user can obtain by visualising a particular website, which can be altered dynamically. Generally errors when it comes to validating HTML code are taken advantage of (all navigators have problems with this, due to the definition of HTML itself, which allows reading of practically any HTML code however incorrect it is). In some cases, the use of vulnerabilities can be direct through scripts in the web page, but normally the navigators have good control of these. At the same time, indirectly there are techniques that allow script code to be inserted, either through access to the user's

cookies from the navigator, or by altering the process of redirecting from one web page to another. There are also techniques using frames, that can redirect the HTML code that is being viewed or directly hang the browser. In particular, web sites' search engines can be vulnerable, for allowing script code to be executed. In general, they are attacks with complex techniques, but designed to capture information such as cookies, which can be used for sessions, and thus allow a determined person to be substituted by redirecting websites or obtaining their information. Once more from the system's perspective, it is a question of the software in use. We need to control and know about vulnerabilities detected in navigators (and make the most of the resources that they offer in order to avoid these techniques) and control the use of software (search engines used, versions of the web server, and APIs used in developments).

Some basic general recommendations for security, could be:

- Controlling a problematic factor: users. One of the factors that can most affect security is the confidentiality of passwords, which is affected by users' behaviour; this facilitates actions within the system itself on the part of potential attackers. Most attacks tend to come from within the system, in other words, once the attacker has obtained access to the system.
- Users include those who are forgetful (or indiscreet) and forget their password on a frequent basis, mention it in conversation, write it down on a piece of paper left somewhere or stuck next to the desk or computer, or that simply lend it to other users or acquaintances. Another type of user uses predictable passwords, whether the same as their user id, national identity number, name of girlfriend, mother, dog etc., which with a minimum amount of information can be easily discovered. Another case is normal users with a certain amount of knowledge, who have valid passwords but we should always bear in mind that there are mechanisms capable of discovering them (cracking of passwords, sniffing, spoofing...). We need to establish a "culture" of security among users and, through the use of techniques, oblige them to change their passwords, without using typical words, for long passwords (of more than 2 or 3 characters) etc. Lately, many companies and institutions are implementing the technique of making a user sign a contract obliging the user not to disclose the password or to commit acts of vandalism or attacks from their accounts (although of course this does not prevent others from doing so through the user).
- Not to use or run programs with no guarantee of origin. Normally, distributors use signature verification mechanisms in order to verify that software packages are what they say, like for example md5 sums (command `md5sum`) or the use of GPG signatures [Hatd] (`gpg` command). The seller or distributor provides an md5 sum of their file (or CD image) and we can check its authenticity. Lately, signatures for both individual packages and

for package repositories are used in distributions as a mechanism to ensure the supplier's reliability.

- Not to use privileged users (like the root user) for the normal working of the machine; any program (or application) would have the permissions to access anywhere.
- Not to access remotely with privileged users' privileges or to run programs that could have privileges. Especially if we do not know or have not checked the system's security levels.
- Not to use elements when we do not know how they behave or to try to discover how they behave through repeated executions.

These measures may not be very productive but if we have not protected the system, we have no control over what can happen and, even so, nobody can guarantee that a malicious program cannot sneak in and breach security if we execute it with the right permissions. In other words, in general we need to be very careful with all type of activities related to access and the execution of more or less privileged tasks.

1.2. Countermeasures

With regard to the measures that can be taken against the types of attacks that occur, we can find some preventive measures and some measures for detecting what is happening to our systems.

Let's look at some of the types of measures that we could take in the sphere of intrusion prevention and detection (useful tools are mentioned, some of which we will examine later):

- **Password cracking:** in attacks of brute force designed to crack passwords, it is common to try and obtain access through repeated logins; if entry is obtained, the user's security has been compromised and the door is left open to other types of attacks, such as backdoor attacks or simply the destruction of the user's account. In order to prevent this type of attack, we need to reinforce the passwords policy, asking for a minimum length and regular changes of password. One thing we need to avoid is the use of common words in the passwords: many of these attacks are made using brute force, with a dictionary file (containing words in the user's language, common terms, slang etc.). This type of password will be the first to be broken. It can also be easy to obtain information on the victim, such as name, national identity number or address, and to use this data for testing a password. For all of the above, it is also not recommended to have passwords with national identity numbers, names (own or of relatives etc.), addresses etc. A good choice tends to be a password of between 6 and 8

characters at minimum with alphabetic and numerical contents in addition to a special character.

Even if the password has been well chosen, it may be unsafe if used for unsafe services. Therefore, it is recommended to reinforce the services using encryption techniques that protect passwords and messages. And, on the other hand, to prevent (or not use) any service that does not support encryption, and consequently that is susceptible of attack using methods, such as sniffers; among these, we could include telnet, FTP, rsh, rlogin services.

- **Bug exploits:** avoid having programs available that are not used, are old or are not updated (because they are obsolete). Apply the latest patches and updates that are available for both applications and the operating system. Test tools that detect vulnerabilities. Keep up to date with vulnerabilities as they are discovered.
- **Virus:** use antivirus mechanisms or programs, systems for filtering suspicious messages; avoid the execution of macros (which cannot be verified). We should not minimise the potential effects of viruses, every day they are perfected and technically it is possible to make simple viruses that can deactivate networks in a matter of minutes (we just have to look at some of the recent viruses in the world of Windows).
- **Worm:** control the use of our machines or users outside of normal hours and control incoming and/or outgoing traffic.
- **Trojan horse** (or Trojans): regularly check the integrity of programs using sum or signature mechanisms. Detection of anomalous incoming or outgoing system traffic. Use firewalls to block suspicious traffic. A fairly dangerous version of trojans consist of rootkits (discussed below), which perform more than one function thanks to a varied set of tools. In order to verify integrity, we can use sum mechanisms like md5 or gpg, or tools that automate this process like Tripwire or AIDE.
- **Backdoor** (or trap door): we need to obtain certification that programs do not contain any type of undocumented hidden backdoor from software sellers or suppliers and, of course, only accept software from places that offer guarantees. When the software belongs to third parties or comes from sources that could have modified the original software, many manufacturers (or distributors) will integrate some type of software verification based on sum codes or digital signatures (md5 or gpg type) [Hatd]. Whenever these are available, it is useful to verify them before proceeding to install the software. We can also test the system intensively, before installing it as a production system.

Another problem consists of software alteration a posteriori. In this case, systems of signatures or sums can also be useful for creating codes over already installed software so as to control that no changes are made to

Web sites

See patches for the operating system at: <http://www.debian.org/security>
<http://www.redhat.com//security>
<http://fedoraproject.org/wiki/Security>

Web site

For vulnerabilities, a good tool is Nessus. To discover new vulnerabilities, see CERT in: <http://www.cert.org/advisories/> (old site) and <http://www.us-cert.gov/cas/techalerts/index.html>.

vital software. Or backup copies, which we can make comparisons with in order to detect changes.

- **Logic bombs:** in this case, they tend to be hidden after activations through time or through user actions. We can verify that there are no non-interactive jobs introduced on the system of the crontab or at type and other processes (of the nohup type for example), which are periodically executed, or executed in the background for a long time (w commands, jobs). In any case, we could use preventive measures to prevent non-interactive jobs for users, or only allow them for users that need them.
- **Keyloggers and rootkits:** in this case there would be some intermediary process that would try to capture our pressing of keys and try to store them somewhere. We will have to examine situations where a strange process appears belonging to our user, or to detect if we have any file open with which we are not working directly (for example, lsof could be helpful, see man), or network connections, if we were dealing with a keylogger with external sending. To test a very basic functioning of a simple keylogger, we can see the system *script* command (see *script* man). In the other case, the rootkit (which also tends to include a keylogger) is usually a package of several programs with various techniques that allow the attacker, once inside an account, to use various elements such as a keylogger, backdoors, Trojans (replacing system commands) etc. in order to obtain information and entrance doors to the system, often accompanied by programs that clean the logs, in order to eliminate evidence of the intrusion. A particularly dangerous case is that of rootkits, that are used or come in the form of kernel modules, which allows them to act at the level of the kernel. In order to detect them, we will need to control that there is no external traffic travelling to a specific address. A useful tool for verifying rootkits is *chkrootkit*.
- **Scanner** (port scanning): scanners tend to be launched over one or more loop systems for scanning known ports in order to detect those that are left open and what services are functioning (and to obtain information on the versions of the services) that could be susceptible to attacks.
- **Sniffers:** avoid tapping and thus prevent the possibility of interceptions being inserted. One technique is the network's hardware construction, which can be divided into segments so that the traffic can only circulate through the zone that will be used, placing firewalls to join these segments to be able to control incoming and outgoing traffic. Use encryption techniques so that the messages cannot be read and interpreted by someone intercepting the network. For the case of both scanners and sniffers, we can use tools such as Wireshark [Wir] (formerly Ethereal) and Snort to check our network or, for port scanning, Nmap. Sniffers can be detected on the network by searching for machines in promiscuous Ethernet mode

Web site

We can find the chkrootkit tool in: <http://www.chkrootkit.org>

(intercepting any circulating package); the network card only usually captures the traffic that goes towards it (or of the broadcast or multicast type).

- **Hijacking:** implement mechanisms for services encryption, requiring authentication, and if possible, regularly renewing authentication. Control incoming or outgoing traffic through the use of firewalls. Monitor the network in order to detect suspicious flows of traffic.
- **Buffer overflows:** they tend to be common as bugs or holes in the system, and tend to be resolved through software updates. In any case, through logs, we can observe strange situations of crashed services that should be functioning. We can also maximise the control of processes and access to resources in order to isolate the problem when it occurs in environments of controlled access, such as the one offered by SELinux (see further on in the module).
- **Denial of Service** ('DoS attack') and others, such as SYN flood, or mail bombing: take measures to block unnecessary traffic on our network (through the use of firewalls for example). With the services where it is possible, we will have to control buffer sizes, the number of clients to be attended, connection timeouts, service capacities etc.
- **Spoofing:** a) IP spoofing, b) ARP spoofing, c) electronic mail. These cases require strong service encryption, control through the use of firewalls, authentication mechanisms based on various aspects (for example, not based on the IP, if it could be compromised), mechanisms can be implemented that control established sessions based on several machine parameters at the same time (operating system, processor, IP, Ethernet address etc.). Also monitor DNS systems, ARP caches, mail spools etc. in order to detect changes in the information that invalidate preceding ones.
- **Social engineering:** this is not an IT issue really, but we have to make sure that users do not make security worse. Appropriate measures such as increasing information or educating users and technicians about security: controlling which personnel will have access to critical security information and in what conditions they may cede it to others. A company's help and maintenance services can be a critical point: controlling who has security information and how it is used.
- In relation to end users, improving the culture of passwords, avoiding leaving them noted down anywhere where third parties can see them or simply disclosing them.

2. System security

In the face of potential attacks, we need to have mechanisms for preventing, detecting and recovering our systems.

For local prevention, we need to examine the different mechanisms of authentication and permissions for accessing the resources in order to define them correctly and be able to guarantee the confidentiality and integrity of our information. In this case, we will be protecting ourselves against attackers that have obtained access to our system or against hostile users who wish to overcome the restrictions imposed on the system.

In relation to network security, we need to guarantee that the resources that we offer (if we provide certain services) have the necessary parameters of confidentiality and that the services cannot be used by unauthorised third parties, meaning that a first step will be to control which of the offered services are the ones we really want, and that we are not offering other services that are uncontrolled at the same time. In the case of services of which we are clients, we will also have to ensure the mechanisms of authentication, in the sense that we access the right servers and that there are no cases of substitution of services or servers (normally fairly difficult to detect).

With regards to the applications and the services themselves, in addition to guaranteeing the right configuration of access levels using permissions and authentication of authorised users, we need to monitor the possible exploitation of software bugs. Any application, however well designed and implemented may have a more or less high number of errors that can be taken advantage of in order to overcome imposed restrictions using certain techniques. In this case, we enforce a policy of prevention that includes keeping the system updated as much as possible, so that we either update whenever there is a new correction or if, we are conservative, we maintain those versions that are the most stable in security terms. Normally, this means periodically checking several security sites in order to learn about the latest failures detected in the software and the vulnerabilities that stem from them that could expose our systems to local or network security failures.

3. Local security

Local security [Peñ] [Hatb] is basic for protecting the system [Deb][Hatc], since normally following a first attempt from the network, it is the second protection barrier before an attack that manages to obtain partial control of the machine. Also, most attacks end up using the system's internal resources.

Note

Various attacks, although they may come from the outside, are designed to obtain local access.

3.1. Bootloaders

With regards to local security, problems can already start with booting with the physical access that an intruder could gain to a machine.

One of the problems starts when the system boots. If the system can be booted from disk or CD, an attacker could access the data of a GNU/Linux (or also Windows) partition just by mounting the file system and placing themselves as root users without needing to have any password. In this case, we need to protect the system's boot from the BIOS, for example, by protecting the access with a password, so that booting from a CD is not allowed (for example through a Live CD or diskette). It is also reasonable to update the BIOS, since it can also have security failures. Plus, we need to be careful because many BIOS manufacturers offer additional known passwords (a sort of backdoor), meaning that we cannot depend exclusively on these measures.

The following step is to protect the boot loader, whether lilo or grub, so that the attacker is not able to modify the start up options of the kernel or directly modify the boot (in the case of grub). Either of the two can also be protected using passwords.

In grub, the file `/sbin/grub-md5-crypt` asks for the password and generates an associated md5 sum. Then, the obtained value is entered into `/boot/grub/grub.conf`. Under the *timeout* line, we introduce:

```
password --md5 sum-md5-calculated
```

For lilo we place, either a global password with:

```
password = <selected password>
```

or one in the partition that we want:

```
image = /boot/vmlinuz-version
    password = <selected password>
    restricted
```

In this case *restricted* also indicates that we will not be able to change the parameters passed onto the kernel from the command line. We need to take care to set the file `/etc/lilo.conf` as protected so that only the root user has read/write privileges (`chmod 600`).

Another issue related to booting is the possibility that someone with access to the keyboard could reinitiate the system because if they press CTRL+ALT+DEL, (in some distributions it is now disabled by default) they will cause the machine to shutdown. This behaviour is defined in `/etc/inittab`, with a line like:

```
ca:12345:ctrlaltdel:/sbin/shutdown -t1 -a -r now
```

If commented, this possibility of reinitiating will become deactivated. Or on the other hand, we can create a file `/etc/shutdown.allow`, which allows certain users to reinitiate.

3.2. Passwords and shadows

The typical passwords of the initial UNIX systems (and of the first versions of GNU/Linux) were encrypted using DES algorithms (but with small keys and a system call that was responsible for encrypting and decrypting, specifically `crypt`, see the man).

Normally, they were in the file `/etc/passwd`, in the second field, for example:

```
user:sndb565sadsd:...
```

But the problem lies in the fact that this file is legible by any user, meaning that an attacker could obtain the file and use an attack of brute force, until decrypting the passwords that the file contained, or use an attack of brute force with dictionaries.

The first step is to use the new files `/etc/shadow`, where the passwords are now saved. This file is only legible by the root user and by nobody else. In this case, in `/etc/passwd` an asterisk (*) appears where previously the encrypted password was. By default, current GNU/Linux distributions use passwords of the shadow type unless told not to use them.

A second step is to change the system of encrypting the passwords for one that is more complex and difficult to break. Now, both Fedora and Debian offer passwords by md5; we are usually allowed to choose the system at the time of the installation. We need to take care with md5 passwords, because if we use NIS, we could have a problem; otherwise, all clients and servers will use md5 for their passwords. Passwords can be recognised in `/etc/shadow` because they have a "\$1\$" prefix.

Other possible actions include obliging users to change password frequently (the *change* command can be useful), imposing restrictions on the size and content of the passwords, and validating them with dictionaries of common terms .

Regarding the tools, it is interesting to have a password cracker (i.e. a program for obtaining passwords), in order to check the real security situation with our users' accounts, and thus forcing change in the ones we detect to be insecure. Two of the ones most commonly used by administrators are *John the Ripper* and *crack*. They can also work with a dictionary, so it will be interesting to have some ASCII dictionary in English (can be found on the web). Another tool is "Slurpie", which can test several machines at the same time.

An issue that we always need to take into account is to run these tests on our systems. We must not forget that the administrators of other systems (or the access or ISP provider) will have intrusion detection systems enabled and that we could be denounced for attempts at intrusion, either before the competent authorities (computer crime units) or before our ISP so that they close down our access. We need to be very careful with the use of security tools, which are always on the edge of security or intrusion.

3.3. Suid and sticky bits

Another important problem affects some special permissions used on files or script.

The sticky bit is used especially on temporary directories, where we want in some (sometimes unrelated) groups for any user to be able to write, but only the owner of the directory to be able to delete, or the owner of the file that is within the directory. A classical example of this bit is the temporary directory */tmp*. We need to make sure that there are no directories of this type, since they can allow anyone to write on them, so that we must check that there are no more than those that are purely necessary as temporaries. The bit is placed using (*chmod +t dir*), and can be removed with *-t*. In an *ls* command it will appear as a directory with *drwxrwxrwt* permissions (take note of the last *t*).

The bit *setuid* allows a user to execute (whether an executable or a shell script) with another user's permissions. In some cases this can be useful, but it is also potentially dangerous. This is the case, for example, of programs with *setuid* as root: a user, although without root permissions, can execute a program with *setuid* that could have internal root user permissions. This is very dangerous in the case of scripts, since they could be edited and modified to do anything. Therefore, we need to keep these programs controlled, and if *setuid* is not necessary, we need to eliminate it. The bit is placed using *chmod +s*, whether applying it to the owner (then it is called *suid*) or to the group (then it is called

bit `sgid`); it can be removed with `-s`. In the case of viewing with `ls` command, the file will appear with `-rwSrwx-rw` (take note of the `S`), if it is only `suid`, in `sgid` the `S` would appear after the second `w`.

In the case of using `chmod` with octal notation, four numbers are used, where the last three are the classical `rw-rw-rw` permissions (remember that we have to add in the number 4 for `r`, 2 `w`, and 1 for `x`), and the first has a value for every special permission that we want (which are added): 4 (for `suid`), 2 (`sgid`), and 1 (for sticky).

3.4. Enabling hosts

The system has several special configuration files that make it possible to enable the access of a number of hosts to some network services, but whose errors could later allow local security to come under attack. We can find:

- `user .rhosts`: allows a user to specify a number of machines (and users) that can use their account through `"r"` commands (`rsh`, `rcp`...) without having to enter the account's password. This is potentially dangerous, since a poor user configuration could allow entry to unwanted users or could allow an attacker (with access to the user account) to change the addresses in `.rhosts` in order to enter comfortably without any type of control. Normally, we should not allow these files to be created and we should even delete them completely and disable the `"r"` commands.
- `/etc/hosts.equiv`: this is exactly the same as with the `.rhosts` files but at the level of the machine, specifying what services, what users and what groups can access `"r"` commands without the need for password control. Also, an error such as putting a `"+"` on a line of that file allows access to "any" machine. Nowadays, this file does not usually exist either and there is always the alternative of the `ssh` service to `"r"`.
- `/etc/hosts.lpd`: in the `LPD` printing system it was used to put the machines that could access the printing system. We need to be very careful, if we are not serving, to completely disable access to the system, and if we are serving, to restrict to a maximum the machines that really make use of it. Or try to change to a `CUPS` or `LPRng` system, which has far more control over the services. The `LPD` system is a common target of worm-type or buffer overflow attacks and several important bugs are documented. We need to be on the lookout if we use this system and the `hosts.lpd` file.

3.5. PAM modules

PAM modules [Peñ][Mor03] are a method that allows the administrator to control how the user authentication process is performed for certain applications. The applications need to have been created and linked to the PAM libraries.

Basically, PAM modules are a set of shared libraries that can be incorporated into applications as a method for controlling their user authentication. Also, the authentication method can be changed (by means of the PAM modules configuration), without having to change the application.

The PAM modules (libraries) tend to be in the `/lib/security` directory (in the form of dynamically loadable file objects). And the PAM configuration is present in the `/etc/pam.d` directory, where a PAM configuration file appears for every application that is using PAM modules. We find the authentication configuration of applications and services such as `ssh`, graphic login of X Window System, like `xdm`, `gdm`, `kdm`, `xscreensaver`... or, for example, the system login (entrance with username and password). Old PAM versions used a file (typically in `/etc/pam.conf`), where the PAM configuration was read if the `/etc/pam.d` directory did not exist.

The typical line of these files (in `/etc/pam.d`) would have this format (if using `/etc/pam.conf` we would have to add the service to which it belongs as a first field):

```
module-type control-flag module-path arguments
```

which specifies:

- a) module type: if it is a module that requires user authentication (*auth*), or has restricted access (*account*); things we need to do when the user enters or leaves (*session*); we the user has to update the password.
- b) control flags: they specify whether it is *required*, a *requisite*, whether it is *sufficient* or whether it is *optional*. This is a syntax. There is another more up to date one that works in pairs of value and action.
- c) the module path.
- d) arguments passed onto the module (they depend on each module).

Because some services need several common configuration lines, it is possible to have operations for including common definitions for other services, we just have to add a line with:

```
@include service
```

A small example of the use of PAM modules (in a Debian distribution), could be their use in the login process (we have also listed the lines included from other services):

auth		requisite pam_securetty.so

Web site

For further information, see "The Linux-PAM System Administrators' Guide": <http://www.kernel.org/pub/linux/libs/pam/Linux-PAM-html>

auth		requisitepam_nologin.so
auth		requiredpam_env.so
auth		requiredpam_unix.so nullok
account	required	pam_unix.so
session	required	pam_unix.so
session	optional	pam_lastlog.so
session	optional	pam_motd.so
session	optional	pam_mail.so standard noenv
password	required	pam_unix.so nullok obscure min = 4 max = 8 md5

This specifies the PAM modules required to control user authentication during login. One of the modules, `pam_unix.so`, is the one that really verifies the user's password (looking at files `password`, `shadow...`).

Others control the session to see when the latest entry was or save when the user enters and leaves (for the `lastlog` command), there is also a module responsible for verifying whether the user has mail to read (authentication is also required) and another that controls that the password changes (if the user is obliged to do so with the first login) and that it has 4 to 8 letters, and that `md5` can be used for encryption.

In this example we could improve user security: the *auth* and the passwords allow passwords of length nil: this is the module's `nullok` argument. This would allow having users with empty passwords (potential source of attacks). If we remove this argument, we no longer allow empty passwords for the login process. The same can be done in the password configuration file (in this case, the `passwords change` command), which also presents `nullok`. Another possible action is to increase the maximum size of the passwords in both files, for example, with `max = 16`.

3.6. System alterations

Another problem can be the alteration of basic system commands or configurations, through the introduction of Trojans, or backdoors, by merely introducing software that replaces or slightly modifies the behaviour of the system's software.

A typical case is the possibility of forcing the root to execute false system commands; for example, if the root were to include the `"."` in its variable `PATH`, this would allow commands to be executed from its current directory, which would enable the placing of files that replaced system commands and that would be executed as a priority before the system's commands. The same process can be done with a user, although because a user's permissions are more

Web site

AusCert UNIX checklist:
<http://www.auscert.org.au/5816>

limited, it may not affect the system as much, rather the security of the user itself. Another typical case is the one of false login screens, replacing the typical login process, password, with a false program that would store the entered passwords.

In the case of these alterations, it will be vital to enforce a policy of auditing changes, whether through a calculation of signatures or sums (gpg or md5), or using some type of control software such as Tripwire or AIDE. For Trojans we can have different types of detections or use tools such as *chkrootkit*, if these come from the installation of some known rootkit.

Web site

chkrootkit, see: <http://www.chkrootkit.org>

4. SELinux

Traditional security within the system has been based on discretionary access control (DAC) techniques, whereby normally each program has full control over the access to its resources. If a specific program (or the user permitting) decides to make an incorrect access (for example, leaving confidential data open, whether through negligence or malfunctioning). Therefore in DAC, a user has full control over the objects that belong to him or her and the programs he or she executes. The executed program will have the same permissions as the user who is executing it. Therefore, the system's security will depend on the applications that are being executed and on the vulnerabilities that these may have or on the malicious software that these may include, and will especially affect the objects (other programs, files or resources) to which the user has access. In the case of the root user, this would compromise the global security of the system.

On a separate note, mandatory access control (MAC) techniques, develop security policies (defined by the administrator) where the system has full control over the rights of access granted over each resource. For example, we can give access to files with permissions (of the Unix type), but, with MAC policies, we have extra control to determine explicitly what files a process is allowed to access and what level of access we wish to grant. Contexts that specify in what situations an object can access another object are established.

SELinux [NSAb] is a MAC type component recently included in branch 2.6.x of the kernel, which the distributions are progressively incorporating: Fedora/Red Hat have it enabled by default (although it is possible to change it during the installation) and it is an optional component in Debian.

SELinux implements MAC-type security policies, which allow more refined access permissions than traditional UNIX file permissions. For example, the administrator could allow data to be added to a log file, but not to rewrite or truncate it (techniques commonly used by attackers to erase their tracks). In another example, we could allow network programs to link to the port (or ports) they require, but deny access to other ports (for example, it could be a technique that helps to control certain Trojans or backdoors).

SELinux was developed by the US NSA agency with direct contributions from various companies for UNIX and free systems, such as Linux and BSD. It was freed in the year 2000 and since then it has been integrated in different GNU/Linux distributions.

In SELinux we have a domain-type model, where each process runs in a so-called security context and any resource (file, directory, socket etc.) has a type associated to it. There is a set of rules that indicates what actions can be performed in each context on each type. One of the advantages of this context-type model is that the policies defined can be analysed (there are tools) for determining what flows of information are allowed, for example, to detect various routes of attack, or whether the policy is sufficiently complete so as to cover all potential accesses.

It has what is known as the *SELinux policy database* which controls all aspects of SELinux. It determines what contexts each program can use to run and specifies what types of each context can be accessed.

