Version 4.0

Linux Administration I

System and Users

tuxcademy – Linux and Open Source learning materials for everyone
www.tuxcademy.org · info@tuxcademy.org

This training manual is designed to correspond to the objectives of the LPI-101 (LPIC-1, version 4.0) certification exam promulgated by the Linux Professional Institute. *Further details are available in Appendix B.*

The Linux Professional Institute *does not endorse specific exam preparation materials or techniques. For details, refer to* `info@lpi.org`.

The tuxcademy project aims to supply freely available high-quality training materials on Linux and Open Source topics – for self-study, school, higher and continuing education and professional training.
Please visit `http://www.tuxcademy.org/`! Do contact us with questions or suggestions.

Linux Administration I **System and Users**
Revision: `adm1:067839db3bb6bd7f:2015-08-08`
`adm1:33e55eeadba676a3:2015-08-08` 1–13, B
`adm1:HWye5vypKrdhKjDySGhqPI`

All representations and information contained in this document have been compiled to the best of our knowledge and carefully tested. However, mistakes cannot be ruled out completely. To the extent of applicable law, the authors and the tuxcademy project assume no responsibility or liability resulting in any way from the use of this material or parts of it or from any violation of the rights of third parties. Reproduction of trade marks, service marks and similar monikers in this document, even if not specially marked, does not imply the stipulation that these may be freely usable according to trade mark protection laws. All trade marks are used without a warranty of free usability and may be registered trade marks of third parties.

Authors: Thomas Erker, Anselm Lingnau
Technical Editor: Anselm Lingnau ⟨*anselm@tuxcademy.org*⟩
English Translation: Anselm Lingnau
Typeset in Palatino, Optima and DejaVu Sans Mono

Contents

List of Tables

List of Figures

Preface

This manual is an introduction to Linux system administration. Based on knowledge about using Linux, it conveys the most important theoretical and practical knowledge for the setup and operation of a standalone Linux-based computer.

The manual is geared towards users who have knowledge and experience using Linux or Unix systems at a level comparable to the tuxcademy manual *Introduction to Linux for Users and Administrators* and are looking for a compact but extensive introduction to system administration. Prerequisites are confidence using the shell and a text editor as well as experience with the common command line tools of a Linux system.

This manual covers, after an introduction to the significance and problems of system administration, the basics of process, user account, and access control management, the management of disk partitions, file systems, and quotas, common boot loaders, the system start and shutdown process, PC hardware, and library and package management.

The successful completion of this course or comparable knowledge are necessary in order to take part in further Linux study and to obtain *Linux Professional Institute* certification.

This courseware package is designed to support the training course as efficiently as possible, by presenting the material in a dense, extensive format for reading along, revision or preparation. The material is divided in self-contained chapters detailing a part of the curriculum; a chapter's goals and prerequisites are summarized clearly at its beginning, while at the end there is a summary and (where appropriate) pointers to additional literature or web pages with further information.

chapters
goals
prerequisites

Additional material or background information is marked by the "lightbulb" icon at the beginning of a paragraph. Occasionally these paragraphs make use of concepts that are really explained only later in the courseware, in order to establish a broader context of the material just introduced; these "lightbulb" paragraphs may be fully understandable only when the courseware package is perused for a second time after the actual course.

Paragraphs with the "caution sign" direct your attention to possible problems or issues requiring particular care. Watch out for the dangerous bends!

Most chapters also contain exercises, which are marked with a "pencil" icon at the beginning of each paragraph. The exercises are numbered, and sample solutions for the most important ones are given at the end of the courseware package. Each exercise features a level of difficulty in brackets. Exercises marked with an exclamation point ("!") are especially recommended.

exercises

Excerpts from configuration files, command examples and examples of computer output appear in `typewriter type`. In multiline dialogs between the user and the computer, user input is given in **`bold typewriter type`** in order to avoid misunderstandings. The "◁◁◁◁◁" symbol appears where part of a command's output had to be omitted. Occasionally, additional line breaks had to be added to make things fit; these appear as "▷

◁". When command syntax is discussed, words enclosed in angle brackets ("⟨*Word*⟩") denote "variables" that can assume different values; material in brackets ("[`-f` ⟨*file*⟩]") is optional. Alternatives are separated using a vertical bar ("`-a`|`-b`").