In SELinux, every system process has a context consisting of three parts: an identity, a role and a domain. The identity is the name of the user account or system_u for system processes or user_u if the user has no defined policies. The role determines what the associated contexts are. For example user_r is not allowed to have the context sysadm_t (main domain for the system administrator). Therefore a user_r with identity user_u cannot obtain a sysadm_t context in anyway. A security context is always specified by this set of values like:

```
root:sysadm_r:sysadm_t
```

is the context for the system administrator, defines its identity, role and security context.

For example, in a machine with SELinux activated (in this case a Fedora) we can see the -Z option of the ps of contexts associated to the processes:

```
# ps ax -Z
```

LABEL	PID	TTY	STAT	TIME	COMMAND
system_u:system_r:init_t	1	?	Ss	0:00	init
system_u:system_r:kernel_t	2	?	S	0:00	[migration/0]
system_u:system_r:kernel_t	3	?	S	0:00	[ksoftirqd/0]
system_u:system_r:kernel_t	4	?	S	0:00	[watchdog/0]
system_u:system_r:kernel_t	5	?	S	0:00	[migration/1]
system_u:system_r:kernel_t	6	?	SN	0:00	[migration/1]

Web sites

Some resources on SELinux:
<http://www.redhat.com/docs/manuals/enterprise/RHEL-4-Manual/selinux-guide/>
<http://www.nsa.gov/research/selinux/index.shtml>
<http://fedoraproject.org/wiki/SELinux>

LABEL	PID	TTY	STAT	TIME	COMMAND
system_u:system_r:kernel_t	7	?	S	0:00	[watchdog/1]
system_u:system_r:syslogd_t	2564	?	Ss	0:00	syslogd -m 0
system_u:system_r:klogd_t	2567	?	Ss	0:00	klogd -x
system_u:system_r:irqbalance_t	2579	?	Ss	0:00	irqbalance
system_u:system_r:portmap_t	2608	?	Ss	0:00	portmap
system_u:system_r:rpcd_t	2629	?	Ss	0:00	rpc.statd
user_u:system_r:unconfined_t	4812	?	Ss	0:00	/usr/libexec/gconfd-2 5
user_u:system_r:unconfined_t	4858	?	Sl	0:00	gnome-terminal
user_u:system_r:unconfined_t	4861	?	S	0:00	gnome-pty-helper
user_u:system_r:unconfined_t	4862	pts/0	Ss	0:00	bash
user_u:system_r:unconfined_t	4920	pts/0	S	0:00	gedit
system_u:system_r:rpcd_t	4984	?	Ss	0:00	rpc.idmapd
system_u:system_r:gpm_t	5029	?	Ss	0:00	gpm -m /dev/input/mice -t exps2
user_u:system_r:unconfined_t	5184	pts/0	R+	0:00	ps ax -Z
user_u:system_r:unconfined_t	5185	pts/0	D+	0:00	Bash

and with `ls` using the `-Z` option we can see the contexts associated to files and directories:

```
# ls -Z
drwxr-xr-x josep josep user_u:object_r:user_home_t Desktop
drwxrwxr-x josep josep user_u:object_r:user_home_t proves
-rw-r--r-- josep josep user_u:object_r:user_home_t yum.conf
```

and from the console we can find out our current context with:

```
$ id -Z
user_u:system_r:unconfined_t
```

In relation to the functioning mode, SELinux presents two modes known as: *permissive* and *enforcing*. In *permissive*, unauthorised access is allowed, but is audited in the corresponding logs (normally directly over `/var/log/messages` or, depending on the distribution, with the use of audit in `/var/log/audit/audit.log`). In *enforcing* mode no type of access that is not allowed by the defined policies is permitted. We can also deactivate SELinux through its configuration file (normally in `/etc/selinux/config`), by setting `SELINUX=disabled`.

We need to take care with activating and deactivating SELinux, especially with the labelling of contexts in the files, since during periods of activation/deactivation labels could be lost or simply not made. Likewise, when backing up the file system, we need to make sure that the SELinux labels are preserved.

Another possible problem to be taken into account is the large number of security policy rules that can exist and that can cause limitations in terms of controlling the services. In the face of a specific type of malfunctioning, it is worth determining first that it is not precisely SELinux that is preventing functioning due to a too strict security limitation (see the section on SELinux criticism) or options that we did not expect to have activated (can require a change in the configuration of the Booleans as we will see).

In relation to the policy applied, SELinux supports two different types: *targered* and *strict*. In targered policy type, most processes operate without restrictions and only specific services (some daemons) are put into different security contexts that are confined to security policy. In strict policy type, all processes are assigned to security contexts and confined to defined policies, in such a way that any action is controlled by the defined policies. In principle, these are the two types of policies defined in general, but the specification is open to include more.

A special case of policy is the multilevel security (MLS), which is a multilevel policy of the strict type. The idea is to define different levels of security within the same policy, with security contexts having an additional field of access level associated to them. This type of security policy (like MLS) tends to be used in governmental and military organisations, where there are hierarchical structures with different levels of privileged information, levels of general access and different capabilities of action at each level. In order to obtain some security certifications, we need to have this type of security policy.

We can define what type of policy will be used in `/etc/selinux/config`, variable `SELINUXTYPE`. The corresponding policy and its configuration will normally be installed in the directories `/etc/selinux/SELINUXTYPE/`, for example, in the policy subdirectory we tend to find the binary file of the policy compiled (which is what is loaded in the kernel, when SELinux is initiated).

4.1. Architecture

SELinux architecture consists of the following components:

- Code at kernel level
- Shared SELinux library
- The security policy (the database)
- Tools

Let us take a look at a few considerations with regards to each component:

- The kernel code monitors the system's activity and ensures that the requested operations are authorised under the current SELinux security policy configuration, not allowing unauthorised operations and normally generating log entries of denied operations. The code is currently integrated in kernels 2.6.x and, in previous ones, is offered as a series of patches.
- Most SELinux utilities and components not directly related to the kernel use the shared library called `libselinux1.so`, which provides an API for interacting with SELinux.
- The security policy is what is integrated in the SELinux rules database. When the system starts up (with SELinux activated), it loads the binary policy file, which normally resides in `/etc/security/selinux` (although it can vary according to the distribution).
The binary policy file is created on the basis of a compilation (via *make*) of the policy source files and some configuration files.
Some distributions (such as Fedora) do not install the sources by default, which we can normally find in `/etc/security/selinux/src/policy` or in `/etc/selinux`.
Normally, these sources consist of various groups of information:
 - The files related to the compilation, *makefile* and associated scripts.
 - Initial configuration files, associated users and roles.
 - Type-enforcement files, which contain most sentences of the policy language associated to a particular context. We need to take into account that these files are enormous, typically tens of thousands of lines, which means that we can encounter the problem of finding bugs or defining changes in policies.
 - And files that serve to label the contexts of the files and directories during loading or at specific moments.
- Tools: include commands used to administrate and use SELinux. Modified versions of standard Linux commands. And Tools for the analysis of policies and for development.

Let's look from this last section at the typical tools that we can generally find:

Some of the main commands:

Name	Use
chcon	Labels a specific file, or set of files with a specific context.

Name	Use
checkpolicy	Performs various actions related to policies, including compilation of policies to binary, typically invoking from the makefile operations.
getenforce	Generates a message with the SELinux mode (<i>permissive</i> or <i>enforcing</i>). Or deactivated if this is the case.
getsebool	Obtains the list of Booleans, in other words, the list of on/off options for every context associated to a service or general option of the system.
newrole	Allows the transition of a user from one role to another.
runn_init	Used in order to activate a service (<i>start</i> , <i>stop</i>), ensuring that it is performed in the same context as when it is started up automatically (with <i>init</i>).
setenforce	Changes the mode of SELinux, 0 <i>permissive</i> , 1 <i>enforcing</i> .
setfiles	Labels directories and subdirectories with the appropriate contexts, it is typically used in the initial SELinux configuration.
setstatus	Obtains the system status with SELinux.

Also, other common programs are modified to support SELinux such as:

- **cron**: modified to include the contexts for jobs in progress by cron.
- **login**: modified to place the initial security context for users when they log in the system.
- **logrotate**: modified to preserve the context of the logs when they have been compiled.
- **pam**: modified to place the user's initial context and to use the SELinux API and obtain privileged access to password information.
- **ssh**: modified to place the user's initial context when the user logs in the system.
- And various additional programs that modify `/etc/passwd` or `/etc/shadow`.

Also some distributions include tools for managing SELinux, such as the `se-tools(-gui)`, which carry several tools for managing and analysing policies. As well as specific tools for controlling the contexts associated to the different services supported by SELinux in the distribution, for example the `system-config-security level` tool in Fedora has a section for configuring SELinux as we can see in the following figure:



Figure 1. Interface in Fedora for configuring SELinux

In the figure, we can see the Booleans configuration for different services and generic options, including the web server. We can also obtain this list with the `getsebool -a` command and, with the `setsebool/togglesebool` command, we can activate/deactivate the options.

In Fedora, for example, we find Booleans support for (among others): Cron, FTP, httpd (apache), dns, grub, lilo, nfs, nis, cups, pam, ppd, samba, protections against undue access to the process memory etc.

The configuration of Booleans allows the SELinux policy to be tailored during running time. Booleans are used as conditional values of the applied policy rules, which allow policy modifications without having to load a new policy.

4.2. Criticism

Some administrators and security experts have criticised SELinux in particular for being too complex to configure and administer. It is argued that due to its intrinsic complexity, even experienced users can commit errors, leaving the SELinux configuration insecure or unusable and the system vulnerable. Although to a certain extent this is debatable, since even if we have SELinux

badly configured, the UNIX permissions would remain active, SELinux will not allow an operation that the original permissions already did not allow, in fact, we can see this as another stricter level of security.

Performance factors may also be affected, due to the enormous size of the policies, which use up a lot of memory and take a lot of time to load, and, in some cases, due to the processing of rules. We need to bear in mind that we are dealing with a system of practically 10,000 policy rules. And that this number can be even greater if we select the *strict* type policy where we need to specify absolutely all the options to be controlled. Normally, the processing of policy in binary format and the use of Booleans in order to disable rules allows the system to be used more efficiently.

Another aspect that tends to bother administrators is the additional problem of determining, in the event of a malfunction, what the origin or initial cause is. Because it is common for us to find in the end that the problem has stemmed from an excessively restrictive configuration (perhaps due to unawareness on the part of the administrator) of SELinux for a particular service.

In the last instance, we need to point out the extensive support that SELinux offers for security and that, as administrators, we need to be aware of the capabilities and dangers of any new technique that we employ.

5. Network security

5.1. Service client

As service clients, we basically need to make sure that we do not put our users in danger (or they put themselves in danger) by using insecure services. Avoid the use of services that do not use data encryption and passwords (FTP, telnet, non-secure mail). Use encrypted connection techniques, such as SSH and SSL.

Note

As service clients, we will need to avoid using insecure services.

Another important point concerns the potential substitution of servers for other false ones or session hijacking techniques. In these cases, we will need to have powerful authentication mechanisms that allow us to verify the servers' authenticity (for example, SSH and SSL have some of these mechanisms). And we will also have to verify the network searching for intruders who try to replace servers, as well as to apply correct package filtering services using firewalls, which allow us to remove our packages from a request and use the right servers, controlling the incoming packages that we receive as a response.

5.2. Server: inetd and xinetd

As we have seen, network services [Mou01] are configured from various places [Ano99][Hat01][Peñ]:

- In `/etc/inetd.conf` or the equivalent directory in `/etc/xinetd.d`: these systems are sort of "superservers" since they control subsidiary services and start up conditions. The `inetd` service is used in Debian and `xinetd` in Fedora (in Debian it can be installed as an option to replace `inetd`).
- Servers initiated during start up: depending on the runlevel we will have a number of servers initiated. The start up will originate in the directory associated to the runlevel. For example, in Debian, the default runlevel is 2, the services will be started up from the `/etc/rc2.d` directory, certainly with links to the scripts contained in `/etc/init.d`, which will run with the parameter `start`, `stop`, `restart`, as applicable.
- Other RPC type services: associated to remote calls between machines are used, for example in NIS and NFS. We can examine which ones with the `rpcinfo -p` command.

Other support files (with useful information) include: `/etc/services`, which consists of a list of known local or network services together with the protocol name, (tcp, udp or others), used for the service and the port that it uses; `/etc/protocols` is a list of known protocols; and `/etc/rpc` is a list of RPC servers

together with the used ports. These files come with the distribution and are a sort of database used by the network tools and commands in order to determine the name of services and their associated protocols or rpc and ports. We should mention that they are more or less historical files, which do not necessarily contain all the definitions of protocols and services; likewise we can search different Internet lists of known ports.

One of the administrator's first actions will be to disable all services that are not being used or that are not scheduled to be used, reading up on the use of services [Mou01] and what software may need them. [Neu]

In the case of `/etc/inetd.conf`, we just have to comment the service line that has to be disabled, by placing a number symbol (#) as the first character on the line.

In the other model of services, used by default in Fedora (and optionally in Debian), `xinetd`, the configuration lies in the `/etc/xinetd.conf` file, where some of the default values of log, control are configured and then the configuration of each subsidiary service is done through a file within the `/etc/xinetd.d` directory. In each file, the service information is defined, equivalent to what appears in the `inetd.conf`, in this case, to disable a service, we just have to enter the line "disable = yes" within the service file. `Xinetd` has a more flexible configuration than `inetd`, since it separates the configuration of the different services into different files and has a fair number of options for limiting connections to a service, their number or capabilities; all of which allows for a better control of the service and with the right configuration we can avoid some of the attacks by denying the service (DoS o DDoS).

With regard to the handling of runlevel services from the distribution's commands, we have already mentioned several tools that allow services to be enabled or disabled in the unit on local administration. There are also graphic tools such as *kysysv* of KDE, or the `system-config-services` and *ntsysv* in Fedora (in Debian, we recommend *sysv-rc-conf*, *rcconf* or *bum*). And at a lower level, we can go to the runlevel that we want (`/etc/rcx.d`) and deactivate the services we wish by changing the initial S or K of the script for other text: for example, one method would be: changing `S20ssh`, for `STOP_S20ssh`, and it will no longer start up; the next time we need it, we can remove the prefix and it will be active again. Or perhaps the recommended use of simple utilities to place, remove or activate a specific service (like `service` and *chkconfig* in Fedora or similar ones in Debian, such as *update-rc.d* and *invoke-rc.d*).

Another aspect is closing down insecure services. Traditionally, in the world of UNIX file transfer systems such as FTP were used with remote connection, such as telnet, and remote run commands (login or copy), many of which started with the letter "r" (for example, rsh, rcp, rexec...). Other potential dangers are *finger* and *rwhod* services, which allowed information to be obtained from the network of the machine users; here the danger lay in the information

that an attacker could obtain that would make the attacker's job easier. All of these services should not be used currently due to the potential dangers that they entail. In relation to the first group:

a) in network transmissions, ftp and telnet do not encrypt passwords and anyone can obtain pde service passwords or the associated accounts (for example, by using a sniffer).

b) rsh, rexec, rcp also have the problem that, under certain conditions, passwords are not even necessary (for example, if run from places validated in the .rhosts file), which means that once again they are insecure and leave the doors wide open to attacks.

The alternative is to use secure clients and servers that support message encryption and the authentication of participants. There are secure alternatives to the classical servers, but currently the most commonly used solution is the OpenSSH package (which can also be combined with OpenSSL for web environments). OpenSSH offers solutions based on the ssh, scp and sftp commands, allowing old clients and servers to be replaced (using a daemon called sshd). The ssh command allows the old functionalities of telnet, rlogin and rsh among others, scp would be the secure equivalent of rcp and sftp the equivalent of ftp.

With regards to SSH, we also have make sure we use ssh version 2. The first version has some known exploits; we need to take care when we install OpenSSH and, if we do not need the first version, install only the support for ssh2 (see the option Protocol in the /etc/ssh/ssh_config configuration file).

Besides, most services that we leave active on our machines would have to be filtered afterwards by a firewall to make sure that they are not used or attacked by people to whom they are not directed.

6. Intrusion detection

With intrusion detection systems [Hat01] (IDS) the aim is to take a step forward. Once we have been able to configure our security correctly, the next step will be to detect and actively prevent intrusions.

IDS systems create listening procedures and generate alerts when they detect suspicious situations, in other words, they look for the symptoms of potential security incidents.

We have systems based on local information, for example, gathering information from the system logs, monitoring changes in the file system or in the configurations of typical services. Other systems are based on the network and verify that there is no strange behaviour, such as spoofing, with the falsification of known addresses; controlling suspicious traffic, potential service denial attacks, detecting excessive traffic towards particular services, controlling that there are no network interfaces in promiscuous mode (a symptom of sniffers or package capturers).

Examples

Some examples of IDS tools: Logcheck (log verification), TripWire (system status through md5 sums applied to the files), AIDE (a free version of TripWire), Snort (IDS for verifying the status of an entire network).

Note

IDS systems allow us to detect on time intruders using our resources or exploring our systems in search of security failures.

7. Filter protection through wrappers and firewalls

TCP wrappers [Mou01] are programs that act as intermediaries between the requests of the users of a service and the daemons of the servers that provide the service. Most distributions already come with the *wrappers* activated and we configure the levels of access. The *wrappers* tend to be used in combination with *inetd* or *xinetd*, so as to protect the services that they offer.

The *wrapper* basically replaces the service's daemon for another that acts as an intermediary (called *tcpd*, normally in */usr/sbin/tcpd*). When this receives a request, it verifies the user and the origin of the request, in order to determine whether the configuration of the service's *wrapper* allows it to be used or not. Also, it includes the ability to generate logs, or to inform via email possible attempts at access and then runs the appropriate daemon assigned to the service.

For example, let's assume the following entry in *inetd*:

```
finger stream tcp nowait nobody /usr/etc/in.fingerd
in.fingerd
```

We change it for:

```
finger stream tcp nowait nobody /usr/sbin/tcpd in.fingerd
```

so that when a request arrives, it is handled by the *tcpd* daemon which will be responsible for verifying the access (for more detailed information, see the *tcpd* man pages).

There is also an alternative method of TCP wrapper that consists of compiling the original application with the wrappers library. This way the application does not have to be in *inetd* and we can control it like in the first case with the configuration that we will discuss next.

The wrappers system is controlled from the */etc/hosts.deny* file, where we specify which services we deny to whom, using options, like a small shell to save the information on the attempt, and the */etc/hosts.allow* file, where we place the service we intend to use, followed by the list of who is allowed to use the service (later, in the workshop, we will look at a small example). We also have the *tcpdchk* commands, which test the configuration of the hosts files (see *man hosts_access* and *hosts_options*) to check that they are

Note

Wrappers allow us to control security through access lists to levels of services.

correct, in other words, it tests the configuration. The other useful command is `tcpdmatch`, to which we give the name of a service and a potential client (user, and/or *host*), and it tells us what the system will do in this situation.

7.1. Firewalls

A firewall is a system or group of systems that reinforces policies of access control between networks. The firewall can be implemented in software as a specialised application running on an individual computer or could be a special device designed to protect one or more computers.

In general, we will have either a firewall application to protect a specific machine directly connected to Internet (directly or through a provider), or we can place one or several machines designed for this function on our network in order to protect our internal network.

Technically, the best solution is to have one computer with two or more network cards that isolate the different connected networks (or network segments), in such a way that the firewall software on the machine (or if it is a special hardware) is responsible for connecting network packages and determining which can pass or not and to which network.

This type of firewall is normally combined with a router to link the packages of the different networks. Another typical configuration is the firewall towards the Internet, for example with two network cards: on one we obtain/provide traffic to the Internet and on the other we send or provide traffic to our internal network, thus eliminating traffic that is not addressed to us and also controlling traffic moving out towards the Internet, in case we do not wish to allow access to certain protocols or if we suspect that there are potential information leaks due to some attack. A third possibility is the individual machine connected with a single card towards the Internet, either directly or through a provider. In this case, we just want to protect our machine from intruders, unwanted traffic or traffic that is susceptible to data robbery.

In other words, in all these cases we can see that a firewall can have different configurations and uses depending on whether it is software or not, on whether the machine has one or several network cards or on whether it protects an individual machine or a network.

In general, the firewall allows the user to define a series of access policies (which machines can be connected to do or which machines can receive information and what type of information) by means of controlling the allowed incoming or outgoing TCP/UDP ports. Some firewalls come with preconfigured policies; in some cases they just ask whether we want a high, medium or low level of security; others allow all options to be tailored (machines, protocols, ports etc.).

Note

Firewalls make it possible to establish security at the level of packages and communication connections.

Another related technique is network address translation (NAT). This technique provides a route for hiding IP addresses used on the private network and hides them from the Internet, but maintains the access from the machines. One of the typical methods is the one known as masquerading. Using NAT masquerading, one or several network devices can appear as a single IP address seen from the outside. This allows several computers to be connected to a single external connection device; for example, the case of an ADSL router at home that allows several machines to be connected without the need for the provider to give us various IP addresses. ADSL routers often offer some form of NAT masquerading, and also firewall possibilities. It is fairly common to use a combination of both techniques. In this case, as well as the configuration of the firewall machine (in the cases we have seen above), the configuration of the internal private network that we want to protect also comes into play.

7.2. Netfilter: IPtables

The Linux kernel (as of versions 2.4.x) offers a filtering subsystem called Netfilter [Net], which offers package filtering features as well as NAT. This system allows different filter interfaces to be used, the most commonly used one is called IPtables. The main control command is *iptables*. Previously [Hata], it provided another filter called *ipchains* in kernels 2.2 [Gre], the system had a different (although similar) syntax. The 2.0 kernels used a different system called *ipfwadm*. Here (and in later examples) we will only deal with Netfilter/IPtables (in other words with the kernel versions 2.4/2.6).

The interface of the IPtables command allows the different tasks to be performed for configuring the rules that affect the filter system: whether the generation of logs, pre and post package routing actions, NAT, and port forwarding.

Service start up with: `/etc/init.d/iptables start`, if not already configured in the runlevel.

The `iptables -L` command lists the active rules at that time in each of the chains. If not previously configured, by default they tend to accept all the packages of the chains of input output and forward.

The IPtables system has the tables as a superior level. Each one contains different chains, which in turn contain different rules. The three tables that we have are: Filter, NAT and Mangled. The first is for the filtering norms themselves, the second is to translate addresses within a system that uses NAT and the third, less frequently used, serves to specify some package control options and how to manage them. Specifically, if we have a system directly connected to the Internet, we will generally only use the Filter table. If the system is on a private network that has to pass through a router, gateway or proxy (or a combination of them), we will almost certainly have a NAT or IP masquerading system; if we are configuring the machine to allow external access,

Web site

Netfilter, ver: <http://www.netfilter.org>
Ipchains, see: <http://www.netfilter.org/ipchains/>

Note

IPtables provides different elements such as the tables, chains and the rules themselves.

we will have to edit the NAT table and the Filter table. If the machine is on a private network system, but is one of the internal machines, it will be enough to edit the Filter table, unless it is a server that translates network addresses to another network segment.

If a package reaches the system, the firewall will first look at whether there are rules in the NAT table, in case addresses towards the internal network need to be translated (addresses are not normally visible outwards); then it will look at the rules in the Filter table in order to decide whether the packages will be allowed to pass or whether they are not for us and we have forward rules to know where to redirect them. On the contrary, when our processes generate packages, the output rules of the Filter table will control whether we allow them out or not, and if there is a NAT system the rules will translate the addresses in order to masquerade them. In the NAT table there are usually two chains: prerouting and postrouting. In the first, the rules have to decide if the package has to be routed and, if so, what the destination address will be. In the second, it is finally decided whether the package is allowed inside or not (to the private network, for example). And there is also an output chain for locally generated outgoing traffic to the private network, since prerouting does not control this (for more details, see iptables man page).

Next we will comment on some aspects and examples of configuring the Filter table (for the other tables, we can consult the associated bibliography).

The Filter table is typically configured as a series of rules that specify what is done inside a particular chain, like the three preceding ones (input, output and forward). Normally, we will specify:

```
iptables -A chain -j target
```

where *chain* is the input, output or forward and *target* is the destination that will be assigned to the packet which corresponds to the rule. Option -A adds the rule to the existing ones. We have to be careful here, because the order does matter. We have to put the least restrictive rules at the beginning, given that, if we put a rule that eliminates the packets at the beginning, even if there is another rule, this will not be taken into account. Option -j can be used to decide what we will do with the packets, typically *accept*, *reject* or *drop*. It is important to note the difference between *reject* and *drop*. With the first, we reject the packet and we will normally inform the sender that we have rejected the connection attempt (normally for an ICMP-type packet). With the second, *drop*, we simply "lose" the package as though it had never existed and we will not send any form of response. Another *target* that is used is *log*, to send the packet to the log system. Normally, in this case, there are two rules, one with the *log* and another identical one with *accept*, *drop* and *reject*, so that the information on the accepted, rejected or dropped packets can be sent to the log.

When entering the rule, we can also use the option `-I` (insert) to indicate a position, for example:

```
iptables -I INPUT 3 -s 10.0.0.0/8 -j ACCEPT
```

which tells us that the rule should be put in the third position in the *input* chain; and that packets (-j) that come from (with *source*, -s) from the subnet 10.0.0.0 with netmask 255.0.0.0 will be accepted. With `-D`, similarly, we can delete either a rule number or the exact rule, as specified below, deleting the first rule of the chain or the rule that we mention:

```
iptables -D INPUT 1
iptables -D INPUT -s 10.0.0.0/8 -j ACCEPT
```

There are also rules that can be used to define a default "policy" for the packets (option `-P`); the same thing will be done with all the packets. For example, we would usually decide to drop all the packets by default and then enable the ones that we require; likewise, we would often avoid forwarding packets if it is not necessary (if we do not act from the router), this could be declared as follows:

```
iptables -P INPUT DENY
iptables -P OUTPUT REJECT
iptables -P FORWARD REJECT
```

This establishes default policies that consist of rejecting any incoming packets and not permitting the sending or resending of packets. Now we will be able to add rules that affect the packets that we wish to use, stating which protocols, ports and origins or destinations we wish to permit or avoid. This can be difficult as we have to know all the ports and protocols that our software or services use. Another strategy would be to only leave active the services that are essential and to enable access to the services for the desired machine through the firewall.

Some examples of these rules on the Filter table could be:

```
1) iptables -A INPUT -s 10.0.0.0/8 -d 192.168.1.2 -j DROP
2) iptables -A INPUT -p tcp --dport 113 -j REJECT --reject-with
tcp-reset
3) iptables -I INPUT -p tcp --dport 113 -s 10.0.0.0/8 -j ACCEPT
```

where:

- 1) We drop the packets that come from 10.x.x.x sent to 192.168.1.2.
- 2) We reject the tcp packets sent to port 113, issuing a tcp-reset type response.

3) The same packets as in 2) but that come from 10.x.x.x will be accepted.

With regard to the names of the protocols and ports, the iptables system uses the information provided by the files `/etc/services` and `/etc/protocols`, and we can specify the information (port or protocol) either with numbers or with the names (we must make sure, in this case, that the information on the files is correct and that it has not been modified, for example, by an attacker).

The configuration of the iptables is usually established through consecutive calls to the iptables command with the rules. This creates a state of active rules that can be consulted with `iptables -L`; if we wish to save them so that they are permanent, we can do this in Fedora with:

```
/etc/init.d/iptables save
```

And they are saved in:

```
/etc/sysconfig/iptables
```

In Debian, we can execute:

```
/etc/init.d/iptables save name-rules
```

We have to be careful and ensure that the directory `/var/log/iptables` already exists, as this is where the files will be saved; `name-rules` will be a file in the directory.

With `(/etc/init.d/iptables load)` we can load the rules (in Debian, we have to provide the name of the rules file), although Debian supports some default file names, which are active for the normal rules (the ones that will be used when the service starts) and inactive for the ones that will remain when the service is deactivated (or stopped). Another similar method that is commonly used is that of putting the rules in a script file with the iptables calls that are necessary and calling them, for example, by putting them in the necessary runlevel, or with a link to the script in `/etc/init.d`.

7.3. Packets of firewalls in the distributions

With regard to the configuration tools that are more or less automatic in the firewall, there are various possibilities, but we should remember that they do not usually offer the same features as the manual configuration of iptables (which in most cases, is the recommended process). Some tools are:

- **lokkit:** in Fedora/Red Hat, on a very basic level, the user can only choose the desired security level (high, medium or low). Afterwards, the services that would be affected are shown and we can leave it, or not, so that we pass on to the service changing the default configuration. The mechanism

used beneath is the iptables. The final configuration of the rules that is made can be seen at `/etc/sysconfig/iptables` which, in turn, is read by the iptables service, which is loaded on boot up or when stopping or booting using `/etc/init.d/iptables` with the start or stop options. It is also possible to install it in Debian, but the rules configuration should be left in `/etc/defaults/lokkit-l` and a script in `/etc/init.d/lokkit-l`. There is also a graphic version called `gnome-lokkit`.

- **Bastille** [Proa]: this is a fairly complete and educational security program that explains different recommended security settings and how we can apply them step by step; it also explains the configuration of the firewall (the program is interactive). It works in various distributions, including in both Fedora and Debian.
- **fwbuilder**: a tool that can be used to build the rules of the firewall using a graphical interface. It can be used in various operating systems (GNU/Linux, both Fedora and Debian, OpenBSD, MacOS), with different types of firewalls (including iptables).
- **firestarter**: a graphical tool (Gnome) for creating a firewall. It is very complete, practically managing all the possibilities of the iptables, but, likewise, it has assistants that make it easy to set up a firewall intuitively. Likewise, there is a real-time monitor for detecting any intrusions.

Web site

See: <http://www.fwbuilder.org/>

Normally, each of these packets uses a rules system that is saved in its own configuration file and that usually starts up as a service or as a script execution in the default runlevel.

7.4. Final considerations

Even if we have well-configured firewalls, we have to remember that they are not an absolute security protection, as there are complex attacks that can pass over the firewalls or falsify the data to create confusion. In addition, modern connectivity sometimes needs to force us to create software that will bypass the firewalls:

Note

We should never rely on one single mechanism or security system. The security of the system must be established at all the different levels.

- Technologies, such as IPP, the printing protocol used by CUPS, or WebDAV, the authoring and versioning protocol for websites, make it possible to bypass (or make it necessary to bypass) the configurations of the firewalls.
- A technique called tunnelling is often used (for example, with the above-mentioned protocols and others). This technique basically encapsulates the non-permitted protocols, on the basis of others that are permitted; for example, if a firewall only permits HTTP traffic to pass (port 80 by default), it is possible to write a client and server (each one on one side of the firewall) that can speak in any protocol known to both, but in which

the network is transformed into a standard HTTP, which means that the traffic can bypass the firewall.

- The mobile codes by web (ActiveX, Java, y JavaScript) bypass the firewalls, and it is therefore difficult to protect the systems if these are vulnerable to attacks against any open holes that are discovered.

Therefore, although firewalls are a very good solution for most security-related aspects, they can always have vulnerabilities and let traffic that is considered valid through, which then includes other possible sources of attack or vulnerabilities. With regard to security, we should never consider (and rely on) only one single solution and expect it to protect us from everything; it is necessary to examine the various problems, to propose solutions that will detect any problems on time and to establish prevention policies that will protect the system before any harm is done.

8. Security tools

Some of these tools can also be considered tools for attacking other machines. Therefore, it is advisable to test these tools on machines in our own local or private network; we should never do this with third party IPs, as these could interpret the tests as intrusions and we or our ISP may be held responsible for them and the corresponding authorities may be notified to investigate us and remove our access.

We will now briefly discuss some tools and the ways in which they can be used:

a) TripWire: this tool maintains a database of sums for checking the important files in the system.

It may serve as a preventive IDS system. We can use it to "take" a snapshot of the system, so that we can subsequently check any modification made and that it has not been corrupted by an attacker. The aim here is to protect the files in the machine itself and to avoid any changes occurring, such as those that, for example, the rootkit might have caused. Therefore, when we execute the tool again, we can check all the changes compared to the previous execution. We have to choose a subset of files that are important in the system or possible sources of attack. TripWire is proprietary, but there is a free open-source tool that is the equivalent called AIDE.

b) Nmap [Insb]: this is a tool that scans ports in large networks. It can scan from individual machines to network segments. It provides various scanning modes, depending on the system's protections. It also provides techniques with which we can determine the operating system used by remote machines. Different TCP and UDP packets may be used to test the connections. There is a graphical interface known as xnmmap.

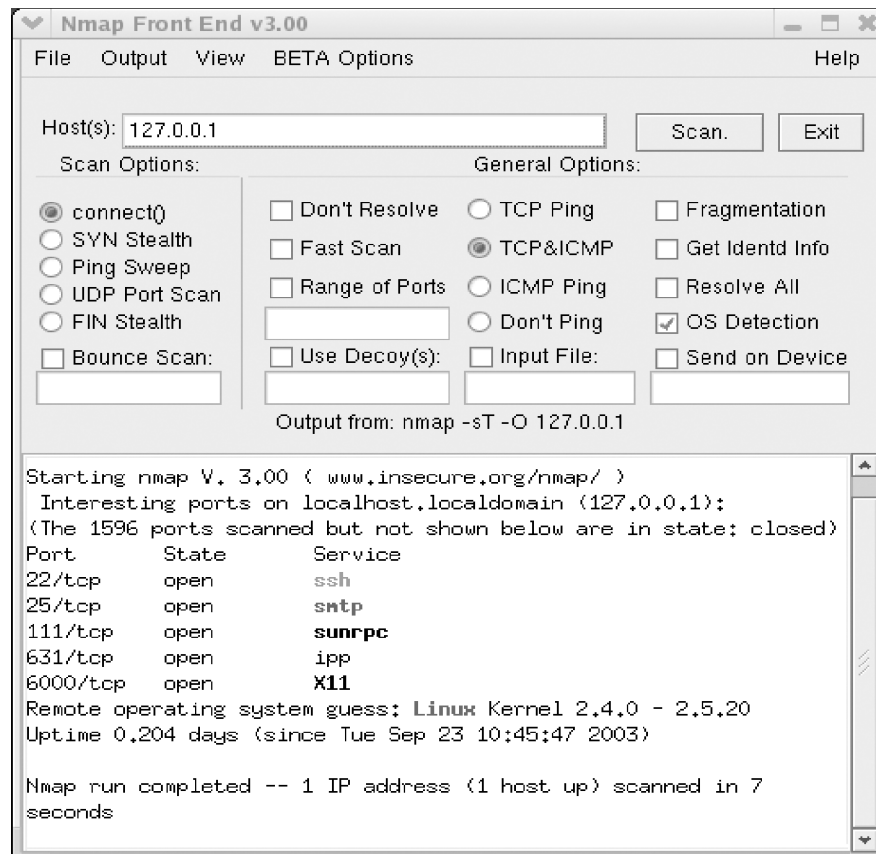


Figure 2. xnmmap analysing the local services

c) **Wireshark** [Wir] (previously called **Ethereal**): is a protocol analyser that captures the traffic in the network (it acts as a sniffer). It can be used to visualise the captured traffic, see the statistics and data of the individual packets and group the packets, either by origin, destination, ports or protocol. It can even reconstruct the traffic from a whole session from a Transmission Control Protocol (TCP).

d) **Snort** [Sno]: is an IDS system that makes it possible to analyse the traffic in real time and save logs of the messages. It can be used to analyse the protocols and search by patterns (protocol, origin, destination etc.). It can be used to detect various types of attack. Basically, it analyses the traffic in the network to detect patterns that might correspond to an attack. The system uses a series of rules to either produce a log of the situation (log) or warn the user (alert) or reject the information (drop).

e) **Nessus** [Nes]: detects any known vulnerabilities (by testing different intrusion techniques) and assesses the best security options for those discovered. It is a modular program that includes a series of plugins (more than 11,000) for performing the different analyses. It uses a client-server architecture, with a graphic client to show the results and the server, which carries out different tests on the machines. It has the capacity to examine whole networks. It

generates reports on the results, which can be exported to different formats (HTML, for example). Up until 2005, Nessus 2 was a free tool, but the company decided to make it proprietary, in version Nessus 3. In GNU/Linux, Nessus 2 is still used, as it continues to have a GPL license and a series of plugins, which are gradually updated. Nessus 3, as a proprietary tool for GNU/Linux, is more powerful and widely used, as it is one of the most popular security tools and there is normally a free version available with plugins that are less updated than the ones in the version that is not free.

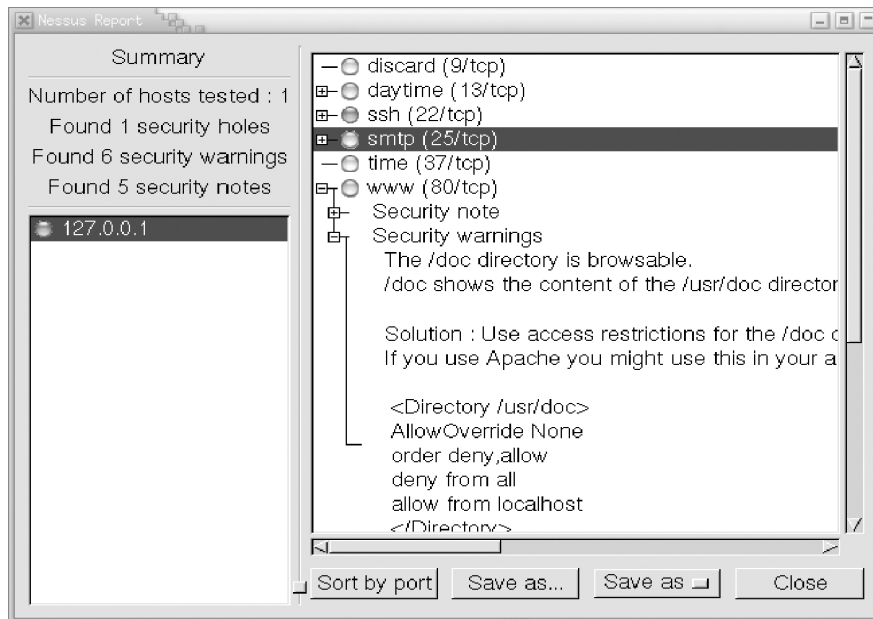


Figure 3. Nessus client showing the vulnerabilities report and the possible solutions

We can find many other security tools that are available. A good place to start is <http://sectools.org>, where the designers of Nmap maintain a list of popular tools, as voted by the users (now, a bit older list, but useful tools).

9. Logs analysis

By observing the log files [Ano99][Fri02], we can quickly get an idea of the global state of the system, as well as the latest events, and detect any irregular intrusions (or intrusion attempts). But it should also be remembered that, if there really has been an intrusion, the logs may have been cleaned or falsified. Most of the log files will be in the `/var/log` directory.

Many of the services may have their own logs, which are normally established during configuration (through the corresponding configuration file). Most of them usually use the log facilities incorporated in the Syslog through the Syslogd daemon. The configuration will be in `/etc/syslog.conf`. This configuration is usually established according to the message levels: there are different types of message according to their importance. Normally, levels such as debug, info, err, notice, warning, err, crit, alert, emerg, appear, in which the order of importance of the messages would be more or less as follows (from least to most important). Normally, most of the messages are sent to the `/var/log/messages` log, but the system can be set so that each message type goes to different files and it is also possible to identify who has created them; typically, the kernel, mail, news, the authentication system etc.

Consequently, it is appropriate to examine (or in any case adapt) the configuration of Syslog so as to determine the logs in which we can find / generate the information. Another important point is to control its growth, as, depending on which are active and the operations (and services) that are performed in the system, the logs can grow very quickly. In Debian and Fedora, this can be controlled through logrotate, a daemon that regularly makes copies and compresses the oldest logs; it is possible to find the general configuration in `/etc/logrotate.conf`, although some applications set specific configurations that can be found in the `/etc/logrotate.d` directory.

In the following points, we will discuss some of the log files that should be taken into account (perhaps the most frequently used):

a) `/var/log/messages`: is the default log file of the Syslogd daemon, but we would have to check its configuration, in case it has been moved to another place or there are several of them. This file contains a wide range of messages from various origins (different daemons, services or the same kernel); anything that looks irregular must be verified. If there has been an intrusion, the date of the intrusion and related files should be checked.

b) `/var/log/utmp`: this file contains binary information for each user that is currently active. It is useful to determine who is logged in the system. The `who` command uses this file to provide this information.

c) `/var/log/wtmp`: each time that a user logs in or out of the system, or the machine reboots, an entry is saved in this file. This is a binary file from which the `last` command obtains the information; the file records which users logged in or out of the system and when and where the connection was made. It can be useful for finding out where (in which accounts) the intrusion started and detect the use of suspicious accounts. There is also a variation in the command called `lastb`, which lists the login attempts that were not correctly validated and the `/var/log/btmp` file is used (you may have to create it if it doesn't exist). These same authentication faults can also be sent to `log auth.log`. In a similar manner, the `lastlog` command uses another file, `/var/loglastlog`, to verify which was the last connection of each of the users.

d) `/var/log/secure`: they are usually used in Fedora for sending the *tcp wrapper* messages (or firewalls). Each time that a connection is established to an `inetd` service, or, in the case of Red Hat 9, to the `xinetd` service (with its own security), a log message is added to this file. We can search for intrusion attempts in services that are not usually used or in unfamiliar machines that try to connect.

In the logs system, another thing that should be checked is that the directory logs in `/var/log` can only be writable by the root (or the daemons associated to the services). Otherwise, any attacker could falsify the information in the logs. Nevertheless, if attackers manage to access the root, they may often delete all their tracks.

10. Tutorial: Tools for security analysis

We will now perform some of the processes described above on a Debian system, to improve the security configuration.

First we will examine what our machine offers the network. In order to do this, we will use the nmap tool as a port scanner. With the command (from the root):

```
nmap -sTU -O localhost
```

we obtain:

```
root@machine:~# nmap -sUT -O localhost
starting nmap 3.27 (www.insecure.org/nmap/) at 2003-09-17
11:31 CEST Interesting ports on localhost (127.0.0.1):
```

(The 3079 ports scanned but not shown below are in state: closed)

Port	State	Service
9/tcp	open	discard
9/udp	open	discard
13/tcp	Open	daytime
22/tcp	Open	smtp
25/tcp	open	time
37/tcp	open	time
37/udp	open	http
80/tcp	open	sunrpc
111/tcp	open	sunrpc
111/udp	open	auth
113/tcp	open	ipp
631/tcp	open	unknown
728/udp	open	
731/udp	open	netviewdm3
734/tcp	open	unknown

```
Remote operating system guess: Linux kernel 2.4.0-2.5.20
```

```
Uptime 2.011 days (since Mon Sep 15 11:14:57 2003)
Nmap run completed --1 IP address (1 host up) scanned in
9.404 seconds
```

We can see that a high number of open services have been detected (depending on the machine, there may be more: telnet, FTP, finger...), in both transmission control protocol (TCP) and user datagram protocol (UDP). Some services, such as discard, daytime, time may be useful on occasion, but they should not normally be open to the network, as they are considered non-secure. SMTP is the resending and routing service, for mail; if we are acting as the host or mail server, this would have to be active; but if we are only reading and writing emails through POP3 or IMAP accounts, this doesn't necessarily have to be active.

Another method for detecting active services would be by searching active listening ports, which can be achieved with *netstat -lut* command.

The *nmap* command can also be applied with the DNS or IP name of the machine; this shows us how the system looks from the exterior (with localhost, we see what the actual machine can see), or, better still, we could even use a machine of an external network (for example, any PC connected to the Internet) to examine what could be seen in our machine from outside.

We will now go to */etc/inetd.conf* to deactivate these services. We should look for lines such as:

```
discard stream tcp nowait root internal
smtp stream tcp nowait mail /usr/sbin/exim exim -bs
```

and we type a number symbol (#) at the beginning of the line (only in the services that we wish to deactivate and when we know what they are really doing (check pages of man as deactivating them has been recommended). Another case of recommended deactivation would be that of the ftp, telnet, finger services and we should use ssh to replace them.

Now we have to reboot inetd so that it rereads the configuration that we have changed: */etc/init.d/inetd restart*.

We return to nmap:

22/tcp	open	ssh
80/tcp	open	http
111/tcp	open	sunrpc
111/udp	open	sunrpc

113/tcp	open	auth
631/tcp	open	ipp
728/udp	open	unknown
734/tcp	open	unknown

From what is left, we have the ssh service, which we wish to leave active, and the web server, which we will stop for the moment:

```
/etc/init.d/apache stop
```

ipp is the printing service associated to CUPS. In the local administration section we saw that there was a CUPS web interface that connected to port 631. If we wish to have an idea of what a specific port is doing, we can look in `/etc/services`:

```
root@machine:~# grep 631 /etc/services
ipp 631/tcp          # Internet Printing Protocol
ipp 631/udp          # Internet Printing Protocol
```

If we are not acting as the printing server to the exterior, we have to go to the CUPS configuration and eliminate this feature (for example, by placing a *listen* 127.0.0.1:631, so that only the local machine listens), or limit the access to the permitted machines.

Some other ports also appear as unknown, in this case, ports 728 and 734; this indicates that the system has not been able to determine which nmap is associated to the port. We will try to see it ourselves. For this, we can execute the netstat command on the system, which offers different statistics on the network system, from the packets sent and received and errors to the elements in which we are interested, which are the open connections and who is using them. We will try to find out who is using the unknown ports:

```
root@machine:~# netstat -anp | grep 728
udp 0 0 0.0.0.0:728 0.0.0.0:* 552/rpc.statd
```

And if we do the same with port 734, we can see that it was rpc.statd that opened the port; rpc.statd is a daemon associated to NFS (in this case, the system has an NFS server). If we repeat this process with ports 111, which appeared as sunrpc, we will see that the daemon that is behind is portmap, which is used in the remote procedure call system (RPC). The RPC system permits users to use the remote calls between two processes that are on different machines. portmap is a daemon that converts the calls that arrive at the port to the internal RPC services numbers and it is used by different servers such as NFS, NIS, NIS+.

The RPC services offered can be seen with the `rpcinfo` command:

```
root@machine:~# rpcinfo -p
```

programme vers	proto	Port
100000 2 tcp	111	portmapper
100000 2 udp	111	portmapper
100024 1 udp	731	status
100024 1 tcp	734	status
391002 1 tcp	39797	sgi_fam
391002 2 tcp	39797	sgi_fam

where we see the RPC services with some of the ports that had already been detected. Another command that may be useful is `lsof`, which, among other functions, makes it possible to relate ports with the services that have opened them (for example: `lsof -i | grep 731`).

The portmap daemon is somewhat critical with regard to security, as, in principle, it does not offer the client authentication mechanisms, as this is supposedly delegated to the service (NFS, NIS...). Consequently, portmap could be subjected to DoS attacks that could cause faults in the services or cause downtime. We usually protect portmap using some kind of wrapper and/or firewall. If we do not use these and we do not intend to use the NFS and NIS services, the best thing to do is to completely deactivate portmap, removing it from the runlevel on which it activates. We can also stop them momentarily with the following scripts (in Debian):

```
/etc/init.d/nfs-common
/etc/init.d/nfs-kernel-server
/etc/init.d/portmap
```

by entering the `stop` parameter to stop the RPC services (in this case NFS).

We will then control the security in the base using a simple wrapper. Let us suppose that we wish to let a specific machine pass through ssh, which we will call 1.2.3.4 (IP address). We will close portmap to the exterior, as we do not have NIS and we have an NFS server but we are not serving anything (we could close it, but we will leave it for future use). We will create a wrapper (we are assuming that the TCP wrappers are already installed) modifying the files `hosts.deny` -j allow. In `/etc/hosts.deny`:

```
ALL : ALL : spawn (/usr/sbin/safe_finger -l @%h \
| /usr/bin/mail -s "%c FAILED ACCESS TO %d!!" root) &
```

we are denying all of the services (be careful, some of them are related to inetd) (primer all), and the next step to take will be to find out who has requested the service and from what machine and we will send an email message to the root user reporting the attempt. We could also write a log file... Now, in /etc/hosts.allow:

```
sshd: 1.2.3.4
```

we enable access for the IP 1.2.3.4 machine in the sshd server (of the ssh). We could also enter the access to portmap, all we would need is a portmap line: la_ip. We can enter a list of machines or subnets that can use the service (see man hosts.allow). Remember that we also have the tcpdchk command to check that the configuration of the wrapper is correct and the tcpdmatch command to simulate what would happen with a specific attempt, for example:

```
root@machine:~# tcpdmatch sshd 1.2.3.4

warning: sshd: no such process name in /etc/inetd.conf client:
hostname machine.domain.es
client: address 1.2.3.4
server: process sshd
matched: /etc/hosts.allow line 13
access: grantedv
```

tells us that access would be provided. One detail is that it tells us that sshd is not in inetd.conf and, if we verify it, we see that it is not: it is not activated by the inetd server, but by the daemon in the runlevel on which we are operating. Besides, in Debian, this is a daemon that is compiled with the included wrapper libraries (which does not therefore require tcpd to work). In Debian, there are various daemons such as: ssh, portmap, in.talk, rpc.statd, rpc.mountd, among others. This allows us to secure these daemons using wrappers.

Another question that should be verified concerns the existing current connections. With the netstat -utp command, we can list the tcp, udp connections established with the exterior, whether they are incoming or outgoing; therefore, at any time, we can detect the connected clients and who we are connected to. Another important command (with multiple functions) is lsof, which can relate open files with established processes or connections on the network through lsof -i, which helps to detect any inappropriate accesses to files.

We could also use a firewall for similar processes (or as an added mechanism). We will begin by seeing how the rules of the firewall are at this time: (iptables -L command)

```
root@aopcjj:&#732;# iptables -L
```

```
Chain INPUT (policy ACCEPT)
target prot opt source      destination
Chain FORWARD (policy ACCEPT)
target prot opt source      destination
Chain OUTPUT (policy ACCEPT)
target prot opt source      destination
```

In other words, the firewall is not placing any restrictions at this time and all the packets can be sent, received and resent.

At this point, we could add a firewall that would provide better management of the packets that we receive and send, and which would be a preliminary control for improving security. Depending on our needs, we would establish the necessary rules similarly to the firewall examples provided in this unit.

If we set up the firewalls, we can consider whether to use this mechanism as a single security element and remove the wrappers: this could be done, because the firewalls (in this case through iptables) offer a very powerful feature that allows us to follow up a packet by type, protocol and by what it is doing in the system. A good firewall could be enough, but, to be on the safe side, more security measures will always be helpful. And if the firewall were not well designed and let some packets escape, the wrapper would be the level-service measure for stopping any non-desired accesses. To offer a metaphor that is often used, if we consider our system to be like a medieval castle that must be defended, the moat and the front walls would be the firewall, and the second containment wall would be the wrapper.

Activities

- 1) Suppose that we locate a website in our machine, using Apache for example. Our site is designed for ten internal users, but we do not control this number. Subsequently, we consider making this system accessible on the Internet, as we think that it could be useful for the clients, and the only thing we have to do is assign a public IP address on the Internet to the system. What types of attack might this system suffer?
- 2) How can we detect the files with suid in our system? Which commands are necessary? And the directories with SUID or SGID? Why is it necessary, for example, for /usr/bin/passwd to have a SUID bit?
- 3) The .rhosts files, as we have seen, are a significant danger for security. Could we use some type of automatic method for regularly checking these files? How?
- 4) Let us suppose that we want to disable a service that we know has its /etc/init.d/service script that controls it: we wish to disable it in all the runlevels in which it appears. How do we find the runlevels in which it is present? (for example, searching for links to the script).
- 5) Examine the active services in your machine. Are they all necessary? How would we protect or deactivate them?
- 6) Practice using some of the described security tools (nmap, nessus etc.).
- 7) Which IPtables rules would be necessary for a machine that we only wish to access through SSH from a specific address?
- 8) What if we only want to access the web server?

Bibliography

Other sources of reference and information

[Deb] [Hatc] The security sites for the distributions.

[Peñ] Essential for Debian, with a very good description of how to configure security, that can be followed step by step, [Hatb] would be the equivalent for Fedora/Red Hat.

[Mou01] Excellent security reference for Red Hat (also applicable to Debian).