Important concepts

definitions

Important concepts are emphasized using "marginal notes" so they can be easily located; **definitions** of important terms appear in bold type in the text as well as in the margin.

References to the literature and to interesting web pages appear as "[GPL91]" in the text and are cross-referenced in detail at the end of each chapter.

We endeavour to provide courseware that is as up-to-date, complete and error-free as possible. In spite of this, problems or inaccuracies may creep in. If you notice something that you think could be improved, please do let us know, e.g., by sending e-mail to

```
info@tuxcademy.org
```

(For simplicity, please quote the title of the courseware package, the revision ID on the back of the title page and the page number(s) in question.) Thank you very much!

LPIC-1 Certification

These training materials are part of a recommended curriculum for LPIC-1 preparation. Refer to Appendix B for further information.

1

System Administration

Contents

Goals

- Reviewing a system administrator's tasks
- Being able to log on as the system administrator
- Being able to assess the advantages and disadvantage of (graphical) administration tools

Prerequisites

- Basic Linux skills
- Administration skills for other operating systems are helpful

1.1 Introductory Remarks

As a mere user of a Linux system, you are well off: You sit down in front of your computer, everything is configured correctly, all the hardware is supported and works. You have no care in the world since you can call upon a system administrator who will handle all administrative tasks for you promptly and thoroughly (that's what we wish your environment is like, anyway).

Should you be (or strive to be) the system administrator yourself—within your company or the privacy of your home—then you have your work cut out for you: You must install and configure the system and connect any peripherals. Having done that, you need to keep the system running, for example by checking the system logs for unusual events, regularly getting rid of old log files, making backup copies, installing new software and updating existing programs, and so on.

changes

Today, in the age of Linux distributions with luxurious installation tools, system installation is no longer rocket science. However, an ambitious administrator can spend lots of time mobilising every last resource on their system. In general, system administration mostly takes place when a noticeable change occurs, for example when new hardware or software is to be integrated, new users arrive or existing ones disappear, or hardware problems arise.

Tools

💡 Many Linux distributions these days contain specialised tools to facilitate system administration. These tools perform different tasks ranging from user management and creating file systems to complete system updates. Utilities like these can make these tasks a lot easier but sometimes a lot more difficult. Standard procedures are simplified but for specialised settings you should know the exact relationships between system components. Furthermore, most of these tools are only available for certain distributions.

responsibility

communication

The administration of a Linux system, as of any other computer system, requires a considerable amount of responsibility and care. You should not see yourself as a demigod (at least) but as a service provider. No matter whether you are the only system administrator—say, on your own computer—or working in a team of colleagues to support a company network: communication is paramount. You should get used to documenting configuration changes and other administrative decisions in order to be able to retrace them later. The Linux way of directly editing text files makes this convenient, since you can comment configuration settings right where they are made (a luxury not usually enjoyed with graphical administration tools). Do so.

1.2 The Privileged `root` Account

super user

For many tasks, the system administrator needs special privileges. Accordingly, he can make use of a special user account called `root`. As `root`, a user is the so-called **super user**. In brief: He may do anything.

unlimited privileges

The normal file permissions and security precautions do not apply to `root`. He has allowing him nearly unbounded access to all data, devices and system components. He can institute system changes that all other users are prohibited from by the Linux kernel's security mechanisms. This means that, as `root`, you can change every file on the system no matter who it belongs to. While normal users cannot wreak damage (e. g., by destroying file systems or manipulating other users' files), `root` is not thus constrained.

💡 In many cases, these extensive system administrator privileges are really a liability. For example, when making backup copies it is necessary to be able to read all files on the system. However, this by no means implies that the person making the backup (possibly an intern) should be empowered to open all files on the system with a text editor, to read them or change them—or start a network service which might be accessible from anywhere in the

> world. There are various ways of giving out administrator privileges only in controlled circumstances (such as `sudo`, a system which lets normal users execute certain commands using administrator privileges), of selectively giving particular privileges to individual process rather than operating on an "all or nothing" principle (cue POSIX capabilities), or of doing away with the idea of an "omnipotent" system administrator completely (for instance, SELinux—"security-enhanced Linux"—a freely available software package by the American intelligence agency, NSA, contains a "role-based" access control system that can get by without an omnipotent system administrator).

`sudo`

POSIX capabilities

SELinux

Why Security?