[Hat01] GNU/Linux security books covering extensive techniques and aspects.

[Line] Small guide (2 pages) to security.

[Sei] Step-by-step guide identifying the key points that have to be verified and the problems that may arise.

[Net] Project Netfilter, and IPTables.

[Ian] A list of TCP/IP ports.

[Proa] [Sno] [Insb] [Nes] Some of the most commonly used security tools.

[NSAb] Linux version focused on security, produced by the NSA. Reference for SELinux.

[CERa][Aus][Insa][Incb] [NSAa] Security organisations' sites.

[CERb][Ins][San] Vulnerabilities and exploits of the different operating systems.

[NSAa][FBI][USA] Some cybercrime "policies" in the United States.

Configuration, tuning and optimisation

Remo Suppi Boldrito

PID_00148473

Index

Introduction.....	5
1. Basic aspects.....	7
1.1. Monitoring on a UNIX System V	8
1.2. Optimising the system	15
1.3. General optimisations	19
1.4. Additional configurations	20
1.5. Monitoring	24
Activities.....	33
Bibliography.....	34

Introduction

A fundamental aspect, once the system has been installed, is the configuration and adjustment of the system to the user's needs to ensure that the features are as adequate as possible for the demands that will be placed on it. GNU/Linux is an efficient operating system that provides an excellent degree of possible configurations and a very delicate optimisation that can be tailored to the needs of the user. This is why, once the system has been installed (or updated, depending on the case), certain configurations that are essential to the system must be tuned. Although the system may "work", it is necessary to make some changes (adapting to the environment or tuning) so that all the needs of the users/services that the machine must provide are met. This tuning will depend on where the machine is working; the tuning will be carried out, in some cases, in order to improve the system's performance and efficiency, and, in other cases (in addition), for security reasons (see module 9, "Security administrator"). When the system is working, it is necessary to monitor the system to see how it performs and behaves and to act accordingly. Although it is a fundamental aspect, tuning an operating system is often left to the opinions of computer experts or gurus; but if we are aware of the parameters that affect the performance, it is possible to achieve good solutions by undertaking a cyclical process of analysis, making changes to the configuration, monitoring and making adjustments.

1. Basic aspects

Before learning about the optimisation techniques, it is necessary to list the causes that might affect the performance of an operating system [Maj96]. Among these, we might mention:

- a) Bottlenecks in the resources: the consequence is that the whole system will be slower because there are resources that cannot satisfy the demand to which they are being subjected. The first step for optimising the system is to find these bottlenecks and their causes, whilst learning about their theoretical and practical limitations.
- b) Amdahl's law; according to this law, "there is a limit to how much an overall system can be improved (or speeded-up) when only one part of the system is improved"; in other words, if we have a program that uses 10% of the CPU and it is optimised to reduce the use by a factor of 2, the program will improve its performance (speedup) by 5%, which means that a tremendous amount of effort is put into something that is not compensated by the ensuing results.
- c) Estimates of the speedup: it is necessary to estimate how much the system will improve so as to avoid any unnecessary efforts and costs. We can use the previously described law to evaluate whether it is necessary to invest time or money in the system.
- d) Bubble effect: it is very common to have the feeling that, once we have solved a problem, another one always appears. A manifestation of this problem is that the system is constantly moving between CPU problems and in/out problems, and vice versa.
- e) Response time in respect of workload: if we have twenty users, improving the productivity will mean that all will get more work done at the same time, but the individual response times will not improve; it may be that the response times for some will be better than for others. Improving the response times means optimising the system so that the individual tasks take as little time as possible.
- f) User psychology: two parameters are fundamental: 1) the user will be generally unsatisfied when there are variations in the response time; and 2) the user will not notice any improvements in execution times of less than 20%.
- g) Test effect: the monitoring measures affect the measures themselves. We should proceed carefully when we are performing tests because of the collateral effects of the actual testing programs.

h) Importance of the average and variation: the results should be taken into account, given that, if we obtain an average of CPU usage of 50% when only 100, 0, 0, 100 has been used, we could come to the wrong conclusions. It is important to see the variation on the average.

i) Basic knowledge on the hardware of the system that will be optimised: to improve something we need to "know" whether it can be improved. The person in charge of optimisation must have a lot of basic knowledge about the underlying hardware (CPU, memory, buses, cache, in/out, disks, video...) and the interconnections in order to determine where the problems lie.

j) Basic knowledge of the operating system that is to be optimised: as with the preceding point, the user must know the minimum aspects of the operating system that they intend to optimise, which would include concepts such as processes and threads (creation, execution, states, priorities, termination), system calls, cache buffers, file system, administration of memory and virtual memory (paging, swap) and tables of the kernel.

Note

When optimising, the saturation of resources must be considered. Amdahl's law lists the knowledge of the software and hardware available, the response times and the number of jobs.

1.1. Monitoring on a UNIX System V

/proc will appear as a directory but in reality, it is a fictitious file system, in other words, it does not exist on the disk and the kernel creates it in the memory. This is used to provide information on the system (originally on the processes, hence the name), which will later be used by the commands that we will now examine. We will now look at some interesting files (check the relevant page on the manual for more information):

/proc/1: a directory with the information on process 1 (the number of directories is the PID of the process).

/proc/cpuinfo: information on the CPU (type, brand, model, performance...).

/proc/devices: list of devices configured in the kernel.

/proc/dma: DMA channels used at this point in time.

/proc/filesystems: file systems configured in the kernel.

/proc/interrupts: shows which interruptions are in use and how many of them have been processed.

/proc/ioports: the same applies to the ports.

/proc/kcore: image of the physical memory of the system.

/proc/kmsg: messages generated by the kernel which are then sent to syslog.

/proc/ksyms: table of kernel symbols.

/proc/loadavg: system load.

/proc/meminfo: information on memory use.

/proc/modules: modules loaded by the kernel.

/proc/net: information on the network protocols.

/proc/stat: statistics on the system.

/proc/uptime: from when the system is working.

/proc/version: version of the kernel.

It should be remembered that these files are visible (text) but sometimes the data are in a "raw" state and commands are necessary to interpret them. These commands will be the ones that we will now examine.

The compatible UNIX SV systems use the `sar` and `sadc` commands to obtain system statistics (in FC included inside the `sysstat` package that also includes `iostat` or `mpstat`). The equivalent in GNU/Linux Debian is `atsar` (and `atsadc`), which is the absolute equivalent to the one we have mentioned. The `atsar` command reads counters and statistics on the `/proc` file and shows them at the standard output. The first way of calling the command is:

```
atsar options t [n]n
```

where the activity is shown in n times every t seconds with a header showing the activity counters (the default value of n is 1). The second way of calling it is:

```
atsar -options -s time -e time -i sec -f file -n day#
```

The command extracts data from the file specified by `-f` (by default `/var/log/atsar/atsarxx`, with `xx` being the day of the month) and that were previously saved by `atsadc` (this is used to collect data, save them and process them and, in Debian, it is in `/usr/lib/atsar`). The parameter `-n` can be used to indicate the day of the month and `-s`, `-e` the time of final-boot, respectively. To activate `atsadc`, for example, we could include a line such as the following in `/etc/cron.d/atsar`:

```
@reboot root test -x /usr/lib/atsadc && /usr/lib/atsar/atsadc  
/var/log/atsar/atsa'date +%d'
```

```
10,20,30,40,50 * * * * root test -x /usr/lib/atsar/atsal &&
/usr/lib/atsar/atsal
```

The 1st creates the file after a reboot. The 2nd saves the data every 10 minutes with the shell script *atsal*, which calls *atsadc*.

In *atsar* (or *sar*), the options are used to indicate which counters have to be shown; some examples include:

Option	Description
u	CPU Use
d	Disk activity
l (i)	Number of interruption/s
v	Use of tables in the kernel
and	Use statistics of <i>ttys</i>
p	Information on paging and <i>swap activity</i>
r	Free memory and used-up <i>swap</i>
l (L)	Network statistics
L	Network errors information
w	IP connection statistics
t	TCP statistics
U	UDP statistics
m	ICMP statistics
N	NFS statistics
A	All options

Note

Monitoring with *atsar*

- CPU: *atsar -u*
- Memory: *atsar -r*
- Disk: *atsar -d*
- Paging: *atsar -p*

Between *atsar* and *sar* there are only a few differences in terms of how the data are shown and *sar* includes a few additional (or different) options. We will now see some examples of how to use *sar* (exactly the same as with *atsar*, the only differences are in the way in which the data are displayed) and the meaning of the information that generates:

1) CPU use


```
sar -u 4 5
```

```
Linux 2.6.19-prep (localhost.localdomain) 24/03/07
```

08:23:22	CPU	%user	%nice	%system	%iowait	%steal	%idle
08:23:26	all	0.25	0.00	0.50	0.00	0.00	99.25
08:23:30	all	0.00	0.00	0.00	0.00	0.00	100.00
08:23:34	all	0.00	0.00	0.00	0.00	0.00	100.00
08:23:38	all	0.25	0.00	0.00	0.00	0.00	99.75
08:23:42	all	0.00	0.00	0.00	0.00	0.00	100.00
Media:	all	0.10	0.00	0.10	0.00	0.00	99.80

- `usr` and `system` show the percentage of CPU time in the user mode with `nice = 0` (normal) and in the kernel mode.
- `nice` is the same but with user processes with `nice > 0` (lower than average priority).
- `idle` indicates the CPU time not used by the processes in standby mode (does not include disk standby).
- `iowait` is the time that the CPU has been free when the system was entering or exiting a disk.
- `steal` is the time wasted uselessly whilst waiting for a virtual CPU (valid in virtualised environments).

In this case `idle=100`, which means that the CPU is idle, which means that there are no processes to execute and the workload is low; if `idle=10` and there are a high number of processes, the optimisation of the CPU should be considered, as it could be a bottleneck in the system.

2) Number of interruptions per second

```
sar -I 4 5
```

```
Linux 2.6.19-prep (localhost.localdomain) 24/03/07
08:24:01 INTR intr/s
08:24:06 4 0.00
Media: 4 0.00
```

Shows the information on the frequency of interruptions of the active levels located in `/proc/interrupts`. This is useful to see if there is any device that is constantly interrupting the CPU's work.

3) Memory and swap

```
sar -r 4 5
```

```
Linux 2.6.19-prep (localhost.localdomain) 24/03/07
```

```
08:24:20 kbmemfree kbmemused %memused kbbuffers kbcached kbswpfree kbswpused %swpused kbswpcad
```

```
08:24:24 296516 729700 71.11 24260 459972 963860 0 0.00 0
```

```
08:24:28 296500 729716 71.11 24268 459968 963860 0 0.00 0
```

```
08:24:32 296516 729700 71.11 24268 459976 963860 0 0.00 0
```

```
08:24:36 296516 729700 71.11 24276 459976 963860 0 0.00 0
```

```
08:24:40 296500 729716 71.11 24276 459976 963860 0 0.00 0
```

```
Media: 296510 729706 71.11 24270 459974 963860 0 0.00 0
```

In this case, kbmemfree is the main free memory (MP); used is the used one, buffers is the MP used in buffers; cached is the main memory used in the pages cache; swpfree/used the free/occupied swap space. It is important to remember that if there is no space in MP, the process pages will end up in the swap, where there should be space. This should be compared with CPU use. We can also check that the size of the buffers is appropriate and is so in relation to the processes that are performing I/O operations.

It is also interesting to examine the free command (fc), as it allows us to see the amount of memory in a simplified representation:

```
total used free shared buffers cached
Mem: 1026216 729716 296500 0 24324 459980
-/+ buffers/cache: 245412 780804
Swap: 963860 0 963860
```

This indicates that almost $\frac{3}{4}$ of the 1 Gb memory is occupied and that almost $\frac{1}{2}$ is cache. Plus, it tells us that the swap is not being used for anything, which means that we can conclude that the system is well. If we wish to see more details, we must use the vmstat command (or sar -r) to analyse what is causing the problems or who is consuming that much memory. The following is an output from vmstat 1 10:

```
procs -----memory----- --swap-- -----io----- --system-- -----cpu-----
 r b swpd free buff cache si so bi bo in cs us sy id wa st
0 0 0 295896 24384 459984 0 0 321 56 1249 724 11 2 81 5 0
0 0 0 295896 24384 459984 0 0 0 28 1179 383 1 0 99 0 0
0 0 0 295896 24384 460012 0 0 0 0 1260 498 0 0 100 0 0
0 0 0 295896 24384 460012 0 0 0 0 1175 342 0 0 100 0
0 0 0 295896 24384 460012 0 0 0 0 1275 526 0 0 100 0 0
1 0 0 295896 24392 460004 0 0 0 72 1176 356 0 0 99 1 0
0 0 0 295896 24392 460012 0 0 0 0 1218 420 0 0 100 0 0
```

```

0 0 0 295896 24392 460012 0 0 0 0 1216 436 0 0 100 0 0
0 0 0 295896 24392 460012 0 0 0 0 1174 361 0 0 100 0 0
1 0 0 295896 24392 460012 0 0 0 0 1260 492 0 0 100 0 0

```

4) Use of the tables of the kernel

```
sar -v 4 5
```

Linux 2.6.19-prep (localhost.localdomain)					24/03/07				
08:24:48	dentunusd	file-sz	inode-sz	super-sz	%super-sz	dquot-sz	%dquot-sz	rtsig-sz	%rtsig-sz
08:24:52	19177	3904	15153	0	0.00	0	0.00 0		0.00
08:24:56	19177	3904	15153	0	0.00	0	0.00 0		0.00
08:25:00	19177	3904	15153	0	0.00	0	0.00 0		0.00
08:25:04	19177	3904	15153	0	0.00	0	0.00 0		0.00
08:25:08	19177	3904	15153	0	0.00	0	0.00 0		0.00
Media:	19177	3904	15153	0	0.00	0	0.00 0		0.00

In this case, `superb-sz` is the current maximum number of superblocks maintained by the kernel, for the mounted file systems; `inode-sz`, the current maximum number of incore-inodes in the kernel necessary, which would be one per disk, at the very least; `file-sz` current maximum number of open files, `dquota-sz` current maximum occupation of quota inputs (for the remaining options, please see `sar` (or `atsar`) man). This monitoring process can be completed with the `ps -edafm` (process status) command and the `top` command, which will show the activity and the status of the processes in the system. The following are two examples of both commands (only some of the lines):

`ps -edafm`

F	S	UID	PID	PPID	C	PRI	NI	AD-DR	SZ	WCHAN	STIME	TTY	TIME	CMD
4	-	root	1	0	0	-	-	-	508	-	08:01?		00:00:00	init [5]
1	-	root	1927	7	0	-	-	-	0	-	08:02?		00:00:00	[kondemand/0]
1	-	rpc	2523	1	0	-	-	-	424	-	08:02?		00:00:00	syslogd -m 0
5	S	rpc	2566	1	0	-	-	-	444	-	08:02?		00:00:00	portmap
5	-	root	-	0	78	0	-	-	-	-	08:02-		00:00:00	-
5		root	2587	1	0	-	-	-	472	-	08:02?		00:00:00	rpc.statd
5	S	root	-	-	0	81		0	-	-	08:02-		00:00:00	-
1	-	root	2620		1	0	-	-	1232	-	08:02?		00:00:00	rpc.idmapd

1	S	root	-	-	0	75		0	-	default	08:02-	00:00:00	-
5	-	root	2804	1	0	-	-	-	1294	-	08:02?	00:00:00	/usr/sbin/sshd
5	S	root	-	-	0	84	0	-	-	-	08:02-	00:00:00	-
5	-	root	2910	1	0	-	-	-	551	-	08:02?	00:00:00	/usr/sbin/atd
5	S	root	-	-	0	84	0	-	-	-	08:02-	00:00:00	-
4	-	root	3066	1	0	-	-	-	407	-	08:02tty1	00:00:00	/sbin/mingetty tty1
4		root	3305	1	0	-	-	-	21636	-	08:03?	00:00:01	nautilus --no-default-win- dow --sm-
4	-	root	3305	1	0	-	-	-	21636	-	08:03?	00:00:01	client-id default3
0	-	root	3643	3541	0	-	-	-	1123	-	08:17pts/ 1	00:00:00	bash
4	-	root	3701	3643	0	-	-	-	1054	-	08:27pts/ 1	00:00:00	ps -edaflm

..

Where the parameters reflect the value indicated in the variable of the kernel for this process; the most important ones from the monitoring perspective are: F flags (in this case 1 is with super privileges, 4 created from the start daemon), S is the status (D: uninterruptible sleep in/out, R: runnable, or run queue, S: Sleeping, T: traced or stopped, Z: a defunct process ('zombie')). PRI is the priority; NI is nice; STIME, the execution start time; TTY, from where it has executed; TIME, the CPU time; CMD, the program that has run and its parameters. If we want to come out and refresh the page (configurable), we can use the top command, which shows the general statistics (processes, statuses, load etc.) and then obtain information on each point, similar to the ps, but updated every 5 seconds by default:

Note

Check the ps command man or the top man for a description of the parameters and the characteristics

```
top - 08:26:52 up 25 min, 2 users, load average: 0.21, 0.25, 0.33
```

```
Tasks: 124 total, 1 running, 123 sleeping, 0 stopped, 0 zombie
```

```
Cpu(s): 10.8%us, 2.1%sy, 0.0%ni, 82.0%id, 4.9%wa, 0.1%hi,
```

```
0.1%si, 0.0%st
```

```
Mem: 1026216k total, 731056k used, 295160k free, 24464k buffers
```

```
Swap: 963860k total, 0k used, 963860k free, 460208k cached
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
3541	root	15	0	42148	14m	981	S	1.9	1.5	0:00.76	gnome-terminal
3695	root	15	0	260	944	1650	R	1.9	0.1	0:00.02	top
1	root	RT	0	2032	680	580	S	0.0	0.1	0:00.85	init

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
2	root	34	0	0	0	0	S	0.0	0.0	0:00.00	migration/0
3	root	RT	19	0	0	0	S	0.0	0.0	0:00.04	ksoftirqd/0
4	root	10	0	0	0	0	S	0.0	0.0	0:00.00	watchdog/0
5	root	16	-5	0	0	0	S	0.0	0.0	0:00.00	events/0
6	root	10	-5	0	0	0	S	0.0	0.0	0:00.00	khelper
7	root	10	-5	0	0	0	S	0.0	0.0	0:00.00	kthread
53	root	11	-5	0	0	0	S	0.0	0.0	0:00.01	kblockd/0
54	root	15	-5	0	0	0	S	0.0	0.0	0:00.00	kacpid
177	root	18	-5	0	0	0	S	0.0	0.0	0:00.00	cqueue/0
178	root	18	-5	0	0	0	S	0.0	0.0	0:00.00	ksuspend_usbd
181	root	10	-5	0	0	0	S	0.0	0.0	0:00.00	khubd
183	root	10	-5	0	0	0	S	0.0	0.0	0:00.01	kseriod
203	root	23	0	0	0	0	S	0.0	0.0	0:00.00	pdflush
204	root	15	0	0	0	0	S	0.0	0.0	0:00.03	pdflush

Debian Linux also includes a whole set of monitoring tools equivalent to `sar`, but which originated in UNIX BSD and have a similar functionality, although from different commands. `vmstat` (CPU statistics, memory and in/out), `io-stat` (disks and CPU statistics), `uptime` (CPU load and general status).

1.2. Optimising the system

We will now look at some recommendations for optimising the system in accordance with the data obtained.

1) Resolving the problems with the main memory

We must ensure that the main memory can handle a high percentage of executing processes, as, otherwise, the operating system may page and go to the swap; but this means that the execution of that process will deteriorate significantly. If we add more memory, the response time will improve significantly. For this, we must take into account the size of the processes (SIZE) is the *R* status and add that which is used by the kernel, which can be obtained with the `dmesg` command, which will show us (or with *free*), for example:

Memory:

```
255048k/262080k available (1423k kernel core, 6644k reserved, 466k data,
240k init, Ok highmem
```

We must then compare this against the physical memory and analyse whether the system is limited by the memory (a lot of paging activity can be seen with `atsar -r` and `-p`).

The solutions for the memory problems are obvious: either we increase the capacity or reduce the demands. Given the current price of memory, it is better to increase its size than to spend lots of hours trying to free up just a few hundred bytes, by deleting, removing, organising or reducing the requirements of the executing processes. The requirements can be reduced by reducing the kernel tables, deleting modules, limiting the maximum number of users, reducing the buffers etc., all of which will downgrade the system (bubble effect) and the performance will be worse (in some cases, the system could be rendered completely inoperative).

Note

Where should we look?

1st Memory
2n CPU
3rd In/Out
4th TCP/IP
5th Kernel

Another aspect that can be reduced is the amount of memory for the users, eliminating any redundant processes and changing the workload. In order to do this, we must monitor the defunct processes (zombie processes) and eliminate them, or those that do not progress in the I/O (knowing whether they are active processes, how much CPU they are using up and whether the 'users want them'). Changing the workload is using the queue planning so that the processes that need a large amount of memory can run when there is little activity (for example, at night, using the `at` command to launch them).

2) Too much CPU usage

Basically, we can get this from the idle time (low values). With `ps` or `top`, we must analyse which processes are the ones that 'devour CPU' and make decisions such as: postponing their execution, stopping them temporarily, changing the priority (less conflictive of all, the priority restart command `PID`), optimise the program (for the next time) or change the CPU (add another one). As we have mentioned, GNU/Linux uses the `/proc` directory to keep all the kernel configuration variables, which can be analysed and, in certain cases, 'adjusted' to achieve different or better performance levels.

To do this, we must use the `systune dump > /tmp/sysfile` command to obtain all the variables and their values in the `/tmp/sysfile` file (in other distributions, this can be done with `sysctl`). This file can be edited, changing the corresponding variable and then using the `systune -c /tmp/sysfile` command to reload them in `/proc`. The `systune` command also reads by default if we do not have the `-c` option in `/etc/systune.conf`. In this case, for example, we could modify (proceed carefully, because the kernel could be left inoperative) the variables of the category `/proc/sys/vm` (virtual memory) or `/proc/sys/kernel` (configuration of the core of the kernel).

In this same sense, it is also possible (for experts or people with nothing to lose) to change the maximum slice time, which the CPU scheduler of the operating system dedicates to each process in a circular manner (it is advisable

to use *renice* as practice). But GNU/Linux, unlike other operating systems, is a fixed value within the code, as it is optimised for different functions (yet it is possible to modify it). We can "play" (at our own risk) with a set of variables that make it possible to touch the *time slice* assigned to CPU (kernel-source-2.x.x/kernel/sched.c).

3) Reducing the number of calls

Another practical way of improving the performance is reducing the number of calls to the system, which cost the most CPU time. These calls are the ones usually invoked by the shell `fork()` and `exec()`. An inadequate configuration of the `PATH` variable and due to the fact that the `exec()` call does not save anything in the cache, the current directory (indicated with a `.`), could have a negative execution relationship. Consequently, we must always configure the `PATH` variable with the current directory as the last route. For example, in *bash* (or in *.bashrc*) we must: `export PATH = $PATH`. If this is not the case, the current directory is not there, or if it is, redo the `PATH` variable to declare it as the last route.

It should be remembered that a lot of interruption activity can affect CPU performance with regard to the processes that are being executed. By monitoring (`atsar -I`), we can see what the relationship of interruptions per second is and make decisions with regard to the devices that are causing them. For example, change the modem for a smarter one or change the communications structure if we detect excessive activity on the serial port to which it is connected.

4) Too much disk use

After the memory, a low response time could be due to the disks system. Firstly, we must verify that there is CPU time (for example, `idle > 20%`) available and that the in/out number is high (for example, `> 30 in/out/s`) using `atsar -u` and `atsar -d`. The solutions might be:

a) In a multi-disk system, planning where the most commonly used files are located to balance the traffic to them (for example, `/home` in a disk and `/usr` on another) and ensuring that they can use all the in/out capacities with the cache and concurrently of GNU/Linux (including, for example, planning the ide bus on which they will be). Then check that there is balance in the traffic using `atsar -d` (or `iostat`). In critical situations, we can consider purchasing a RAID disk system, which would make these adjustments automatically.

b) Bear in mind that better performance levels are achieved using two small disks instead of one large disk, equal to the combined size of the first two.

c) In systems with only one disk, for generally reducing space, four partitions are made in the following manner (from outside to inside): /, swap, /usr, /home; but this generates terrible in/out response times because if, for example, a user compiles from their directory /home/user and the compiler is in /usr/bin, the disk head will move along the whole length. In this case, it is better to join the partitions /usr and /home in one single one (larger) although this could present some inconveniences in terms of maintenance.

d) Increase the buffers of the cache of the in/out (see, for example: /proc/ide/hd...).

e) If we use an ext2fs, we can use the command: `dumpe2fs -h /dev/hd...` to obtain information on the disk and `tune2fs /dev/hd...` to change some of the configurable parameters of the disk.

f) Obviously, changing the disk for a higher-speed disk (RPM) will always have a positive effect on a system limited by the disk's in/out. [Maj96]

5) Improving TCP/IP aspects.

k) Examine the network with the `atsar` command (or also with `netstat -i` or with `netstat -s | more`) to analyse whether there are any fragmented packets, errors, drops, overflows etc., that may be affecting the communications and, consequently, the system (for example, in an NFS, NIS, FTP or Web server). If any problems are detected, we can analyse the network to consider any of the following actions:

a) Fragmenting the network through active elements that discard packets with problems or those that are not for machines in the segment.

b) Planning where the servers will be to reduce the traffic to them and the access times.

c) Adjust parameters of the kernel (/proc/sys/net/), for example, to obtain improvements in the throughput, type:

```
echo 600 > /proc/sys/net/core/netdev_max_backlog (300 by default).
```

6) Other actions on the parameters of the kernel.

There is another set of parameters on the kernel that can be tuned to obtain better performance levels, although, considering the points we have discussed, this should be performed with care, given that we could cause the opposite

effect or disable the system. Consult the distribution for the source code in kernel- *source*-2.4.18/Documentation/sysctl the vm.txt, fs.txt, kernel.txt and sunrpc.txt files:

a) `/proc/sys/vm`: controls the virtual memory (MV) of the system. The virtual memory makes it possible for the processes that do not access the main memory to be accepted by the system but in the swap device, for which, the programmer has no limit with regard to the size of their program (obviously, it must be less than the swap device). The parameters that may be tuned can be changed very easily with `gpowertweak`.

b) `/proc/sys/fs`: the kernel-FS interaction parameters can be adjusted, such as `file-max`.

c) And also over `/proc/sys/kernel`, `/proc/sys/sunrpc`.

7) Generating the kernel appropriate to our needs.

The optimisation of the kernel means choosing the compilation parameters in accordance with our needs. It is very important to first read the readme file in `/usr/src/linux`. A good configuration of the kernel will make it possible for it to run faster, providing more memory for the user processes and making the overall system more stable. There are two ways of building a kernel: monolithic (better performance levels) or modular (based on modules, there will be more portability if we have a very heterogeneous system and we do not wish to compile a kernel for each one of them). To compile your own kernel and adapt it to your hardware and needs, each distribution has its own rules (although the procedure is similar).

8) The following articles are very interesting:

http://people.redhat.com/alikins/system_tuning.html for information on optimising and tuning Linux server systems.

<http://www.linuxjournal.com/article.php?sid=2396> Performance Monitoring Tools for Linux; although this is an old article and some options are not available, the methodology still stands.

1.3. General optimisations

There is a series of general optimisations that can improve the system's performance:

1) Static or dynamic libraries: when a program is compiled, this can be done with a static library (`lib.a`), whose functioning code is included in the executable or with a dynamic library (`lib.so.xx.x`), where the library is loaded at the time of execution. Although the first guarantees a portable and secure

code, it consumes more memory. The programmer must decide which option is appropriate for their program including `-static` in the compiler options (not adding this means dynamic) or `o --disable-shared`, when the `configure` command is used. It is advisable to use (almost all new distributions do this) the standard library `libc.a` and `libc.so` of versions 2.2.x or higher (known as Libc6) which replaces the older ones.

2) Selecting the appropriate processor: generating executable code for the architecture on which the applications will be running. Some of the most influential parameters of the compiler are: `-march` (for example, `marchi 686` or `-march k6`) by simply typing `gcc -marchi 686`, the optimisation attributed `-O1,2,3` (`-O3` will generate the fastest version of the program, `gcc -O3 -march = i686`) and the attributes `-f` (consult the documentation for the different types).

3) Disk optimisation: currently, most computers include the UltraDMA (100) disk by default; however, in many cases, these are not optimised to provide the best performance levels. There is a tool (`hdparm`) that can be used to tune the kernel to the parameters of the IDE-type disk. We have to be careful with this tool, especially in UltraDMA disks (check the BIOS to ensure that the parameters for supporting DMA are enabled), as they can disable the disk. Check the references and the documentation ([Mou01] and `man hdparm`) to see which optimisations are the most important (and the risks involved), for example: `-c3`, `-d1`, `-X34`, `-X66`, `-X12`, `-X68`, `-mXX`, `-a16`, `-u1`, `-W1`, `-k1`, `-K1`. Each option means one form of optimisation and some are very high-risk, which means that we must know the disk very well. To consult the optimised parameters, we could use `hdparm -vtT /dev/hdX` (where X is the optimised disk) and the call to `hdparm` with all the parameters can be used in `/etc/init.d` to load it in the boot.

1.4. Additional configurations

There are more complementary configurations from the perspective of the security provided by optimisation, but they are mostly necessary when the system is connected to an Intranet or to the Internet. These configurations require the following actions [Mou01]:

a) Disabled the boot-up or other operating system: if someone has physical access to the machine, they could start up another preconfigured operating system and modify the existing one, which means that we should access the computer's BIOS settings to disable the boot using floppies or CD-ROMs and set up a password (remember the BIOS password, or you might have problems if you wish to change the configuration).

b) Configuration and network: it is advisable to disconnect from the network whenever we are adjusting the system. You can remove the cable or disable the device with `/etc/init.d/networking stop` (start to re-activate it) or with `ifdown eth0` (use `ifup eth0` to enable it) for any specific device.

c) Modify the `/etc/security` files in accordance with the system's usage and security needs. For example, in `access.conf` on who can log in to the system.

Format: permission: users : origins to - : users: from where	
-:ALL EXCEPT root: tty1	Disable access to all <i>no-root</i> over <i>tty1</i> .
-:ALL EXCEPT user1 user2 user3:console	prevents access except for <i>users1,2,3</i> but the latter may only access from the <i>console</i> .
-:user1:ALL EXCEPT LOCAL .uoc.edu 'group.conf':	

We should also configure the group to control what and how and also the maximum limits (`limits.conf`) for establishing the maximum times of usage of CPU, I/O etc. to avoid, for example, DoS attacks.

d) Maintain the security of the passwords of the root user: use at least 6 characters, with at least one character in capitals or some other symbol `'-._'`; this is not trivial; likewise, it is advisable to activate the password expiry option to force yourself to change it regularly, as well as limiting the amount of times one can enter an incorrect password. Likewise, we will have to change the parameter *min x* in the entry in `/etc/pam.d/passwd` to indicate the minimum number of characters used in the passwords (*x* is the number of characters).

e) Do not log in the system as the root user: create an account such as `sysadm` and work with it. If you access it remotely, you will always have to use `ssh` to connect to `sysadm` and, if necessary, carry out a `su -` to work as the *root*.

f) Set the maximum inactivity time: startup the `TMOUT` variable, at 360 for example (value expressed in seconds), which will be the maximum inactivity time that the shell will let pass before blocking; it is possible to put it in the configuration files of the shell (for example, `/etc/profile`, `/.bashrc`...). If we are using graphical environments (KDE, Gnome etc.), activate the option to exit the screensaver with password.

g) Configuration of the NFS in restricted mode: in `/etc/exports` export only what is necessary, without using wildcards, permitting only the read access and not permitting the write access by root, for example, with `/directory_exported host.domain.com (ro, root_squash)`.

h) Avoid boot ups from lilo (or grub) with the parameters: the system may be booted as Linux single, which will start up the operating system in single user mode. Configure the system so that the password is always required when booting up in this mode. In order to do this, in the `/etc/inittab` file, verify that the following line exists: `S:wait:/sbin/sulogin` and that `/bin/sulogin` is enabled. In addition, the `/etc/lilo.conf` file must have all the adequate permissions so that no one can modify it except the root user (`chmod 600 /etc/lilo.conf`). To avoid any accidental changes, change the blocking attributed with `chattr +i /etc/lilo.conf` (use `-i` when you wish to change). This file permits a series of options that should be considered: timeout or, if the system only has one operating system for booting immediately, restricted, to prevent others from being able to insert commands when booting such as `linux init = /bin/sh`, and have access as an unauthorised root user; in this case, the password must be used; if we only enter the password, we will be asked for the password for loading the image of the kernel. Grub has similar options.

i) Combination control Ctrl-Alt-Delete. To prevent others from being able to turn off the machine from the keyboard, insert a comment (#) in the first column of the following line:

```
ca:12345:ctrlaltdel:/sbin/shutdown -t1 -a -r now
Activate the changes with telinit q.
```

j) Avoid services that are not offered: block the `/etc/services` file so as not to admit non-contemplated services by blocking the file with `chattr +i /etc/services`.

k) Connection of the root: modify the file `/etc/securetty` which contains the TTY and VC (*virtual console*) in which the root can connect, leaving only one of each, for example, `tty1` and `vc/1`, and if it is necessary to connect as `sysadm` execute a `su`.

l) Eliminate user accounts that are not in use: delete the users/groups that are not necessary, including those that come by default (for example, operator, shutdown, ftp, uucp, games...), and leave only the necessary ones (root, bin, daemon, sync, nobody, sysadm) and the ones that were created with the installation of packages or using commands (the same with `/etc/group`). If the system is critical, we might consider blocking

(`chattr +i file`) the `/etc/passwd`, `/etc/shadow`, `/etc/group` and `/etc/gshadow` files to avoid their modification (be careful with this operation, because you will not subsequently be able to change the password).

m) Mount the partitions in a restrictive manner: in `/etc/fstab` use attributes for the partitions such as `nosuid` (makes it impossible to replace the user or group on the partition), `nodev` (does not interpret devices of characters or blocks on that partition) and `noexec` (does not permit the execution of files on this partition). For example:

```
/tmp /tmp ext2 defaults,nosuid,noexec 0 0
```

It is also advisable to mount the `/boot` on a separate partition and with read-only attributes.

n) Various protections: change the protections of the files in `/etc/init.d` (system services) to 700 so that only the root may modify them, start them up or stop them, and modify the `/etc/issue` and `/etc/issue.net` files so that they do not provide any information (operating system, version...) when someone connects through telnet, ssh etc.

o) SUID and SGID: a user may execute a command as an owner if they have the SUID or SGID bit activated, which would be reflected in an 's' SUID (`-rwsr-xr-x`) and SGID (`-r-xr-sr-x`). Therefore, it is necessary to delete the bit (`chmod a-s file`) from the commands that do not need it. These files can be searched with:

```
find / -type f -perm -4000 or -perm -2000 -print
```

We must proceed carefully with regard to the files that the SUID- GUID removes because the command could be disabled.

p) Suspicious files: you should regularly check for files with unusual names, hidden files, or files without a valid uid/gid, such as `'...'` (three points), `'..'` (point point space), `'..^G'`, for this, you will have to use:

```
find / -name ".*" -print | cat -v
```

or otherwise:

```
find / name ".." -print
```

To search non-valid uid/gids, use: `find / -nouser or -nogroup` (careful, because some installations are made with a user who is subsequently not identified and the administrator has to change).

q) Connection without password: do not allow the `.rhosts` file in any user unless it is strictly necessary (we recommend using `ssh` with a public password instead of methods based on `.rhosts`).

r) X Display manager: modify the file `/etc/X11/xdm/Xaccess` to specify the hosts that may connect through XDM and avoid any host having a login screen.

1.5. Monitoring

There are two very interesting tools for monitoring the system: Munin and Monit. Munin produces graphics on different parameters of the server (load average, memory usage, CPU usage, MySQL throughput, `eth0` traffic etc.) without excessive configurations, whereas `monit` verifies the availability of services such as Apache, MySQL, Postfix, and implements different actions such as re-activating a service that is not present. The combination provides important graphics for recognising where problems are being generated and what is generating them.

Let's say that our system is called `pirulo.org` and we have our page configured as `www.pirulo.org` with the documents in `/var/www/pirulo.org/web`. To install Munin on Debian Sarge, we can execute, for example, `apt-get install munin munin-node`.

We must then configure `munin` (`/etc/munin/munin.conf`) with:

```
dbdir /var/lib/munin
htmldir /var/www/www.pirulo.org/web/monitoring
logdir /var/log/munin
rundir /var/run/munin
tmpldir /etc/munin/templates
[pirulo.org]
address 127.0.0.1
use_node_name yes
```

The directory is then created, the permissions are changed and the service is restarted.

```
mkdir -p /var/www/pirulo.org/web/monitoring
chown munin:munin /var/www/pirulo.org/web/monitoring
/etc/init.d/munin-node restart
```

After a few minutes we will be able to see the first results in `http://www.pirulo.org/monitoring/` in the browser. For example, two graphs (load and memory) are shown below.

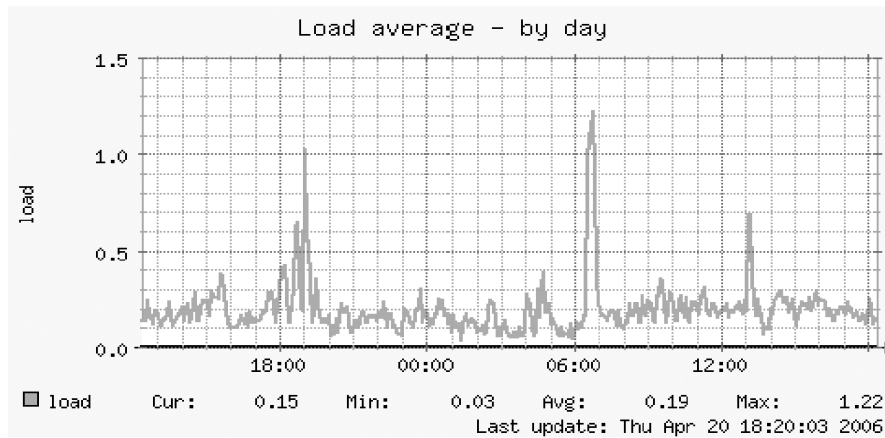


Figure 1

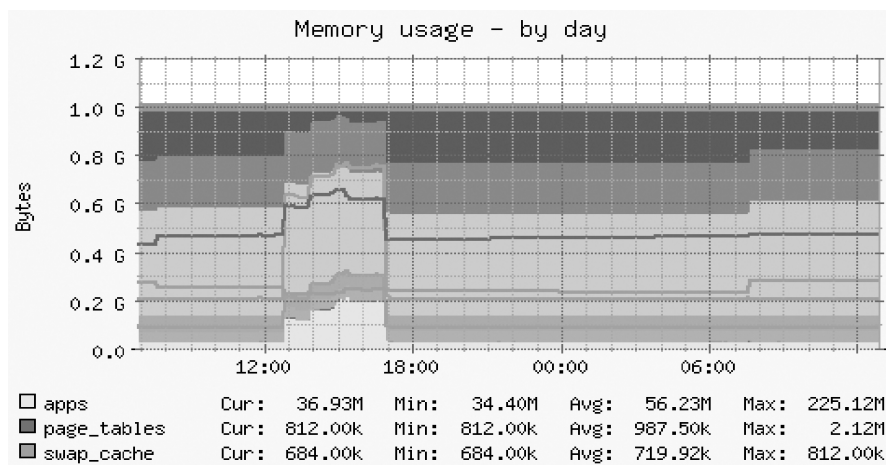


Figure 2

If you wish to maintain privacy in the graphs, all you have to do is set in a password to the access the directory with apache. For example, we can save the file `.htaccess` with the following contents in the directory `/var/www/pirulo.org/web/monitoring`:

```
AuthType Basic
AuthName "Members Only"
AuthUserFile /var/www/pirulo.org/.htpasswd
<limit GET PUT POST>
require valid-user
</limit>
```

We must then create the password file in `/var/www/pirulo.org/.htpasswd` with the command (such as root):

```
htpasswd -c /var/www/pirulo.org/.htpasswd admin
```

When we connect to www.pirulo.org/monitoring, it will not ask for the username (admin) and the password that we have entered after the preceding command.

To install monit, we execute `apt-get install monit` and we edit `/etc/monit/monitrc`. The default file includes a set of example, but we can obtain more from <http://www.tildeslash.com/monit/doc/examples.php>. For example, if we want to monitor proftpd, sshd, mysql, apache and postfix, by enabling the web interface of monit on port 3333, on `monitrc`, we can type:

```
set daemon 60
set logfile syslog facility log_daemon
set mailserver localhost
set mail-format { from: monit@pirulo.org }
set alert root@localhost
set httpd port 3333 and
allow admin:test

check process proftpd with pidfile /var/run/proftpd.pid
start program = "/etc/init.d/proftpd start"
stop program = "/etc/init.d/proftpd stop"
if failed port 21 protocol ftp then restart
if 5 restarts within 5 cycles then timeout

check process sshd with pidfile /var/run/sshd.pid
start program "/etc/init.d/ssh start"
stop program "/etc/init.d/ssh stop"
if failed port 22 protocol ssh then restart
if 5 restarts within 5 cycles then timeout

check process mysql with pidfile /var/run/mysqld/
mysqld.pid
group database
start program = "/etc/init.d/mysql start"
stop program = "/etc/init.d/mysql stop"
if failed host 127.0.0.1 port 3306 then restart
if 5 restarts within 5 cycles then timeout

check process apache with pidfile /var/run/apache2.pid
group www
start program = "/etc/init.d/apache2 start"
stop program = "/etc/init.d/apache2 stop"
if failed host www.pirulo.org port 80 protocol http
and request "/monit/token" then restart
if cpu is greater than 60% for 2 cycles then alert
if cpu > 80% for 5 cycles then restart
if totalmem > 500 MB for 5 cycles then restart
```



```

if children > 250 then restart
if loadavg(5min) greater than 10 for 8 cycles then stop
if 3 restarts within 5 cycles then timeout

check process postfix with pidfile /var/spool/postfix/
pid/master.pid
group mail
start program = "/etc/init.d/postfix start"
stop program = "/etc/init.d/postfix stop"
if failed port 25 protocol smtp then restart
if 5 restarts within 5 cycles then timeout

```

Consult the manual for more details <http://www.tildeslash.com/monit/doc/manual.php>. To verify that the Apache server works with Monit, we have to put the configuration that accesses to *if failed host www.pirulo.org port 80 protocol http and request "/monit/token" then restart*. If we cannot access this, it means that Apache does not work, which means that this file must exist (`mkdir /var/www/pirulo.org/web/monit; echo "pirulo" > /var/www/pirulo.org/web/monit/token`). It is also possible to configure monit so that it works on SSL (see http://www.howtoforge.com/server_monitoring_monit_munin_p2).

Finally, we must modify `/etc/default/monit` to enable monit and change `startup=1` and `CHECK_INTERVALS=60` for example (in seconds). If we start up monit (`/etc/init.d/monit start`) and we connect to `http://www.pirulo.org:3333`, we will see a screen similar to:

Monit Service Manager				
Process	Status	Uptime	CPU	Memory
proftpd	runnig	1h 19m	0,0%	1,2% [2348 Kb]
sshd	runnig	1h 56m	0,0%	0,7% [1508 Kb]
mysql	runnig	1h 29m	0,0%	7,1% [13664Kb]
apache	runnig	15m	0,0%	5,0% [9628 Kb]
postfix	runnig	1h 26m	0,0%	0,6% [1322 Kb]
Process status				
Parameter	Value			
Name	apache			
Pid file	/var/run/apache2.pid			
Status	running			
Group	www			
Monitoring mode	active			
Monitoring status	monitored			
Start program	/etc/nt.d/apache2 start			
Stop program	/etc/nt.d/apache2 stop			

Figure 3

There are more sophisticated tools for monitoring the network and network services using simple network management protocol (SNMP) and multi-router traffic grapher (MRTG), for example. More information on this subject can be found at http://www.linuxhomenetworking.com/wiki/index.php/Quick_HOWTO_:_Ch22_:_Monitoring_Server_Performance.

The MRTG (<http://oss.oetiker.ch/mrtg/>) was created basically to graph network data, but other data can be used to visualise its behaviour, for example, to generate load average statistics in the server. For this, we use the mrtg and atsar packages. Once installed, we will configure the /etc/mrtg.cfg file:

```
WorkDir: /var/www/mrtg
Target[average]: '/usr/local/bin/cpu-load/average'
MaxBytes[average]: 1000
Options[average]: gauge, nopercent, growright, integer
YLegend[average]: Load average
kMG[average]: ,,
ShortLegend[average]:
Legend1[average]: Load average x 100
LegendI[average]: load:
LegendO[average]:
Title[average]: Load average x 100 for pirulo.org
PageTop[average]: <H1>Load average x 100 for pirulo.org</
H1>
<TABLE>
<TR><TD>System:</TD>
<TD>pirulo.org</TD></TR>
<TR><TD>Maintainer:</TD>    <TD>webmaster@pirulo.org</
TD></TR>
<TR><TD>Max used:</TD> <TD>1000</TD></TR>
</TABLE>
```

To generate the data with atsar (or sar) we create a script in /usr/local/bin/cpu-load/average (which should have execution permissions for all) that will pass the data to mrtg:

```
#!/bin/sh
load='/usr/bin/atsar -u 1 | tail -n 1 | awk -F" " '{print
$10}''
echo "$load * 100" | bc | awk -F"." '{print $1}'
```

We must create and change the permissions in the directory /var/www/mrtg. By default, mrtg executes in the cron, but if we want to execute it, we can run mrtg /etc/mrtg.cfg and this will generate the graphs in /var/www/mrtg/average.html that we can visualise with the browser from <http://www.pirulo.org/mrtg/average.html>.

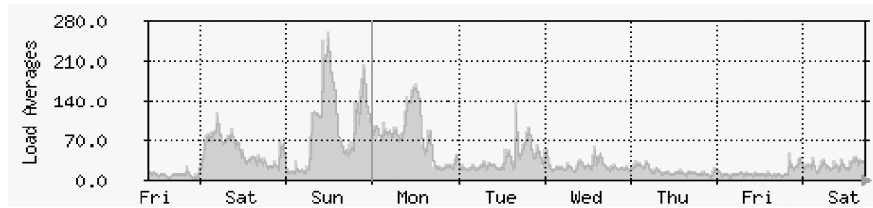


Figure 4

Other interesting packages that should be taken into account when monitoring a system are:

- Frysk (<http://sources.redhat.com/frysk/>): the objective of the frysk project is to create a monitoring system that is distributed and intelligent to monitor processes and threads.
- Cacti (<http://cacti.net/>): Cacti is a graphic solution designed for working together with the data of RRDTool's. Cacti provides different forms of graphs, acquisition methods and characteristics that the user can control very easily and is a solution that is adapted from a machine to a complex environment of machines, networks and servers.

We will now describe other tools which are no less interesting (in alphabetic order) that GNU/Linux incorporates (for example Debian) for monitoring the system. This is not an exhaustive list, but simply a selection of the most commonly used (we recommend seeing the man page of each tool for more information):

- atsar, ac, sac, sysstat, isag: auditing tools such as ac, last, accton, sa, atsar or isag (Interactive System Activity Grapher) for auditing hw and sw resources.
- arpwatsh; mon: ethernet/FDDI activity monitor that indicates whether changes have been made in the MACIP tables; network services monitor.
- diffmon, fcheck: generation of reports on changes to the configuration of the system and monitoring of the file systems so as to detect intrusions.
- fam: file alteration monitor.
- genpower: monitor for managing faults in the power supply.
- gkrellm: graphical CPU monitoring, processes (memory), file systems and users, disk, network Internet, swap etc.
- ksensors: (lm-sensors): motherboard monitor (temperature, power supply, fans etc.).

- .lcap, systune: removes capacities assigned to the kernel in the /proc/sys/ kernel file and adapts it to the needs with systune.
- logwatcher: log analyser.
- Munin and monit: graphical monitoring of the system.
- Powertweak and gpowertweak: monitoring and modifying different parameters of the hardware, kernel, network, VFS, or VM (allows us to modify some of the parameters shown before over /proc).
- gps, gtop, tkps, lavaps (from the most to the least user-friendly): various types of process monitors (they generally use information from /proc) and allow us to see resources, sockets, files, environment and other information that these use, as well as to administer their resources/statuses.
- swatch: system activity monitor through log files.
- vtgrab: monitoring remote machines (similar to VNC).
- whowatch: real time tool for monitoring users.
- wmnnd, dmachinemon: network traffic monitor and monitoring of a cluster on the network.
- xosview, si: graphical resources monitor and System Information.

The following figure shows the interfaces of ksensors, gkrellm and xosview, which present the results from the monitoring process in real time.

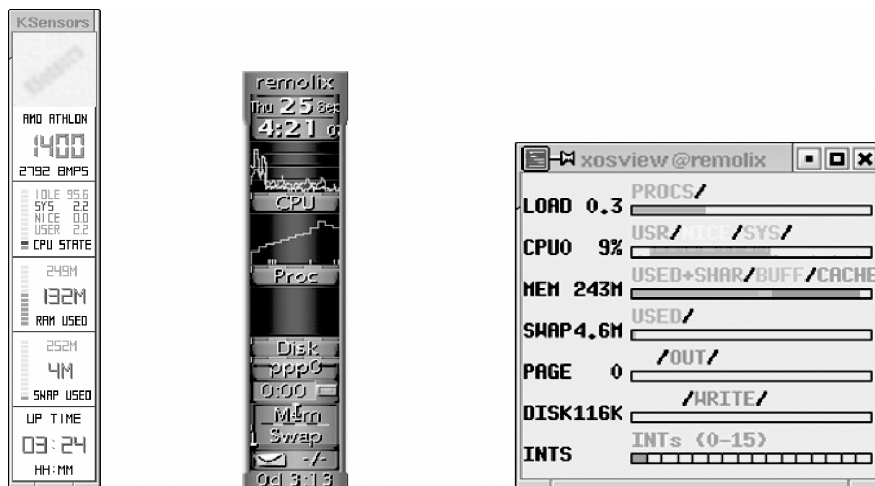


Figure 5

Activities

1) Perform a full system monitoring process using the tools that you think are most adequate and reach a diagnostic on the use of resources and the bottleneck that might exist in the system. Simulate the system's workload of the code of `sumdis.c` given in the unit that covers the clusters. For example, use:

```
sumdis 1 2000000
```

2) Change the parameters of the kernel and the compiler and execute the code mentioned in the preceding point (`sumdis.c`) with, for example:

```
time ./sumdis 1 1000000
```

3) The same with both kernels and formulate a conclusion regarding the results.

Bibliography

Other sources of reference and information

[Debc, Ibi]

Optimisation of Linux servers: http://people.redhat.com/alikins/system_tuning.html

Performance Monitoring Tools for Linux <http://www.linux>

Munin: <http://munin.projects.linpro.no/>

Monit: <http://www.tildeslash.com/monit/>

Monitoring with Munin and monit: http://www.howtoforge.com/server_monitoring_monit_munin

Monitoring with SNMP and MRTG: http://www.linuxhomenetworking.com/wiki/index.php/Quick_HOWTO_-_Ch22_-_Monitoring_Server_Performance

MRTG: <http://oss.oetiker.ch/mrtg/>

Frysk: <http://sources.redhat.com/frysk/>

Cacti: <http://cacti.net/>

Clustering

Remo Suppi Boldrito

PID_00148472



Universitat Oberta
de Catalunya

www.uoc.edu

Index

Introduction	5
1. Introduction to High Performance Computing (HPC)	7
1.1. Beowulf	8
1.1.1. How do we configure the nodes?	9
1.1.2. Benefits of distributed computing	10
1.2. How should we program to take advantage of concurrent computing?	12
1.2.1. Parallel virtual machine (PVM)	13
1.2.2. Message passing interface (MPI)	18
2. OpenMosix	24
3. Metacomputers, grid computing	27
3.1. Different computing architectures	27
3.2. Globus	29
3.3. Software, installation and administration of Globus	31
Activities	33
Bibliography	34
GNU Free Documentation License	43

Introduction

A computer cluster refers to a group of computers working closely together with a common aim. These computers consist of hardware, communication networks and software for working together as though they were all part of one single system. There are various reasons for which it would be desirable to set up these clusters, but one of the main ones is so as to be able to process information more efficiently and quicker, as though it were a single system. Generally, a cluster works on a local area network (LAN) and provides efficient communication, although the machines will be located close to each other physically. A bigger version of the concept is the grid, where the aim is the same, but it involves groups of computers connected to each other through a wide area network (WAN). Some programmers think of the grid as a cluster of clusters in a 'global' sense. Although the increasingly advanced technology and decreasing costs make it easier to set up these types of systems, the complexity and efforts required to use dozens or hundreds (or, in some cases, thousands) of computers are very great. However, the advantages in computing time mean that, despite this situation, these types of high performance computing (HPC) solutions are considered very attractive and are constantly developing. In this unit, we will show some of the most widely spread and used approaches. [Rad, Dieb, Prob, Prod, Proe, Gloa]

Note

A cluster is a group of computers working closely together, often connected on a LAN.

Grids are groups of computers connected with wide area networks (WAN).

1. Introduction to High Performance Computing (HPC)

The advances in technology have resulted in fast, low-cost and highly efficient processors and networks, which have brought about a change in the cost/performance ratio in favour of using interconnected processing systems in a single high-speed processor. This type of architecture can be classified into two basic configurations:

- *Tightly coupled systems*: these are systems in which the memory is shared by all the processors (shared memory systems) and the memory of each processor is 'seen' (by the programmer) as one single memory.
- *Loosely coupled systems*: they do not share memory (each processor has its own) and they communicate through messages passed through a network (message passing systems).

The first example is known as a parallel processing system and the second as a distributed computing system. In the latter case, we can say that a distributed system is a set of processors interconnected on a network in which each processor has its own resources (memory and peripherals) and they communicate by exchanging messages on the network.

Computing systems are a relatively recent phenomenon (we could say that computing history started in the seventies). Initially, they consisted of large, heavy, expensive systems, which could only be used by a few experts and they were inaccessible and slow. In the seventies, advances in technology led to some substantial improvements carried out using interactive jobs, time sharing and terminals and the sizes of the computers were reduced considerably. The eighties were characterised by a significant improvement in the performance and efficiency (which has continued to today) and a dramatic reduction in the sizes, with the creation of microcomputers. Computing continued to develop through workstations and advances in networking (from 10 Mbits/s LANs and 56 Kbytes/s WANs in 1973 to today's 1Gbit/s LANs and WANs with asynchronous transfer mode (ATM) and 1.2 Gbits/s), which is a fundamental factor in current multimedia applications and those that will be developed in the near future. Distributed systems, for their part, originated in the seventies (systems with 4 or 8 computers), but really became widespread in the nineties.

Although administrating/installing/maintaining distributed systems is a complicated task, given that they continue to grow, the basic reasons for their popularity are the increase in performance and efficiency that they provide in inherently distributed applications (due to their nature), the information that

can be shared by a group of users, the sharing of resources, the high fault tolerance and the possibility of ongoing expansion (the ability to add more nodes to gradually and continuously increase the performance and efficiency).

In the following sections we will look at some of the most common parallel/distributed processing systems, as well as the programming models used to generate code that can use these features.

1.1. Beowulf

Beowulf [Rad, Beo] is a multi-computer architecture that can be used for parallel/distributed applications (APD). The system basically consists of a server and one or more clients connected (generally) through Ethernet, without using any specific hardware. To explore this processing capacity, it is necessary for the programmers to have a distributed programming model that, whilst it is true that it is possible to do this through UNIX (socket, rpc), may require a very significant effort, given that the programming models are at the level of systems calls and C language, for example; but this working method can be considered as low-level.

The software layer provided by systems such as parallel virtual machine (PVM) and message passing interface (MPI) facilitates significantly the abstraction of the system and makes it possible to program parallel/distributed applications easily and simply. The basic working form is master-workers, in which there is a server that distributes the task that the workers perform. In large systems (systems with 1,024 nodes), there is more than one master and nodes dedicated to special tasks such as, for example, in/out or monitoring.

One of the main differences between Beowulf and a cluster of workstations (COW) is that Beowulf is 'seen' as a single machine in which the nodes are accessed remotely, as they do not have a terminal (or a keyboard), whereas a COW is a group of computers that can be used by both the COW users and other users interactively through the screen and keyboard. We must remember that Beowulf is not software that transforms the user's code into distributed code or that affects the kernel of the operating system (like Mosix, for example). It is simply a way of creating a cluster of machines that execute GNU/Linux and act as a supercomputer. Obviously, there are many tools that make it possible to achieve an easier configuration, library or modification to the kernel for obtaining better performance levels, but it is also possible to build a Beowulf cluster from a GNU/Linux standard and conventional software. The construction of a Beowulf cluster with two nodes, for example, can be achieved simply with the two machines connected through Ethernet using a hub, a standard GNU/Linux distribution (Debian) and the network file system (NFS) and after enabling the network services such as rsh or ssh. In such a situation, we might argue that we have a simple two node cluster.

Note

Various options:

- Beowulf
- OpenMosix
- Grid (Globus)

1.1.1. How do we configure the nodes?

First, we must modify (each node) /etc/hosts so that the localhost line only has 127.0.0.1 and does not include any machine name, such as:

```
127.0.0.1 localhost
```

And add the IPs of the nodes (and for all the nodes), for example:

```
192.168.0.1  pirulo1
192.168.0.2  pirulo2
...
```

It is possible to create a user (nteum) in all the nodes, create a group and add this user to the group:

```
groupadd beowulf
adduser nteum beowulf
echo umask 007 >> /home/nteum/.bash_profile
```

In this way, any file created by the nteum user or any within the group can be modified by the Beowulf cluster.

We must create an NFS server (and the rest of the nodes will be clients of this NFS). On the server, we create a directory as follows:

```
mkdir /mnt/nteum
chmod 770 /mnt/nteum
chown -R nteum:beowulf /mnt/nteum
```

Now we can export this directory from the server.

```
cd /etc
cat >> exports
/mnt/wolf 192.168.0.100/192.168.0.255 (rw)
<control d>
```

We must remember that our network will be 192.168.0.xxx and it is a private network, in other words, the cluster will not be seen from the Internet and we must adjust the configurations so that all the nodes can see each other (from the firewalls).

We should verify that the services are working:

```
chkconfig -add sshd
chkconfig -add nfs
chkconfig -add rexec
chkconfig -add rlogin
```

```
chkconfig -level 3 rsh on
chkconfig -level 3 nfs on
chkconfig -level 3 rexec on
chkconfig -level 3 rlogin on
```

To work securely, it is important to work with ssh instead of rsh, which means that we must generate the keys for interconnecting the machines-nteum user securely, without a password. To do this, we modify (we remove the comment #) the following lines in /etc/ssh/sshd_config:

```
RSAAuthentication yes
AuthorizedKeysFile .ssh/authorized_keys
```

We reboot the machine and we connect as the nteum user, given that this user will operate the cluster. To generate keys:

```
ssh-keygen -b 1024 -f ~/.ssh/id_rsa -t rsa -N ""
```

The id_rsa and id_rsa.pub files will have been created in the /home/nteum/.ssh library directory and we must copy id_rsa.pub in a file called authorized_keys in the same directory. And we modify the permissions with `chmod 644 ~/.ssh/aut*` and `chmod 755 ~/.ssh`.

Given that only the main node will be connected to the others (and not the other way round) we only need to copy the public key (id_rsa.pub) to each node in the directory/file /home/nteum/.ssh/authorized_keys of each node. In addition, on each node, we will have to mount the NFS adding /etc/fstab the line `pirulo1:/mnt/nteum /mnt/nteum nfs rw,hard,intr 0 0`.

As of this point, we already have a Beowulf cluster for executing applications that could be PVM or MPI (we will see this in the following sections). Over FC, there is an application (system-config-cluster) that makes it possible to configure a cluster based on a graphic tool. For more information, please see: http://www.redhat.com/docs/manuals/enterprise/RHEL-5-manual/Cluster_Administration/index.html.

1.1.2. Benefits of distributed computing

What are the benefits of parallel computing? We will see these with an example [Rad]. We have a program for adding numbers (for example, $4 + 5 + 6 \dots$) called sumdis.c and written in C:

```
#include <stdio.h>

int main (int argc, char** argv){

float initial, final, result, tmp;
```

```

if (argc < 2) {
    printf ("Use: %s N.° initial N.° final\n",argv[0]);
    exit(1);
}
else {
    initial = atol (argv[1]);
    final = atol (argv[2]);
    result = 0.0;
}
for (tmp = inicial; tmp <= final; tmp++){
    result + = tmp; }
printf("%f\n", result)
return 0;
}

```

We compile it with `gcc -o sumdis sumdis.c` and if we look at the execution of this program, with, for example:

```
time ./sumdis 1 1000000 (from 1 to 106)
```

we can see that the time in a Debian 2.4.18 machine with AMD Athlon 1.400 MHz 256 Mb RAM is (approximately) `real = 0,013` and `user = 0,010` in other words, 13 ms in total and 10 ms in user zone. If, however, we enter:

```
time ./sum 1 16000000 (from 1 to 16 * 106)
```

the time will be `real = 182`, in other words, 14 times more, which means, if we consider 160.000.000 ($160 \cdot 10^6$), the time will be approximately dozens of minutes.

The idea of distributed computing is: if we have a cluster of 4 machines (node1-node4) with a server, where the file is shared by NFS, it would be interesting to divide the execution through `rsh` (not advisable, but it is acceptable for this example), so that the first adds from 1 to 40.000.000, the second from 40.000.001 to 80.000.000, the third from 80.000.001 to 120.000.000 and the fourth from 120.000.001 to 160.000.000. The following commands show one possibility. We consider that the system has the directory `/home` shared by NFS and that the user (nteum) who will execute the script has configured `.rhosts` adequately so that it is possible to access their account without the password. In addition, if `tcpd` has been activated in `/etc/inetd.conf` in the `rsh` line, there must be the corresponding file in `/etc/hosts.allow`, which would allow us to access the four machines in the cluster:

<code>mkfifo out</code>	Creates a fifo queue in <code>/home/nteam</code>

```
./distr.sh & time cat salida | awk '{total + = $1 } \
END printf "%lf", total}'
```

Executes the command `distr.sh`; the results are collected and added whilst the execution time is measured

The shell script `distr.sh` can be something like:

```
rsh node1 /home/nteum/sumdis 1 40000000 > /home/nteum/out <
/dev/null &
rsh node2 /home/nteum/sumdis 40000001 80000000 > /home/nteum/
out < /dev/null &
rsh node3 /home/nteum/sumdis 80000001 120000000 > /home/
nteum/out < /dev/null &
rsh node4 /home/nteum/sumdis 120000001 160000000 > /home/
nteum/out < /dev/null &
```

We can observe that the time is significantly reduced (by a factor of approximately 4) and not exactly lineally, but almost. Obviously, this example is very simple and is only used for demonstrative purposes. The programmers use libraries that allow them to set the execution time, the creation and communication of processes in a distributed system (such as PVM and MPI).

1.2. How should we program to take advantage of concurrent computing?

There are various ways of expressing the concurrency in a program. The most common two are:

- 1) Using threads (or processes).
- 2) Using processes in different processors that communicate through messages (MPS, *message passing system*).

Both methods can be implemented on different hardware configurations (share memory or messages) but MPS systems can involve latency and speed problems with the messages on the network, which can be a negative factor. However, with the advances in network technology, these systems have grown in popularity (and in number). A message is extremely simple:

```
send(destination,msg)
recv(origin,msg)
```

The most common APIs today are PVM and MPI and, in addition, they do not limit the possibility of using threads (even if it is at a local level) or of having concurrent processing and in/out. On the other hand, on a machine with

shared memory (SHM) it is only possible to use threads and there is the severe problem of scalability, given that all the processors use the same memory and the number of processors in the system is limited by the memory's bandwidth.

To summarise, we can conclude that:

- 1) Proliferation of multitask (multi-user) machines connected through a network with distributed services (NFS and NIS YP).
- 2) They are heterogeneous systems with networked operating systems (NOS) that offer a series of distributed and remote services.
- 3) Distributed applications can be programmed at different levels:
 - a) Using a client-server model and programming at low-level (sockets).
 - b) The same model but with a "high-level" API (PVM, MPI).
 - c) Using other programming models such as programming oriented to distributed objects (RMI, CORBA, Agents...).

1.2.1. Parallel virtual machine (PVM)

PVM [Proe] is an API that makes it possible to generate, from the perspective of the application, a dynamic cluster of computers, which constitutes a virtual machine (VM). The tasks can be created dynamically (spawned) and/or eliminated (killed) and any PVM task can send a message to another. There is no limit to the size or number of messages (according to the specifications, although there may be hardware/operating system combinations that result in limitations on message size) and the model supports fault tolerance, resource control, processes control, heterogeneity in the networks and in the hosts.

The system (VM) has tools for controlling the resources (adding or deleting hosts from the virtual machine), processes control (dynamic creation/elimination of processes), different communication models (blocking send, blocking/nonblocking receive, multicast), dynamic task groups (a task can be attached or removed from a group dynamically) and fault tolerance (the VM detects the fault and it can be reconfigured).

The PVM structure is based, on the one hand, on the daemon (pvm3d) that resides in each machine and is interconnected using UDP, and, on the other hand, the PVM library (libpvm3.a), which contains all the routines for sending/receiving messages, creating/eliminating processes, groups, synchronisation etc. which will use the distributed application.

PVM has a console (pvm) that makes it possible to start up the daemon, create the VM, execute applications etc. It is advisable to install the software from the distribution, given that the compilation requires a certain amount of 'dedication'. To install PVM on Debian, for example, we must include two packages (minimum): pvm and pvm-dev (the pvm console and utilities are in the first and the libraries, header and the rest of the compiling tools are in the second). If we only need the library because we already have the application, we can install only the libpvm3 package).

To create a parallel/distributed application in PVM, we can start with the standard version or look at the physical structure of the problem and determine which parts can be concurrent (independent). The concurrent parts will be candidates for being rewritten as parallel code. In addition, we must consider whether it is possible to replace the algebraic functions with their paralleled versions (for example, ScaLapack, Scalable Linear Algebra Package, available in Debian as scalapack-pvm | mpich-test | dev, scalapack1-pvm | mpich depending on whether it is PVM or MPI). It is also convenient to find out whether there is any similar parallel application (<http://www.epm.ornl.gov/pvm>) that might guide us as to the construction method of the parallel application.

Parallelising a program is not an easy task, as we have to take into account Amdahl's law.

Amdahl's law states that speedup is limited by the fraction of code (f) that can be paralleled: **speedup = $1/(1-f)$** .

This law implies that a sequential application $f = 0$ and the speedup = 1, with all the parallel code $f = 1$ and speedup = infinite (!), with possible values, 90% of the parallel code means a speedup = 10 but with $f = 0.99$, speedup = 100. This limitation can be avoided with scalable algorithms and different application models:

- 1) Master-worker: the master starts up all the workers and coordinates the work and in/out.
- 2) Single process multiple data (SPMD): the same program that executes with different sets of data.
- 3) Functional: various programs that perform a different function in the application.

With the pvm console and with the add command we can configure the VM whilst adding all the nodes. In each of these nodes, there must be the directory ~/pvm3/bin/LINUX, with the binaries of the application. The variables `PVM_ROOT` = Directory must be declared, where the lib/LINUX/libpvm3.a is

Note

Amdahl's law
 $\text{speedup} = 1/(1-f)$
f is the fraction of parallel code

and PVM_ARCH=LINUX, which can be placed, for example, in file `/.cshrc`. The default shell of the user (generally a NIS user or, if not, the same user must be in each machine with the same password) should be `csh` (if we use `rsh` as a means of remote execution) and the file `/.rhosts` must be configured to provide access to each node without the password. The PVM package incorporates an *rsh-pvm* that can be found in `/usr/lib/pvm3/bin` as an `rsh` specifically made for PVM (see the documentation), as there are some distributions that do not include it, for security reasons. It is advisable to configure, as we have shown, the `ssh` with the public keys of the server in `./ssh/authorized_keys` of the directory of each user.

As an example of PVM programming, we show a program of the server-client type, where the server creates the child nodes, sends the data, these nodes circulate the data a determined number of times between the child nodes (the first node on the left receives a piece of data, processes it and sends it to the one on the right), whilst the parent nodes waits for each child node to finish.

Example of PVM: master.c

To compile in Debian:

```
gcc -O -I/usr/share/pvm3/include/ -L/usr/share/pvm3/lib/LINUX -o master master.c -lpvm3
```

The directories in `-I` and in `-L` must be where the includes `pvm3.h` and `libpvm*` are located, respectively.

Execution:

- 1) execute the `pvm` daemon with `pvm`
- 2) execute `add` to add the nodes (this command can be skipped if we only have one node)
- 3) execute `quit` (we leave `pvm` but it continues to execute)
- 4) we execute `master`

Note

Compiling PVM:

```
gcc -O -I/usr/include/ -o output output.c -lpvm3
```

```
#include <stdio.h>
#include "pvm3.h"
#define SLAVENAME "/home/nteum/pvm3/client"
main() {
    int mytid, tids[20], n, nproc, numt, i, who, msgtype, loops;
    float data[10]; int n_times;

    if( pvm_parent() ==PvmNoParent ){
        /*Return if this is the parent or child process */
        loops = 1;
        printf("\n How many children (120)? ");
        scanf("%d", &nproc);
        printf("\n How many child-child communication loops (1 - 5000)? ");
        scanf("%d", &loops); }

    /*Redirects the in/out of the children to the parent */
    pvm_catchout(stdout);

    /*Creates the children */
    numt = pvm_spawn(SLAVENAME, (char**)0, 0, "", nproc, tids);
    /*Starts up a new process, 1st: executable child, 2nd: argv, 3rd :options,
```

```
4th :where, 5th :N.º copies, 6th :matrix of id*/
printf("Result of Spawn: %d \n", numt);

/*Has it managed?*/
if( numt &lt; nproc ){
    Printf("Error creating the children. Error code:\n");
    for( i = numt ; i<nproc ; i++ ) {
        printf("Tid %d %d\n",i,tids[i]); }
    for( i = 0 ; i<numt ; i++ ){
        pvm_kill( tids[i] ); } /*Kill the processes with id in tids*/
    pvm_exit();
    exit(); /*Finish*/
}

/*Start up parent program, initialising the data */
n = 10;
for( i = 0 ; i<n ; i++ ){
    data[i] = 2.0;}
/*Broadcast with initial data to slaves*/
pvm_initsend(PvmDataDefault);.
/*Delete the buffer and specify message encoding*/
pvm_pkint(&loops, 1, 1);
/*Package data in the buffer, 2nd N.º, 3*:stride*/
pvm_pkint(&nproc, 1, 1);
pvm_pkint(tids, nproc, 1);
pvm_pkint(&n, 1, 1);
pvm_pkfloat(data, n, 1);
pvm_mcast(tids, nproc, 0);
/*Multicast in the buffer to the tids and wait for the result from the children*/
msgtype = 5;
for( i = 0 ; i < nproc ; i++ ){
    pvm_rcv( -1, msgtype );
    /*Receive a message, -1 :of any, 2nd:tag of msg*/
    pvm_upkint( &who, 1, 1 );
    /*Unpackage*/
    printf("Finished %d\n",who);
}
pvm_exit();
}
```


Example of PVM: *client.c*

To **compile** in Debian:

```
gcc -O -I/usr/share/pvm3/include/ -L/usr/share/pvm3/lib/LINUX -or client client.c -lpvm3
```

The directories in -I and in -L must be where the included pvm3.h and libpvm* are located, respectively.

Execution:

This is not necessary as the master will start them up, but the client must be in /home/nteum/pvm3

```
#include <stdio.h>
#include "pvm3.h"

main() {
    int mytid; /*Mi task id*/
    int tids[20]; /*Task ids*/
    int n, me, i, nproc, master, msgtype, loops; float data[10];
    long result[4]; float work();
    mytid = pvm_mytid(); msgtype = 0;

    pvm_recv( -1, msgtype );
    pvm_upkint(&loops, 1, 1);
    pvm_upkint(&nproc, 1, 1);
    pvm_upkint(tids, nproc, 1);
    pvm_upkint(&n, 1, 1);
    pvm_upkfloat(data, n, 1);
    /*Determines which child it is (0 -- nproc-1) */
    for( i = 0; i < nproc ; i++ )
        if( mytid == tids[i] ){ me = i; break; }

    /*Processes and passes the data between neighbours*/
    work( me, data, tids, nproc, loops);

    /*Send the data to the master */
    pvm_initsend( PvmDataDefault );
    pvm_pkint( &me, 1, 1 );
    msgtype = 5;
    master = pvm_parent(); /*Find out who created it */
    pvm_send( master, msgtype);
    pvm_exit();
}

float work(me, data, tids, nproc, loops)
int me, *tids, nproc; float *data; {
    int i,j, dest; float psum = 0.0, sum = 0.1;
    for (j = 1; j <= loops; j++){
        pvm_initsend( PvmDataDefault );
        pvm_pkfloat( &sum, 1, 1 );
        dest = me + 1;
```

```

    if( dest == nproc ) dest = 0;
    pvm_send( tids[dest], 22 );
    i = me - 1;
    if (me == 0 ) i = nproc-1;
    pvm_recv( tids[i], 22 );
    pvm_upkfloat( &psum, 1, 1 );
  }
}

```

The programmer is assisted by a graphic interface that is of great help (see following figure), which acts as a PVM console and monitor, called `xpvm` (in Debian XPVM, install package `xpvm`), which makes it possible to configure the VM, execute processes, visualise the interaction between tasks (communications), statuses, information etc.



Figure 1

1.2.2. Message passing interface (MPI)

The definition of the API of MPI [Prob, Proc] has been the work resulting from MPI Forum (MPIF), which is a consortium of more than 40 organisations. MPI has influences from different architectures, languages and works in the world of parallelism such as: WRC (Ibm), Intel NX/2, Express, nCUBE, Vertex, p4, Parmac and contributions from ZipCode, Chimp, PVM, Chamaleon, PICL. The main objective of MPIF was to design an API, without any particular relation with any compiler or library, so that efficient memory-to-memory copy communication, computing and concurrent communication and communication downloads would be possible, provided there is a communications coprocessor. In addition, it supports development in heterogeneous environments, with interface C and F77 (including C++, F90), where communication will be reliable and the faults resolved by the system. The API also needed an interface for different environments (PVM, NX, Express, p4...) and an implementation that was adaptable to different platforms with insignificant changes that did not interfere with the operating system (thread-safety). This API was designed especially for programmers that used message passing

paradigm (MPP) in C and F77 to take advantage of the most important characteristic: portability. The MPP can be executed on multiprocessor machines, WS networks and even on machines with shared memory. The MPI1 version (the most widespread version) does not support the dynamic creation (spawn) of tasks, but MPI2 (which is developing at a growing rate) does.

Many aspects have been designed to take advantage of the benefits of communications hardware on scalable parallel computers (SPC) and the standard has been mostly accepted by parallel and distributed hardware manufacturers (SGI, SUN, Cray, HPConvex, IBM, Parsystec...). There are freeware versions (mpich, for example) (which are completely compatible with the commercial implementations from the hardware manufacturers) and they include point-to-point communications, collective operations and process groups, communications and topology contexts, support for F77 and C and a control, administration and profiling environment. But there are also some unresolved points, such as: SHM operations, remote execution, program construction tools, debugging, control of threads, administration of tasks, concurrent in/out functions (most of these problems arising from a lack of tools are resolved in version 2 of API MPI2). The function in MPI1, as there is no dynamic process creation, is very simple, given that of so many processes as tasks that exist, autonomous and executing their own multiple instruction multiple data (MIMD) style code and communicating via MPI calls. The code may be sequential or multithread (concurrent) and MPI works in threadsafe mode, in other words, it is possible to use calls to MPI in concurrent threads, as the calls re-enter.

To install MPI, it is recommended that you use the distribution, given that compiling it is extremely complex (due to the dependencies that it needs from other packages). Debian includes Mpich version 1.2.7-2 (Etch) in the mpich-bin package (the mpich one is obsolete) and also mpich-mpd-bin (version of a multipurpose daemon that includes support for scalable processes, management and control). The mpich-bin implements the MPI 1.2 standard and some parts of MPI 2 (such as, for example, parallel in/out). In addition, this same distribution includes another implementation of MPI called LAM (lam* packages and documentation in </usr/doc/lam-runtime/release.html>). The two implementations are equivalent, from the perspective of MPI, but they are managed differently. All the information on Mpich can be found (after installing the *mpich** packages) in </usr/share/doc/mpi> (or in </usr/doc/mpi>). Mpich needs rsh to execute in other machines, which means that we have to insert the user directory in a `~/.rhosts` file with lines in the following format: *host username* to allow the *username* to enter the *host* without the password (the same as PVM). It should be remembered that we have to install the rshserver package on all the machines and if there is tcpd in `/etc/inetd.conf` on *rsh.d*, we must enable the *hosts* in `/etc/hosts.allow`. In addition, we must have mounted the directory of the user by NFS in all the machines and the `/etc/mpich/machines.LINUX`

file must contain the *hostname* of all the machines that comprise the cluster (one machine per line, by default, appears as *localhost*). In addition, the user must have the Csh as the shell by default.

On Debian, we can install the *update-cluster* package to help with the administration. The installation of Mpich on Debian uses ssh instead of rsh for security reasons, although there is a link of rsh =>ssh for compatibility. The only difference is that we must use the ssh authentication mechanisms for the connection without password through the corresponding files. Otherwise, for each process that executes, we will have to enter the password before execution. To allow the connection between machines, with ssh, without the password, we must follow the procedure mentioned in the preceding section. To check it, we can run `ssh localhost` and then we should be able to log in without the password. Bear in mind that if we install Mpich and LAM-MPI, the `mpirun` of Mpich will be called `mpirun.mpich` and the `mpirun` will be that of LAM-MPI. It is important to remember that `mpirun` of LAM will use the `lamboot` daemon to form the distributed topology of the VM.

The `lamboot` daemon has been designed so that users can execute distributed programs without having root permissions (it also makes it possible to execute programs in a VM without calls to MPI). For this reason, to execute `mpirun`, we will have to do it as a user other than the root and execute `lamboot` beforehand. `lamboot` uses a configuration file in `/etc/lam` for the default definition of the nodes (see `bhost*`); please consult the documentation for more information (<http://www.lam-mpi.org/>). [Lam]

To compile MMPI programs, we can use the `mpicc` command (for example, `mpicc -o test test.c`), which accepts all the options of `gcc` although it is advisable to use (with modifications) some of the *makefiles* that are located in the `/usr/doc/mpich/examples` file. It is also possible to use *mpireconfig Makefile*, that uses the `Makefile.in` file as an entry to generate the *makefile* and is much easier to modify. After, we can run:

```
mpirun -np 8 programme
```

or:

```
mpirun.mpich -np 8 programme
```

where `np` is the number of processes or processors in which the program will execute (8, in this case). We can put in the number we like, as Mpich will try to distribute the processes in a balanced manner better between all the machines of `/etc/mpich/machines.LINUX`. If there are more processes than processors, Mpich will use the swap characteristics of GNU/Linux to simulate

parallel execution. In Debian and in the directory `/usr/doc/mpich-doc` (a link to `/usr/share/doc/mpich-doc`), we can find all the documentation in different formats (commands, API of MPI etc.).

To compile MPI: `mpicc -O -o output output.c`

Execute Mpich: `mpirun.mpich -np N°_processes output`

We will now see two examples (which are included in the distribution of Mpich 1.2.x in directory `/usr/doc/mpich/examples`). `Srtest` is a simple program for establishing communications between point-to-point processes and `cpi` calculates the value of Pi in distributed form (through integration).

Point-to-point communications: `srtest.c`

For compiling: `mpicc -O -o srtest srtest.c`

Execution of Mpich: `mpirun.mpich -np N°_processes srtest` (will ask for the password [*N°_processes* - 1] times if we do not have direct access through *ssh*).

Execution of LAM: `mpirun -np N°_processes srtest` (must be a user other than the root)

```
#include "mpi.h"
#include <stdio.h>
#define BUFLLEN 512
int main(int argc, char *argv[]) {
    int myid, numprocs, next, namelen;
    char buffer[BUFLLEN], processor_name[MPI_MAX_PROCESSOR_NAME]; MPI_Status status;
    MPI_Init(&argc,&argv);

    /* Must be placed before other MPI calls, always */
    MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
    MPI_Comm_rank(MPI_COMM_WORLD,&myid);
    /*Integrates the process in a communications group*/
    MPI_Get_processor_name(processor_name,&namelen);
    /*Obtains the name of the processor*/
    fprintf(stderr,"Process %d on %s\n", myid, processor_name);
    strcpy(buffer,"Hello People");
    if (myid==numprocs-1) next = 0;
    else next = myid+1;
    if (myid==0) { /*If it is the initial, send string of buffer*/.
        printf("%d Send '%s' \n",myid,buffer);
        MPI_Send(buffer, strlen(buffer)+1, MPI_CHAR, next, 99,
            MPI_COMM_WORLD);
        /*Blocking Send, 1 o :buffer, 2 or :size, 3 or :type, 4 or :destination, 5
        or :tag, 6 or :context*/
        /*MPI_Send(buffer, strlen(buffer)+1, MPI_CHAR,
        MPI_PROC_NULL, 299,MPI_COMM_WORLD);*/
        printf("%d receiving \n",myid);
        /* Blocking Recv, 1 o :buffer, 2 or :size, 3 or :type, 4 or :source, 5
        or :tag, 6 or :context, 7 or :status*/
        MPI_Recv(buffer, BUFLLEN, MPI_CHAR, MPI_ANY_SOURCE, 99, MPI_COMM_WORLD,&status);
```

```

        printf("%d received '%s' \n",myid,buffer) }
    else {
        printf("%d receiving \n",myid);
        MPI_Recv(buffer, BUFLen, MPI_CHAR, MPI_ANY_SOURCE, 99, MPI_COMM_WORLD,&status);
        /*MPI_Recv(buffer, BUFLen, MPI_CHAR, MPI_PROC_NULL, 299,MPI_COMM_WORLD,&status);*/
        printf("%d received '%s' \n",myid,buffer);
        MPI_Send(buffer, strlen(buffer)+1, MPI_CHAR, next, 99,
        MPI_COMM_WORLD);
        printf("%d sent '%s' \n",myid,buffer);}
    MPI_Barrier(MPI_COMM_WORLD); /*Synchronises all the processes*/    MPI_Finalize();
    /*Frees up the resources and ends*/ return (0);
}

```

Calculation of distributed PI: **cpi.c**

For compiling: *mpicc O* or *cpi cpi.c*.

Execution of Mpich: *mpirun.mpich -np N.º* processes *cpi* (will ask for the password (N.º processes - 1) times if we do not have direct access through *ssh*).

Execution of LAM: *mpirun -np N.º* processes *cpi* (must be a user other than root).

```

#include "mpi.h"
#include <stdio.h>
#include <math.h>
double f( double );
double f( double a) { return (4.0 / (1.0 + a*a)); }
int main( int argc, char *argv[] ) {
    int done = 0, n, myid, numprocs, i;
    double PI25DT = 3.141592653589793238462643;
    double mypi, pi, h, sum, x;
    double startwtime = 0.0, endwtime;
    int namelen;
    char processor_name[MPI_MAX_PROCESSOR_NAME];

    MPI_Init(&argc,&argv);
    MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
        /*Indicates the number of processes in the group*/
    MPI_Comm_rank(MPI_COMM_WORLD,&myid);
        /*Id of the process*/    MPI_Get_processor_name(processor_name,&namelen);
        /*Name of the process*/

    fprintf(stderr,"Process %d on %s\n", myid, processor_name);
    n = 0;
    while (!done) {
        if (myid ==0) { /*If it is the first...*/
            if (n ==0) n = 100; else n = 0;
            startwtime = MPI_Wtime();} /* Time Clock */
        MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);    /*Broadcast to the rest*/
        /*Send from 4th arg. to all
        the processes of the group. All others that are not 0
        will copy the buffer from 4 or arg -process 0-*/ /*1.º:buffer,

```

```

    2nd :size, 3rd :type, 5th :group */
    if (n == 0) done = 1; else {
        h = 1.0 / (double) n;
        sum = 0.0;
        for (i = myid + 1; i <= n; i += numprocs) {
            x = h * ((double)i - 0.5); sum += f(x); }
        mypi = h * sum;
        MPI_Reduce(&mypi, &pi, 1, MPI_DOUBLE, MPI_SUM, 0,
        MPI_COMM_WORLD);
        /* Combines the elements of the Send Buffer of each process of the
        group using the operation MPI_SUM and returns the result in
        the Recv Buffer. It must be called by all the processes of the
        group using the same arguments*/ /*1st :sendbuffer, 2nd
        :recvbuffer, 3rd :size, 4th :type, 5th :oper, 6th :root, 7th
        :context*/
        if (myid == 0) { /*Only the P0 prints the result*/
            printf("Pi is approximately %.16f, the error is %.16f\n", pi, fabs(pi - PI25DT));
            endwtime = MPI_Wtime();
            printf("Execution time = %f\n", endwtime-startwtime); }
        }
    }

    MPI_Finalize(); /*Free up resources and finish*/
    return 0;
}

```

As XPVM exists in PVM, in MPI there is an analogous application (more sophisticated) called XMPI (xmapi in Debian). It is also possible to install a library, libxmpi3, which implements the XMPI protocol to graphically analyse MPI programs with more details than offered in xmapi. The following figure shows some of the possible graphics in xmapi.

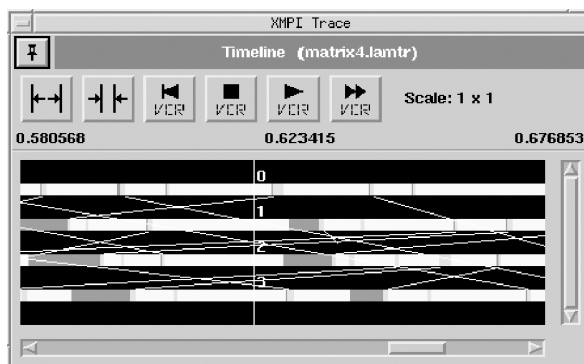


Figure 2. XMPI

2. OpenMosix

OpenMosix [Prod] is a software package that transforms a set of machines connected by a network under GNU/Linux in a cluster. This balances the workload automatically between the different nodes of the cluster and the nodes can be joined or the cluster left without interrupting the service. The load is distributed between the nodes, taking into account the speed of the connection and the CPU. OpenMosix is part of the kernel (through a Linux Kernel Patch) and maintains total compatibility with GNU/Linux, the user programs, files and resources. Another characteristic of OpenMosix is that it incorporates a powerful and optimised file system (oMFS) for HPC (high performance computing) applications. In Debian Woody, we can install OpenMosix from `openmosix-dev` (libraries and headers), `kernel-patch-openmosix` (OpenMosix patch), `openmosix` (administration tools). Likewise, it is possible to install `mosix` (see the documentation for the difference, especially with regard to the licenses, between Mosix and OpenMosix). In Debian versions subsequent to Woody, it is not included as a package (stable) and it will be necessary to go to <http://openmosix.sourceforge.net/> to obtain the packages (or resources) and the installation guides (<http://howto.x-tend.be/openMosix-HOWTO/>).

OpenMosix uses a configuration file that is generally found in `/etc` (see documentation for older versions of this file), which is called `openmosix.map` and which should be in each node. Its format is very simple and each line has three fields: `Nodo_ID IP-Address(or hostname) Range-size`

An example would be:

```
1 node1 1
2 node2 1
3 node3 1
4 192.168.1.1 1
5 192.168.1.2 1
```


It is also possible to use a range where the ID and the IP increase respectively. We have to ensure that we have the same configuration and the same version of OpenMosix in each node. To execute OpenMosix, in each node we must type:

```
setpe -w -f /etc/openmosix.map
```

We can also use the OpenMosix script (copying it from userspace-tools to /etc/init.d) to start it up during boot.

The oMFS file system permits remote access to all the files in the cluster, as though they were locally mounted. The file systems (FS) of the other nodes can be mounted on /mfs and, therefore, the files in /home on node 3 will be seen on each machine in /mfs/3/home.

All the UIDs (User IDs) and GIDs (Group IDs) of the FS on each node of the cluster must be equal (OpenLdap could be used for this).

To mount the oMFS, we must modify /etc/fstab with an entry such as: `mfs_mnt /mfs mfs dfsa = 1 0 0` and to enable or disable it: `mfs_mnt /mfs mfs dfsa = 0 0 0`.

Afterward, the FS of each node will be seen in `mfs/[openMosixNode ID]/`. Once installed, it will be possible to execute a very simple script various times, such as, for example (see *Howto* of OpenMosix):