Why does Linux contain security precautions in the first place? The most important reason is for users to be able to determine the access privileges that apply to their own files. By setting permission bits (using the `chmod` command), users can ascertain that certain files may be read, written to or executed by certain others (or no) users. This helps safeguard the privacy and integrity of their data. You would certainly not approve of other users being able to read your private e-mail or change the source code of an important program behind your back.

Access control for devices

The security mechanisms are also supposed to keep users from damaging the system. Access to many of the device files in `/dev` corresponding to hardware components such as hard disks is constrained by the system. If normal users could access disk storage directly, all sorts of mayhem might occur (a user might overwrite the complete content of a disk or, having obtained information about the layout of the filesystem on the disk, access files that are none of his business). Instead, the system forces normal users to access the disks via the file system and protects their data in that way.

It is important to stress that damage is seldom caused on purpose. The system's security mechanisms serve mostly to save users from unintentional mistakes and misunderstandings; only in the second instance are they meant to protect the privacy of users and data.

groups

On the system, users can be pooled into **groups** to which you may assign their own access privileges. For example, a team of software developers could have read and write permission to a number of files, while other users are not allowed to change these files. Every user can determine for their own files how permissive or restrictive access to them should be.

Privileged system calls

The security mechanisms also prevent normal users from performing certain actions such as the invocation of specific system calls from a program. For example, there is a system call that will halt the system, which is executed by programs such as `shutdown` when the system is to be powered down or rebooted. If normal users were allowed to invoke this routine from their own programs, they could inadvertently (or intentionally) stop the system at any time.

The administrator frequently needs to circumvent these security mechanisms in order to maintain the system or install updated software versions. The `root` account is meant to allow exactly this. A good administrator can do his work without regard for the usual access permissions and other constraints, since these do not apply to `root`. The `root` account is not better than a normal user account because it has more privileges; the restriction of these privileges to `root` is a security measure. Since the operating system's reasonable and helpful protection and security mechanisms do not apply to the system administrator, working as `root` is very risky. You should therefore use `root` to execute only those commands that really require the privileges.

> Many of the security problems of other popular operating systems can be traced back to the fact that normal users generally enjoy administrator privileges. Thus, programs such as "worms" or "Trojan horses", which users often execute by accident, find it easy to establish themselves on the system. With a Linux system that is correctly installed and operated, this is hardly possible since users read their e-mail without administrator privi-

leges, but administrator privileges are required for all system-wide configuration changes.

Of course, Linux is not magically immune against malicious pests like “mail worms”; somebody could write and make popular a mail program that would execute “active content” such as scripts or binary programs within messages like some such programs do on other operating systems. On Linux, such a “malicious” program from elsewhere could remove all the caller’s files or try to introduce “Trojan” code to his environment, but it could not harm other users nor the system itself—unless it exploited a security vulnerability in Linux that would let a local user gain administrator privileges “through the back door” (such vulnerabilities are detected now and again, and patches are promptly published which you should install in a timely manner).

Exercises

1.1 [2] What is the difference between a user and an administrator? Name examples for tasks and actions (and suitable commands) that are typically performed from a user account and the `root` account, respectively.

1.2 [!1] Why should you, as a normal user, not use the `root` account for your daily work?

1.3 [W]hat about access control on your computer at home? Do you work from an administrator account?

1.3 Obtaining Administrator Privileges

There are two ways of obtaining administrator privileges:

1. You can log in as user `root` directly. After entering the correct `root` password you will obtain a shell with administrator privileges. However, you should avoid logging in to the GUI as `root`, since then all graphical applications including the X server would run with `root` privileges, which is not necessary and can lead to security problems. Nor should direct `root` logins be allowed across the network.

   You can determine which terminals are eligible for direct `root` login by listing them in the `/etc/securetty` file. The default setting is usually “all virtual consoles and `/dev/ttyS0`” (the latter for users of the “serial console”).

2. You can, from a normal shell, use the `su` command to obtain a new shell with administrator privileges. `su`, like `login`, asks for a password and opens the `root` shell only after the correct `root` password has been input. In GUIs like KDE there are similar methods.

(See also *Introduction to Linux for Users and Administrators*.)

Single-user systems, too! Even if a Linux system is used by a single person only, it makes sense to create a normal account for this user. During everyday work on the system as `root`, most of the kernel’s normal security precautions are circumvented. That way errors can occur that impact on the whole system. You can avoid this danger by logging into your normal account and starting a `root` shell via “`/bin/su -`” if and when required.