```
awk 'BEGIN {for(i = 0;i<10000;i++)for(j = 0;j<10000;j++);}'
```

And, subsequently, we can observe the behaviour with `mosmom` or with `open-mosixview` (recommended). OpenMosix has a daemon (`omdiscd`), which makes it possible to automatically configure the cluster eliminating the need to edit and configure /etc/openmosix.map. This daemon uses multicast to indicate the other nodes that it is also an OpenMosix node, which means that, once `omdiscd` has booted, this daemon will join the cluster automatically. For this to happen, we need to have the default routing (GW) of the network properly configured. Once it has executed (`omdiscd`), a series of messages indicating the status of the cluster and the configuration will be generated. We can use the `showmap` command to see the new configuration generated by `omdiscd`. OpenMosix provides a set of tools that the administrator can use to configure and tune the OpenMosix cluster. These tasks can be performed with tools in the space of a user (*migrate*, *mon*, *mosctl*, *mosrun*) or through the /

proc/hpc interface. It is important to remember that up to OpenMosix version 2.4.16, the interface was called /proc/mosix and that, since version 2.4.17, it has been called /proc/hpc.

We will now present a summary of the configuration tools that are executed in the space of a user; for /proc/hpc consult the references:

- `migrate [PID] [OpenMosix ID]`: sends a migration request to a process.
- `mon`: is a monitor with a text interface that shows information on the cluster through a bar diagram.
- `mosctl`: is the configuration tool of OpenMosix. Using the options (`stay`, `lstay`, `block`, `quiet`, `mfs`, `expel`, `bring`, `get-tune`, `getyard`, `getdecay`) we can indicate whether processes can migrate or not, the use of MFS, obtain information on the load, balance on the load etc.
- `mosrun [h | OpenMosix ID | list of OpenMosix IDs] command [arguments]`: executes a command on a determined node.

3. Metacomputers, grid computing

The computing requirements that are needed for certain applications are so large that they require thousands of hours to be able to execute in cluster environments. Such applications have promoted the creation of virtual computers on networks, metacomputers or grid computers. This technology has made it possible to connect execution environments, high-speed networks, databases, instruments etc., distributed in different geographic locations. This makes it possible to achieve a processing power that would not be economically viable in any other way with excellent results. Examples of their application are experiments such as the I-WAY networking (which connects supercomputers from 17 different places) in North America, or DataGrid, CrossGrid in Europa or IrisGrid in España. These metacomputers or grid computers have a lot in common with parallel and distributed systems (SPD), but they are also different in certain important aspects. Although they are connected through networks, the networks can have different characteristics, the service cannot be guaranteed and they will be located in different domains. The programming model and interfaces must be radically different (in respect of the model of distributed systems) and adequate for high performance computing. As with SPD, the metacomputing applications require a communications plan to provide the required performance levels; but given their dynamic nature, new tools and techniques are needed. In other words, whilst metacomputing can be formed with the base of the SPDs, it is necessary to create new tools, mechanisms and techniques for these. [Fos]

3.1. Different computing architectures

If we only consider the calculative power aspect, we can see that there are various solutions depending on the size and characteristics of the problem. Firstly, we could think of a supercomputer (server) but these have problems such as the lack of scalability, costly equipment and maintenance, peak computing (a lot of time resources are not taken advantage of) and reliability problems. The economic alternative is a set of computers interconnected by a high performance network (Fast Ethernet – LAN – or Myrinet – SAN) which would form a cluster of stations dedicated to parallel/distributed computing (SPD) with a very high performance level (3 to 15 times cost/performance ratio). But these systems have inconveniences such as the high cost of communications, maintenance, programming model etc. However, it is an excellent solution for medium range or high time computing (HTC). Another interesting concept is intranet computing, which means using the equipment of a local network (for example, a C class network) to execute sequential or parallel jobs with assistance of an administration and load tool; In other words, it is a step down from a cluster and it permits the exploitation of the computational power in a large local network with the ensuing advantages, as we increase the effec-

tiveness of the use of resources (low cost CPU cycles), improve the scalability and the administration is not too complex. For these types of solutions, there is software such as Sun Grid Engine by Sun Microsystems [Sun], Condor by the University of Wisconsin (both free) [Uni] or LSF by Platform Computing (commercial) [Pla].

The option of intranet computing presents some inconveniences such as the impossibility of managing resources outside the domain of administration. Some of the abovementioned tools (Condor, LSF or SGE) permit cooperation between different sub-nodes of the system, but all of them must have the same administrative structure, the same security policies and the same philosophy of resource management. Although this is a step forward in terms of computational power at low-cost, they only manage the CPU and not the data shared between the sub-nodes. Besides, the protocols and interfaces are proprietary and they are not based on an open standard, it is not possible to amortise the resources when they are not fully in use and neither can we share resources with other organisations. [Beo, Ext, Die]

The growth of computers between 1986 and 2000 has multiplied by 500 and the networks by 340,000, but forecasts would indicate that, between 2001 and 2010, computers will only multiply by 60 and networks by 4,000. This indicates the standard of the next architecture for HPC: computing distributed by Internet or grid computing (GC) or metacomputing.

Grid computing is a new emerging technology, the objective of which is to share resources by Internet in a uniform, transparent, secure, efficient and reliable manner. This technology is complementary to the preceding technologies, in that it permits the interconnection of resources in different administration domains whilst respecting their internal security policies and their resource management software on the intranet. According to one of its precursors, Ian Foster, in his article "What is the Grid? A Three Point Checklist" (2002), a grid is a system that:

- 1) coordinates resources that are not subject to centralised control,
- 2) using standard, open, general-purpose protocols and interfaces,
- 3) to deliver non-trivial qualities of service.

Among the advantages that this new technology provides, we might mention the lease of resources, the amortisation of own resources, a great amount of power without having to invest in resources and installations, collaboration/sharing between institutions and virtual organisations etc.

The following figure provides a view of all these concepts. [Llo]

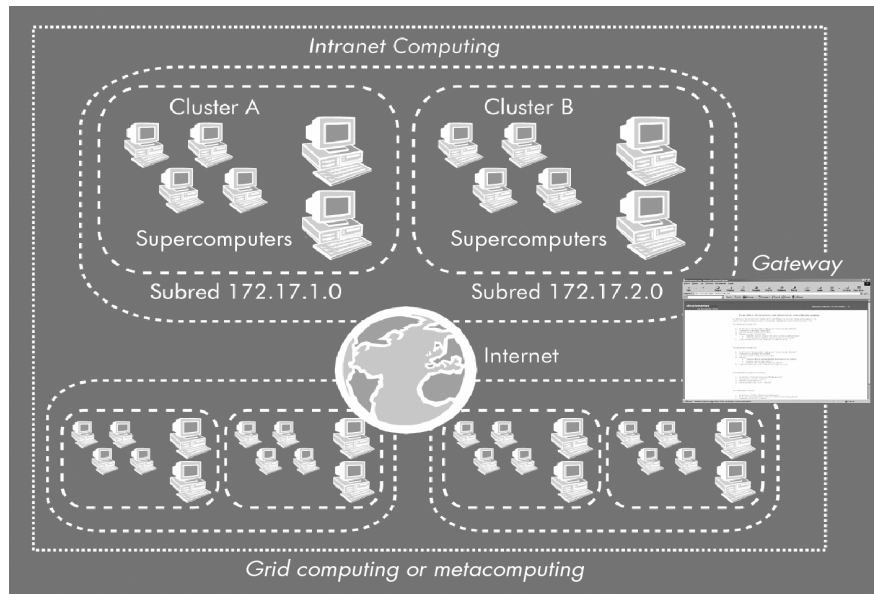


Figure 3

3.2. Globus

The Globus Project[Gloa, Glob] is one of the most emblematic in this sense, as it is the precursor in the development of a toolkit for metacomputing or grid computing and it provides considerable advances in the areas of communication, information, location and planning of resources, authentication and access to data. In other words, Globus makes it possible to share resources located in different administration domains, with different security and resource management policies and it is formed by a middleware software package that includes a set of libraries, services and API.

The *globus* tool (*Globus toolkit*) is formed by a set of modules with well-defined interfaces for interacting with other modules and/or services. The functions of these modules are as follows:

- Location and allocation of resources; this allows us to tell the applications what the requirements are and the resources that we need, given that an application cannot know where the resources on which it will execute are located.
- Communications; this provides the basic communication mechanisms, which represent an important aspect of the system, as they have to allow various methods for the applications to use them efficiently. These include message passing, remote procedure calls (RPC), shared distributed memory, (stream-based) dataflow and multicast.

- *Unified resource information service* provides a uniform mechanism for obtaining information in real time as to the status and structure of the meta-system where the applications are executing.
- Authentication interface; these are the basic authentication mechanisms for validating the identity of the users and resources. The module generates the upper layer that will then use the local services for accessing the data and resources of the system.
- Creation and execution of processes; this is used to start the execution of tasks that have been allocated to the resources, transmitting the execution parameters and controlling them until execution is completed.
- Data access; this has to provide high-speed access to the data saved in the files. For DB, it uses distributed access technology or through CORBA and it is able to achieve optimal performance levels when it accesses parallel file systems or in/out devices through the network, such as high performance storage system (HPSS).

The internal structure of Globus can be seen in the following figure (<http://www.globus.org/toolkit/about.html>).

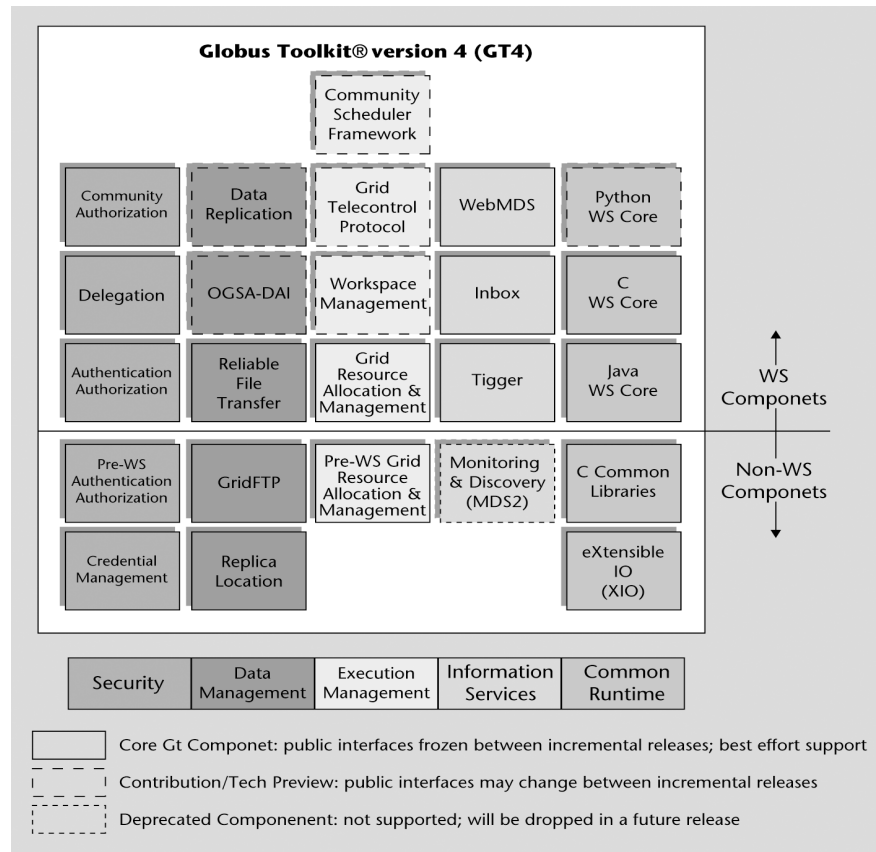


Figure 4

3.3. Software, installation and administration of Globus

The 'The Globus Alliance' website is <http://www.globus.org> [Gloa]. Here we can find source code and all the documents that we might need to transform our intranet into a part of a grid. Being part of a grid means agreeing to and implementing the policies of all the institutions and companies that are part of that grid. There are various different initiatives based on Globus in Spain. One of these is IrisGrid [Llo], which we can join if we wish to take advantage of the benefits of this technology. For more information, see: <http://www.rediris.es/irisgrid/>.

The first step for setting up Globus is to obtain the software (currently Globus Toolkit 4) called GT4. This software implements the services with a combination of C and Java (the C components can only be executed in UNIX GNU/Linux platforms, generally), which is why the software is divided into the services that it offers. Certain packages, or others, should be installed, depending on the system that we wish to set up.

A quick installation guide, with the download, system requirements and certificates can be found at <http://www.globus.org/toolkit/docs/4.0/admin/docbook/quickstart.html>. To summarise, the following steps must be taken:

- 1) Pre-requisites: verify the software and versions (zlib, j2se, disable gcj, apache, C/C++, tar, make, sed, perl, sudo, postgres, iodbc)
- 2) Create user, download and compile GT4
- 3) Start up system security (certificates)
- 4) Start up GridFTP
- 5) Start up the Webservices Container
- 6) Configure RFT (Reliable File Transfer)
- 7) Start up WS GRAM (job management)
- 8) Start up the second machine
- 9) Start up the Index Service hierarchy
- 10) Start up the cluster
- 11) Establish Cross-CA Trust

As you will observe, installing and setting up GT4 is not an easy task, but it is justified if we wish to incorporate a cluster into a *grid* or if we wish to perform tests (we recommend an extra dose of enthusiasm and patience) to appreciate the real power of GT4. For detailed information on installing GT4, please see:

<http://www.globus.org/toolkit/docs/4.0/admin/docbook/>

Activities

- 1) Install PVM on a node and execute the program master.c and client.c given as examples and observe their behaviour through xpmv.
- 2) Install and configure Mpich on a node; compile and execute the program cpi.c.
- 3) Install and configure LAM-MPI on a node; compile and execute the program cpi.c. and observe the behaviour through xmpi.

Bibliography

Other sources of reference and information

[Debc, Ibi, Mou01]

Lam-mpi: <http://www.lam-mpi.org/>

System-config-cluster (FC): http://www.redhat.com/docs/manuals/enterprise/RHEL-5-manual/Cluster_Administration/index.html

OpenMosix: <http://openmosix.sourceforge.net/>

HowTo Openmosix: <http://howto.x-tend.be/openMosix-HOWTO/>

Globus4: <http://www.globus.org/toolkit/docs/4.0/>

GT4 Quick Guide: <http://www.globus.org/toolkit/docs/4.0/admin/docbook/quickstart.html>

Bibliography

[Aiv02] **Tigran Aivazian** (2002). "Linux Kernel 2.4 Internals". *The Linux Documentation Project* (guías).

[Ano99] **Anonymous**. *Maximum Linux Security: A Hacker's Guide to Protecting*

[Apa] Apache2 + SSL.
<<http://www.debian-administration.org/articles/349>>

[Apab] Apache2 + WebDav
<<http://www.debian-administration.org/articles/285>>

[Apac] Apache2 + Subversion
<<http://www.debian-administration.org>>

[Ar01] **Jonathan Corbet; Alessandro Rubini**. *Linux Device Drivers 2nd Editon*. O'Reilly, 2001.

[Arc] **Roberto Arcomano**. "Kernel Analysis-HOWTO". *The Linux Documentation Project*.

[Aus] **CERT Australia**. "Australian CERT".
<<http://www.auscert.org.au/>>

[Bac86] **Maurice J. Bach** (1986). *The Design of the UNIX Operating System*. Prentice Hall.

[Bai03] **Edward C. Bailey** (2003). *RedHat Maximum RPM*.
<<http://www.redhat.com/docs/books/max-rpm/index.html>>

[Ban] **Tobby Banerjee**. "Linux Installation Strategies HOWTO". *The Linux Documentation Project*.

[Bar] **Slashdot**. *slashdot site*.
<<http://barrapunto.com>>

[Bas] **Mike G.** "BASH Programming - Introduction HOWTO". *The Linux Documentation Project*.

[Beo] **Beowulf.org**. *Beowulf Web Site*.
<<http://www.beowulf.org>>

[Bor] **Matthew Borowski** (2000). "FTP". *The Linux Documentation Project*.

[Bro] **Scott Bronson** (2001). "VPN PPP-SSH". *The Linux Documentation Project*.

[Bul] **Bulma**. "Bulma Linux User Group".
<<http://bulmalug.net>>

[Bur02] **Hal Burgiss** (2002). "Security QuickStart HOWTO for Linux". *The Linux Documentation Project*.

[Cac] Monitoring with Cacti.

<<http://cacti.net/>>

[Cdg] **Cedega**. (Environment for portability of GNU/Linux games)
<<http://www.transgaming.com/>>

[Ced] **Cederqvist**. "Version Management with CVS".
<<http://www.cvshome.org>>

[Cen] The Community ENTERprise Operatyng System
<<http://www.centos.org>>

[CERa] **CERT**. "CERT site".
<<http://www.cert.org>>

[CERb] **CERT** (2003). "CERT vulnerabilities".
<http://www.cert.org/nav/index_red.html>

[Cerc] **Cervisia**. "Cervisia interface for CVS".
<<http://cervisia.sourceforge.net>>

[Cis00] **Cisco** (2000). "TCP/IP White Paper".
<<http://www.cisco.com>>

[Com01] **Douglas Comer** (2001). *TCP/IP Basic principles, protocols and architecture* . Prentice Hall.

[Coo] **Mendel Cooper** (2006). "Advanced bashScripting Guide". *The Linux Documentation Project* (guías).

[CVS] **CVShome.org**. "CVS Home".
<<http://www.cvshome.org>>

[CVSI] Graphic interfaces for CVS
<<http://www.twobarleycorns.net/tkcv.html>>

[DBo] **Marco Cesati; Daniel Bovet** (2006). *Understanding the Linux Kernel* (3.^a ed.). O'Reilly.

[Deb] **Debian**. "Debian Security Site".
<<http://www.debian.org/security/>>

[Deb04] **Debian** (2004). "APT-HOWTO".
<<http://www.debian.org/doc/manuals/apt-howto/index.en.html>>

[Deba] **Debian**. "Free Software vs Open Software".
<<http://www.debian.org/intro/free.es.html>>

[Debb] **Comunidad Debian**. "Debian Distribution".
<<http://www.debian.org>>

[Dieb] **Hank Dietz** (2004). "Linux Parallel Processing". *The Linux Documentation Project*.

[Dis] **Distrowatch**. "Available Linux distributions".
<<http://www.distrowatch.com>>

[Dgn] The Dot Gnu Project.
<<http://www.gnu.org/software/dotgnu/>>

[DNS] Start up a DNS Server.
<<http://tldp.org/HOWTO/DNS-HOWTO-7.html>>

[Dra] **Joshua Drake** (1999). "Linux Networking". *The Linux Documentation Project*.

[DSL] **Digital Line Subscriber** (2002). *The Linux Documentation Project*.

[Buy] **Kris Buytaert and others** (2002). "The OpenMosix". *The Linux Documentation Project*.

[Ext] **Extremelinux.org**. "Extreme Linux Web Site".
<<http://www.extremelinux.org>>

[Exim] **Exim**. Mail service (MTA).

<<http://www.exim.org/docs.html>>

[FBI] FBI. "FBI Brigade for cybercrime".
<<http://www.emergency.com/fbi-nccs.htm>>

[Fed] The Fedora Project.
<<http://fedoraproject.org>>

[Fen02] **Kevin Fenzi**. "Linux security HOWTO". *The Linux Documentation Project*.

[Fos] **Ian Foster; Carl Kesselmany** (2003). "Globus: A Metacomputing Infrastructure Toolkit".
<<http://www.globus.org7gt>;

[Fre] **Freshmeat**. "Freshmeat site".
<<http://freshmeat.org>>

[Fri02] **Aleen Frisch** (2002). *Essential System Administration*. O'Reilly.

[Fry] Monitoring with Frysk.
<<http://sources.redhat.com/frysk/>>

[FSF] **FSF**. "Free Software Foundation and GNU Project".
<<http://www.gnu.org>>

[Gar98] **Bdale Garbee** (1998). *TCP/IP Tutorial*. N3EUA Inc.

[Gloa] **Globus. GT4**. "Admin Guide Installation" and "Admin Guide Configuration".
<<http://www.globus.org>>

[Glob] **Globus**. "User's Guide Core Framework Globus Toolkit ",
<<http://www.globus.org>>

[Gt] **Dirk Allaert Grant Taylor**. "The Linux Printing HOWTO". *The Linux Documentation Project*.

[GT4] Quick Guide.
<<http://www.globus.org/toolkit/docs/4.0/admin/docbook/quickstart.html>>

[Gnu] **Gnupg.org**. *GnuPG Web Site*.
<<http://www.gnupg.org/>>

[Gon] **Guido Gonzato**. "From DOS/Windows to Linux HOWTO". *The Linux Documentation Project*.

[Gor] **Paul Gortmaker** (2003). "The Linux BootPrompt HOWTO". *The Linux Documentation Project*.

[Gre] **Mark Grennan**. "Firewall and Proxy Server HOWTO". *The Linux Documentation Project*.

[Hat01] **Brian Hatch** (2001). *Hacking Linux Exposed*. McGraw-Hill.

[Hat03] **Red Hat** (2003). "Firewalls" en Red Hat 9 manual.
<<http://www.redhat.com/docs/manuals/linux/RHL-9-Manual/security-guide/ch-fw.html#S1-FIREWALL-IPT>>

[Hatb] **Red Hat** (2003). "Red Hat 9 Security Guide".
<<http://www.redhat.com/docs/manuals/linux/RHL-9-Manual/security-guide/>>

[Hatc] **Red Hat** (2003). "Red Hat Security Site".
<<http://www.redhat.com/security/>>

[Hatd] **Red Hat** (2003). *Use of GPG signatures in Red Hat*.
<<http://www.redhat.com/docs/manuals/linux/RHL-7.3-Manual/custom-guide/ch-gnupg.html>>

[Hen03] **Bryan Henderson**. "Linux Loadable Kernel Module HOWTO". *The Linux Documentation Project*.

[Him01] **Pekka Himanen** (2001). *Hacker ethics and the spirit of the information age*. Destination.

[Hin00] **Martin Hinner**. "Filesystems HOWTO". *The Linux Documentation Project*.

[His] **HispaLinux**. "Linux Hispanic Community".
<<http://www.hispalinux.es>>

[IET] **IETF**. "Request For Comment Repository developed by the Internet Engineering Task Force (IETF) in the Network Information Center (NIC)".
<<http://www.cis.ohio-state.edu/rfc/>>

[Ian] **Iana**. "List of TCP/IP ports".
<<http://www.iana.org/assignments/port-numbers>>

[IP] Routing with the ip tool. <ftp://ftp.inr.ac.ru/ip_routing/>

[ipw] Firmware for wireless cards IPW2200.
<<http://ipw2200.sourceforge.net/firmware.php>>

[Ibi] **Ibiblio.org** (2003). "Linux Documentation Center".
<<http://www.ibiblio.org/pub/Linux/docs/HOWTO/>>

[Inc] **Incidents.org**. "vulnerabilities Incidents".
<<http://isc.incidents.org>>

[Ins] **Insecure.org** (1998). "Vulnerabilities and exploits".
<<http://www.insecure.org/sploits.html>>

[Insa] **Insecure.org**. "Insecure.org site".
<<http://www.insecure.org>>

[Insb] **Insecure.org** (2003). "Nmap".
<<http://www.insecure.org/nmap/index.html>>

[Log] LogCheck.
<<http://logcheck.org/>>

[LWP] LWP: Apache+MySQL+PHP.
<http://www.lawebdelprogramador.com/temas/tema_stablephpapachemysql.php>

[Joh98] **Michael K. Johnson** (1998). "Linux Information Sheet". *The Linux Documentation Project*.

[Jou] **Linux Journal**. *Linux Journal [Linux Magazine]*.
<<http://www.linuxjournal.com>>

[Kan] **Ivan Kanis**. "Multiboot with GRUB Mini-HOWTO". *The Linux Documentation Project*.

[Kat] **Jonathan Katz**. "Linux + Windows HOWTO". *The Linux Documentation Project*.

[KD00] **Olaf Kirch; Terry Dawson**. *Linux Network Administrator's Guide*. O'Reilly Associates. And how *e-book* (free) in Free Software Foundation, Inc., 2000.
<<http://www.tldp.org/guides.html>>

[Ker02] **Kernelhacking.org** (2002). "Kernel Hacking Doc Project".
<<http://www.kernelhacking.org>>

[Kera] **Kernelnewbies.org**. "Kernel Newbies".
<<http://www.kernelnewbies.org>>

[Kerb] **Kernel.org**. "Linux Kernel Archives".
<<http://www.kernel.org>>

[Kie] **Robert Kiesling** (1997). "The RCS (Revision Control System)". *The Linux Documentation Project*.

[Knp] Knoppix Distribution.
<<http://knoppix.org>>

[Koe] Kristian Koehntopp. "Linux Partition HOWTO". *The Linux Documentation Project*.

[Kuk] **Thorsten Kukuk** (2003). "The Linux NIS(YP)/NYS/NIS+". *The Linux Documentation Project*.

- [Lam] **LamMPI.org**. "LAM (Local Area Multicomputer)".
<<http://www.lam-mpi.org>>
- [Law07] **David Lawyer** (2007). "Linux Modem". *The Linux Documentation Project*.
- [Lev02] **Bozidar Levi** (2002). *UNIX administration*. CRC Press.
- [Lev] **Eric Levenez**. "UNIX History".
<<http://www.levenez.com/unix>>
- [Lin03b] *FHS Standard*, 2003.
<<http://www.pathname.com/fhs>>
- [Linc] *Linux Standards Base project*.
<<http://www.linux-foundation.org/en/LSB>>
- [Line] **Linuxsecurity.com**. *Linux Security Reference Card*.
<<http://www.linuxsecurity.com/docs/QuickRefCard.pdf>>
- [lkm] **lkml**. *Linux Kernel Mailing List*.
<<http://www.tux.org/lkml>>
- [Llo] **Ignacio Martín Llorente**. *State of Grid Technology and IrisGrid Initiative*.
<<http://www.rediris.es/irisgrid>>
- [Lan] **Nicolai Langfeldt; Jamie Norrish** (2001). "DNS". *The Linux Documentation Project*.
- [Log] **Logcheck**. "Logcheck Web Site".
<<http://logcheck.org/>>
- [LPD] **LPD**. *The Linux Documentation Project*.
<<http://www.tldp.org>>
- [Mag] **Linux Magazine**. *Linux Magazine*.
<<http://www.linux-mag.com/>>
- [Maj96] **Amir Majidimehr** (1996). *Optimizing UNIX for Performance*. Prentice Hall.
- [Mal96] **Fred Mallett** (1996). *TCP/IP Tutorial*. FAME Computer Education.
- [Mal07] **Luiz Ernesto Pinheiro Malère** (2007). "Ldap". *The Linux Documentation Project*.
- [Miq] **Miquel, S.** "NIS Debian". *On Debian Woody, /usr/doc/nis/ nis.debian.howto*.
- [Moin] Moin Moin
<<http://moinmoin.wikiwikiweb.de/>>
- [Moi] Moin Moin + Debian.
<<http://moinmoin.wikiwikiweb.de/MoinMoinPackages/DebianLinux>>
- [Mon] Monit.
<<http://www.tildeslash.com/monit/>>
- [Monb] Monitoring with Munin and monit.
<http://www.howtoforge.com/server_monitoring_monit_munin>
- [Monc] Monitoring with SNMP and MRTG.
<http://www.linuxhomenetworking.com/wiki/index.php/>Quick_HOWTO_:_Ch22_:_Monitoring_Server_Performance>
- [Mono] Mono project.
<http://www.mono-project.com/Main_Page>
- [Mor03] **Daniel Morill** (2003). *Configuration of Linux systems*. Anaya Multimedia.
- [Mou01] **Gerhard Mourani** (2001). *Securing and Optimizing Linux: The Ultimate Solution*. Open Network Architecture, Inc.
- [Mun] Munin.
<<http://munin.projects.linpro.no/>>
- [MRTG] MRTG.

<<http://oss.oetiker.ch/mrtg/>>

[Mur] **Gary Lawrence Murphy**. *Kernel Book Project*.
<<http://kernelbook.sourceforge.net>>

[Mutt] Mutt mail client.
<<http://www.mutt.org>>

[Mys] **Mysql**. "Reference Manual".
<<http://www.mysql.com/>>

[MysqlA] **Mysql Administrator**.
<<http://www.mysql.com/products/tools/administrator/>>

[Nes] **Nessus.org**. "Nessus".
<<http://www.nessus.org>>

[Net] **Netfilter.org**. *Netfilter/IPtables Project*.
<www.netfilter.org>

[Neu] **Christopher Neufeld**. "Setting Up Your New Domain Mini-HOWTO". *The Linux Documentation Project*.

[New] **Newsforge**. "Newsforge site".
<<http://newsforge.org>>

[NIS] Setting up a NIS Server.
<<http://tldp.org/HOWTO/NIS-HOWTO/verification.html>>

[NSAa] **NSA**. "NIST site".
<<http://csrc.nist.gov/>>

[NSAb] **NSA** (2003). "Security Enhanced Linux".
<<http://www.nsa.gov/selinux>>

[Nt3] NTFS-3g Project: NTFS-3G Read/Write Driver.
<<http://www.ntfs-3g.org/>>

[Oke] **Greg O'Keefe**. "From Power Up To bash Prompt HOWTO". *The Linux Documentation Project*.

[Open] **OpenVPN**. Virtual private network.
<<http://openvpn.net/howto.html>>

[OpenM] OpenMosix.
<<http://openmosix.sourceforge.net/>>>

[OpenMb] HowTo Openmosix.
<<http://howto.x-tend.be/openMosix-HOWTO/>>

[OSDa] **OSDL**. "Open Source Developement Laboratories".
<<http://www.osdl.org>>

[OSDb]OSDN. "Open Source Development Network".
<<http://osdn.com>>

[OSIa] **OSI**. "List of Open Source licenses".
<<http://www.opensource.org/licenses/index.html>>

[OSIb] **OSI** (2003). "Open Source Definition".
<<http://www.opensource.org/docs/definition.php>>

[OSIc] **OSI** (2003). "Open Source Initiative".
<<http://www.opensource.org>>

[Peñ] **Javier Fernández-Sanguino Peña** (2007). "Securing Debian Manual".
<<http://www.debian.org/doc/manuals/securing-debian-howto/>>

[Pga] **PgAccess**. Client for PostgreSQL.
<<http://www.pgaccess.org/>>

[Pla] **Plataform**. "LSF".

<<http://www.platform.com>>

[Posa] **PostgreSQL.org**. "PostgreSQL Administrator's Guide".
<<http://www.postgresql.org/docs/>>

[Per] Performance Monitoring Tools for Linux.
<<http://www.linuxjournal.com/article.php?sid=2396>>

[Pose] **PostgreSQL.org**. "PostgreSQL Web Site".
<<http://www.postgresql.org>>

[PPP] **Linux PPP** (2000). "Corwin Williams, Joshua Drake and Robert Hart". *The Linux Documentation Project*.

[Pra03] **Joseh Pranevich** (2003). "The Wonderful World of Linux 2.6".
<<http://www.kniggit.net/wwol26.html>>

[Pri] **Steven Pritchard**. "Linux Hardware HOWTO". *The Linux Documentation Project*.

[Pro] **GNU Project**. "Grub Manual".
<<http://www.gnu.org/software/grub/manual/>>

[Proa] **Bastille Project**. "Bastille".
<<http://bastille-linux.sourceforge.net/>>

[Prob] **Mpich Project**. "MPI".
<<http://www.mcs.anl.gov:80/mpi/>>

[Proc] **Mpich Project**. "Mpich MPI Freeware".
<<http://www-unix.mcs.anl.gov/mpi/>>

[Prod] **OpenMosix Project**. "OpenMosix".
<<http://openMosix.sourceforge.net>>

[Proe] **PVM Project**. "PVM Web Site".
<<http://www.csm.ornl.gov/pvm/>>

[Proc] ProcMail.
<<http://www.debian-administration.org/articles/242>>

[ProX] Proxy Cache.
<<http://www.squid-cache.org/>>

[ProT] Transparent Proxy.
<<http://tldp.org/HOWTO/TransparentProxy-1.html>>

[Prof] ProFTP: FTP file server.
<<http://www.debian-administration.org/articles/228>>

[PS02] **Ricardo Enriquez Pio Sierra** (2002). *Open Source*. Anaya Multimedia.

[PurF] PureFTP: FTP file server.
<<http://www.debian-administration.org/articles/383>>

[Qui01] **Ellie Quigley** (2001). *Linux shells by Example*. Prentice Hall.

[Ran] **David Ranch** (2005). "Linux IP Masquerade" and **John Tapsell**. *Masquerading Made Simple*. The Linux Documentation Project.

[Ray98] **Eric Raymond** (1998). "The cathedral and the bazaar".
<<http://es.tldp.org/Otros/catedral-bazar/catedral-es-paper-00.html>>

[Ray02a] **Eric Raymond** (2002). "UNIX and Internet Fundamentals". *The Linux Documentation Project*.

[Rayb] **Eric Steven Raymond**. "The Linux Installation HOWTO". *The Linux Documentation Project*.

[Rad] **Jacek Radajewski; Douglas Eadline** (2002). "Beowulf: Installation and Administration". In: Kurt Swendson. *Beowulf HOWTO (tldp)*.
<<http://www.sci.usq.edu.au/staff/jacek/beowulf>>

[Red] Optimisation of Linux servers.
<http://people.redhat.com/alikins/system_tuning.html>

[Redb] System-config-cluster (FC).
<http://www.redhat.com/docs/manuals/enterprise/RHEL-5-manual/Cluster_Administration/index.htm>

[Redh] Red Hat Inc. "Red Hat Distribution".
<<http://www.redhat.com>>

[Rid] **Daniel Lopez Ridruejo** (2000). "The Linux Networking Overview". *The Linux Documentation Project*.

[Rus] **Rusty Russell**. "Linux IPCHAINS". *The Linux Documentation Project*.

[SM02] **Michael Schwartz and other** (2002). *Multitool Linux - Practical Uses for Open Source Software*. Addison Wesley.

[Sal94] **Peter H. Salus** (1994). "25th anniversary of UNIX" (no. 1, November). *Byte Spain*.

[Sam] Samba Project.
<<http://samba.org>>

[Sama] Samba HOWTO and Reference Guide (Chapter Domain Control).
<<http://samba.org/samba/docs/man/Samba-HOWTO-Collection/samba-pdc.html>>

[Samb] Samba Guide (Chapter Adding Domain member Servers and Clients).
<<http://samba.org/samba/docs/man/Samba-Guide/unixclients.html>>

[San] **Sans**. "Top20 vulnerabilities".
<<http://www.sans.org/top20/>>

[Sci] Scientific Linux.
<<http://www.scientificlinux.org>>

[Sec] **Andrés Seco** (2000). "Diald". *The Linux Documentation Project*.

[Sei02] **Kurt Seifried** (2002). "Securing Linux, Step by Step".
<<http://seifried.org/security/os/linux/20020324-securing-linux-step-by-step.html>>

[Skoa] **Miroslav Skoric**. "LILO mini-HOWTO". *The Linux Documentation Project*.

[Skob] **Miroslav Skoric**. "Linux+WindowsNT mini-HOWTO". *The Linux Documentation Project*.

[Sla] **Slashdot**. "Slashdot site".
<<http://slashdot.org>>

[Smb] Wikipedia entry for "Server Message Block".
<http://en.wikipedia.org/wiki/Server_Message_Block>

[Smi02] **Rod Smith** (2002). *Advanced Linux Networking*. Addison Wesley.

[Sno] **Snort.org**. *Snort*.
<<http://www.snort.org>>

[Sou] **Sourceforge**. "Sourceforge site".
<<http://sourceforge.org>>

[Squ] Squid proxy server.
<<http://www.squid-cache.org/>>

[Sta02] **Richard Stallman** (2002). "Discussion by Richard Stallman on relationship between GNU and Linux".
<<http://www.gnu.org/gnu/linux-and-gnu.html>>

[Stu] **Michael Stutz**. "The Linux Cookbook: Tips and Techniques for Everyday Use". *The Linux Documentation Project* (guías).

[Ste07] Steve French, Linux CIFS Client guide.
<<http://us1.samba.org/samba/ftp/cifs-cvs/linux-cifs-client-guide.pdf>>

[Steij] **Tony Steidler-Dennison** (2005). *Your Linux Server and Network*. Sams.

[Sub] Subversion.
<<http://subversion.tigris.org>>

[Subb] Control of versions with Subversion. Free Book.
<<http://svnbook.red-bean.com/index.es.html>>

*[Sun02] **Rahul Sundaram** (2002). "The dosemu HOWTO". *The Linux Documentation Project*.

[Sun] **Sun**. "Sun Grid Engine".
<<http://www.sun.com/software/gridware/>>

[Tan87] **Andrew Tanenbaum** (1987). *Operating system: Design and Implementation*. Prentice Hall.

[Tan06] **Andrew Tanenbaum; Albert S. Woodhull** (2006). *The Minix Book: Operating Systems Design and Implementation* (3rd ed.). Prentice Hall.

[Tkc] **Tkcv**s (2003). "Tkcv's interface for CVS".
<<http://www.tkcv.org>>
<<http://www.twobarleycorns.net/tkcv.html>>

[Tri] **Tripwire.com**. *Tripwire Web Site*.
<<http://www.tripwire.com/>>

[Tum02] **Enkh Tumenbayar** (2002). "Linux SMP HOWTO". *The Linux Documentation Project*.

[Ubn] Ubuntu Distribution.
<<http://www.ubuntu.com>>

[Uni] **Wisconsin University** (2003). *Condor Web Site*.
<<http://www.cs.wisc.edu/condor>>

[USA] **Dep. Justice USA**. "Division of the US Justice Department for cybercrime".
<<http://www.usdoj.gov/criminal/cybercrime/>>

[Vah96] **Uresh Vahalia** (1996). *UNIX Internals: The New Frontiers*. Prentice Hall.

[Vas] **Alavoor Vasudevan** (2000). "Modem-Dialup-NT". *The Linux Documentation Project*.

[Vasa] **Alavoor Vasudevan** (2003). "CVS-RCS (Source Code Control System)". *The Linux Documentation Project*.

[Vasb] **Alavoor Vasudevan**. "The Linux Kernel HOWTO". *The Linux Documentation Project*.

[Wm02] **Matt Welsh and others** (2002). *Running Linux 4th edition*. O'Reilly.

[War] **Ian Ward**. "Debian and Windows Shared Printing mini-HOWTO". *The Linux Documentation Project*.

[Web] **Webmin**. *Tool for administrating Linux systems*.
<<http://www.webmin.com/>>

[Wil02] **Matthew D. Wilson** (2002). "VPN". *The Linux Documentation Project*.

[Win] Wine Project.
<<http://www.winehq.com/>>

[Wir] WireShark.
<<http://www.wireshark.org/download.html>>

[Woo] **David Wood**. "SMB HOWTO". *The Linux Documentation Project*.

[Xin] Xinetd Web Site.
<<http://www.xinetd.org/>>

[Zan] **Renzo Zanelli**. *Win95 + WinNT + Linux multiboot using LILOmini-HOWTO*. *The Linux Documentation Project*.

GNU Free Documentation License

Version 1.2, November 2002

Copyright (C) 2000,2001,2002 Free Software Foundation, Inc. 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondly, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent.

An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or

XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language (here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History"). To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly

and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition.

Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material.

If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties--for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number.

Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit.

When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form.

Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright hold-

ers, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See [http:// www.gnu.org/copyleft/](http://www.gnu.org/copyleft/).

Each version of the License is given a distinguishing version number.

If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the

Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

Copyright (c) YEAR YOUR NAME.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts.

A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with...Texts." line with this:

with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

WITH SUPPORT FROM THE



Education and Culture DG

Lifelong Learning Programme

THE GNU/LINUX SYSTEMS HAVE REACHED AN IMPORTANT LEVEL OF MATURITY, ALLOWING TO INTEGRATE THEM IN ALMOST ANY KIND OF WORK ENVIRONMENT, FROM A DESKTOP PC TO THE SERVER FACILITIES OF A BIG COMPANY.

IN THE MODULE CALLED "THE GNU/LINUX OPERATING SYSTEM", THE MAIN CONTENTS ARE RELATED WITH SYSTEM ADMINISTRATION. THIS BOOK IS THE MAIN DOCUMENTATION FOR THE MODULE

WE WILL LEARN HOW TO INSTALL AND CONFIGURE SEVERAL COMPUTER SERVICES, AND HOW TO OPTIMISE AND SYNCHRONISE THE RESOURCES.

THE ACTIVITIES THAT WILL TAKE PLACE IN THIS MODULE COVER THE STUDIED TOPICS IN A PRACTICAL APPROACH, APPLYING THESE CONCEPTS IN REAL GNU/LINUX SYSTEMS.