Using `su`, you can also assume the identity of arbitrary other users (here `hugo`) by invoking it like

```
$ /bin/su - hugo
```

You need to know the target user's password unless you are calling su as user root.

The second method is preferable to the first for another reason, too: If you use the su command to become root after logging in to your own account, su creates a message like

```
Apr  1 08:18:21 HOST su: (to root) user1 on /dev/tty2
```

system log

in the system log (such as /var/log/messages). This entry means that user user1 successfully executed su to become root on terminal 2. If you log in as root directly, no such message is logged; there is no way of figuring out which user has fooled around with the root account. On a system with several administrators it is often important to retrace who entered the su command when.

Ubuntu is one of the "newfangled" distributions that deprecate–and, in the default setup, even disable—logging in as root. Instead, particular users may use the sudo mechanism to execute individual commands with administrator privileges. Upon installation, you are asked to create a "normal" user account, and that user account is automatically endowed with "indirect" administrator privileges.

When installing Debian GNU/Linux, you can choose between assigning a password to the root account and thereby enabling direct administrator logins, and declining this and, as on Ubuntu, giving sudo-based administrator privileges to the first unprivileged user account created as part of the installation process.

shell prompt

On many systems, the shell prompt differs between root and the other users. The classic root prompt contains a hash mark (#), while other users see a prompt containing a dollar sign ($) or greater-than sign (>). The # prompt is supposed to remind you that you are root with all ensuing privileges. However, the shell prompt is easily changed, and it is your call whether to follow this convention or not.

Of course, if you are using sudo, you never get to see a prompt for root.

Misuse of root

Like all powerful tools, the root account can be abused. Therefore it is important for you as the system administrator too keep the root password secret. It should only be passed on to users who are trusted both professionally and personally (or who can be held responsible for their actions). If you are the sole user of the system this problem does not apply to you.

Administration: alone or by many

Too many cooks spoil the broth! This principle also applies to system administration. The main benefit of "private" use of the root account is not that the possibility of misuse is minimised (even though this is surely a consequence). More importantly, root as the sole user of the root account knows the complete system configuration. If somebody besides the administrator can, for example, change important system files, then the system configuration could be changed without the administrator's knowledge. In a commercial environment, it is necessary to have several suitably privileged employees for various reasons—for example, safeguarding system operation during holidays or sudden severe illness of the administrator—; this requires close cooperation and communication.

accountability

If there is only one system administrator who is responsible for system configuration, you can be sure that one person really knows what is going on on the system (at least in theory), and the question of accountability also has an obvious asnwer. The more users have access to root, the greater is the probability that somebody will commit an error as root at some stage. Even if all users with root access possess suitable administration skills, mistakes can happen to anybody. Prudence and thorough training are the only precautions against accidents.

There are a few other useful tools for team-based system administration. For example, Debian GNU/Linux and Ubuntu support a package called `etckeeper`, which allows storing the complete content of the `/etc` directory in a revision control system such as Git or Mercurial. Revision control systems (which we cannot cover in detail here) make it possible to track changes to files in a directory hierarchy in a very detailed manner, to comment them and, if necessary, to undo them. With Git or Mercurial it is even possible to store a copy of the `/etc` directory on a completely different computer and to keep it in sync automatically—great protection from accidents.

Exercises

1.4 [2] What methods exist to obtain administrator rights? Which method is better? Why?

1.5 [!2] On a conventionally configured system, how can you recognise whether you are working as `root`?

1.6 [2] Log in as a normal user (e. g., `test`). Change over to `root` and back to `test`. How do you work best if you frequently need to change between both these accounts (for example, to check on the results of a new configuration)?

1.7 [!2] Log in as a normal user and change to `root` using `su`. Where do you find a log entry documenting this change? Look at that message.

1.4 Distribution-specific Administrative Tools

Many Linux distributions try to stand out in the crowd by providing more or less ingenious tools that are supposed to simplify system administration. These tools are usually tailored to the distributions in question. Here are a few comments about typical specimens:

A familiar sight to SUSE administrators is "YaST", the graphical administration interface of the SUSE distributions (it also runs on a text screen). It allows the extensive configuration of many aspects of the system either by directly changing the configuration files concerned or by manipulating abstract configuration files below `/etc/sysconfig` which are then used to adapt the real configuration files by means of the `SuSEconfig` tool. For some tasks such as network configuration, the files below `/etc/sysconfig` *are* the actual configuration files.

Unfortunately, YaST is not a silver bullet for all problems of system administration. Even though many aspects of the system are amenable to YaST-based administration, important settings may not be accessible via YaST, or the YaST modules in question simply do not work correctly. The danger zone starts where you try to administer the computer partly through YaST and partly through changing configuration files manually: Yast does exercise some care not to overwrite your changes (which wasn't the case in the past—up till SuSe 6 or so, YaST and SuSEconfig used to be quite reckless), but will then not perform its own changes such that they really take effect in the system. In other places, manual changes to the configuration files will actually show up in YaST. Hence you have to have some "insider knowledge" and experience in order to assess which configuration files you may change directly and which your grubby fingers had better not touch.

Some time ago, Novell released the YaST source code under the GPL (in SUSE's time it used to be available but not under a "free" licence). However, so far no other distribution of consequence has adapted YaST to its purposes, let alone made it a standard tool (SUSE fashion).

The Webmin package by Jamie Cameron (`http://www.webmin.com/`) allows the convenient administration of various Linux distributions (or Unix versions) via a web-based interface. Webmin is very extensive and offers special facilities for administering "virtual" servers (for web hosters and their customers). However you may have to install it yourself, since most distributions do not provide it. Webmin manages its own users, which means that you can extend administrator privileges to users who do not have interactive system access. (Whether that is a smart idea is a completely different question.)

Most administration tools like YaST and Webmin share the same disadvantages:

- They are not extensive enough to take over all aspects of system administrations, and as an administrator you have to have detailed knowledge of their limits in order to be able to decide where to intervene manually.

- They make system administration possible for people whose expertise is not adequate to assess the possible consequences of their actions or to find and correct mistakes. Creating a user account using an administration tool is certainly not a critical job and surely more convenient than editing four different system files using `vi`, but other tasks such as configuring a firewall or mail server are not suitable for laypeople even using a convenient administration tool. The danger is that inexperienced administrators will use an administration tool to attempt tasks which do not look more complicated than others but which, without adequate background knowledge, may endanger the safety and/or reliability of the system.

- They usually do not offer a facility to version control or document any changes made, and thus complicate teamwork and auditing by requiring logs to be kept externally.

- They are often intransparent, i. e., they do not provide documentation about the actual steps they take on the system to perform administrative tasks. This keeps the knowledge about the necessary procedures buried in the programs; as the administrator you have no direct way of "learning" from the programs like you could by observing an experienced administrator. Thus the adminstration tools keep you artificially stupid.

- As an extension of the previous point: If you need to administer several computers, common administration tools force you to execute the same steps repeatedly on every single machine. Many times it would be more convenient to write a shell script automating the required procedure, and to execute it automatically on every computer using, e. g., the "secure shell", but the administration tool does not tell you what to put into this shell script. Therefore, viewed in a larger context, their use is inefficient.

From various practical considerations like these we would like to recommend against relying too much on the "convenient" administration tools provided by the distributions. They are very much like training wheels on a bicycle: They work effectively against falling over too early and provide a very large sense of achievement very quickly, but the longer the little ones zoom about with them, the more difficult it becomes to get them used to "proper" bike-riding (here: doing administration in the actual configuration files, including all advantages such as documentation, transparency, auditing, team capability, transportability, …).

Excessive dependence on an administration tool also leads to excessive dependence on the distribution featuring that tool. This may not seem like a real liability, but on the other hand one of the more important *advantages* of Linux is the fact that there are multiple independent vendors. So, if one day you should be fed up with the SUSE distributions (for whatever reason) and want to move over to Red Hat or Debian GNU/Linux, it would be very inconvenient if your administrators knew only YaST and had to relearn Linux administration from scratch. (Third-party administration tools like Webmin do not exhibit this problem to the same degree.)

Exercises

1.8 [!2] Does your distribution provide an administration tool (such as YaST)? What can you do with it?

1.9 [3] (Continuation of the previous exercise—when working through the manual for the second time.) Find out how your administration tool works. Can you change the system configuration manually so the administration tool will notice your changes? Only under some circumstances?

1.10 [!1] Administration tools like Webmin are potentially accessible to everybody with a browser. Which advantages and disadvantages result from this?

Summary

- Every computer installation needs a certain amount of system administration. In big companies, universities and similar institutions these services are provided by (teams of) full-time administrators; in smaller companies or private households, (some) users usually serve as administrators.
- Linux systems are, on the whole, straightforward to administer. Work arises mostly during the initial installation and, during normal operation, when the configuration changes noticeably.
- On Linux systems, there usually is a privileged user account called `root`, to which the normal security mechanisms do not apply.
- As an administrator, one should not work as `root` exclusively, but use a normal user account and assume `root` privileges only if necessary.
- Administration tools such as YaST or Webmin can help perform some administrative duties, but are no substitute for administrator expertise and may have other disadvantages as well.

2

User Administration

Contents

Goals

- Understanding the user and group concepts of Linux
- Knowing how user and group information is stored on Linux
- Being able to use the user and group administration commands

Prerequisites

- Knowledge about handling configuration files

2.1 Basics

2.1.1 Why Users?

Computers used to be large and expensive, but today an office workplace without its own PC ("personal computer") is nearly inconceivable, and a computer is likely to be encountered in most domestic "dens" as well. And while it may be sufficient for a family to agree that Dad, Mom and the kids will put their files into different directories, this will no longer do in companies or universities—once shared disk space or other facilities are provided by central servers accessible to many users, the computer system must be able to distinguish between different users and to assign different access rights to them. After all, Ms Jones from the Development Division has as little business looking at the company's payroll data as Mr Smith from Human Resources has accessing the detailed plans for next year's products. And a measure of privacy may be desired even at home—the Christmas present list or teenage daughter's diary (erstwhile fitted with a lock) should not be open to prying eyes as a matter of course.

We shall be discounting the fact that teenage daughter's diary may be visible to the entire world on Facebook (or some such); and even if that is the case, the entire world should surely not be allowed to *write* to teenage daughter's dairy. (Which is why even Facebook supports the notion of different users.)

The second reason for distinguishing between different users follows from the fact that various aspects of the system should not be visible, much less changeable, without special privileges. Therefore Linux manages a separate user identity (`root`) for the system administrator, which makes it possible to keep information such as users' passwords hidden from "common" users. The bane of older Windows systems—programs obtained by e-mail or indiscriminate web surfing that then wreak havoc on the entire system—will not plague you on Linux, since anything you can execute as a common user will not be in a position to wreak system-wide havoc.

Unfortunately this is not entirely correct: Every now and then a bug comes to light that enables a "normal user" to do things otherwise restricted to administrators. This sort of error is extremely nasty and usually corrected very quickly after having been found, but there is a considerable chance that such a bug has remained undetected in the system for an extended period of time. Therefore, on Linux (as on all other operating systems) you should strive to run the most current version of critical system parts like the kernel that your distributor supports.

Even the fact that Linux safeguards the system configuration from unauthorised access by normal users should not entice you to shut down your brain. We do give you some advice (such as not to log in to the graphical user interface as `root`), but you should keep thinking along. E-mail messages asking you to view web site *X* and enter your credit card number and PIN there can reach you even on Linux, and you should disregard them in the same way as everywhere else.

user accounts

Linux distinguishes between different users by means of different **user accounts**. The common distributions typically create two user accounts during installation, namely `root` for administrative tasks and another account for a "normal" user. You (as the administrator) may add more accounts later, or, on a client PC in a larger network, they may show up automatically from a user account database stored elsewhere.

Linux distinguishes between *user accounts*, not users. For example, no one keeps you from using a separate user account for reading e-mail and surfing the web, if you want to be 100% sure that things you download from the

Net have no access to your important data (which might otherwise happen in spite of the user/administrator divide). With a little cunning you can even display a browser and e-mail program running under your "surfing account" among your "normal" programs[1].

Under Linux, every user account is assigned a unique number, the so-called *user ID* (or **UID**, for short). Every user account also features a textual **user name** (such as root or joe) which is easier to remember for humans. In most places where it counts—e. g., when logging in, or in a list of files and their owners—Linux will use the textual name whenever possible.

UID
user name

The Linux kernel does not know anything about textual user names; process data and the ownership data in the filesystem use the UID exclusively. This may lead to difficulties if a user is deleted while he still owns files on the system, and the UID is reassigned to a different user. That user "inherits" the previous UID owner's files.

There is no technical problem with assigning the same (numerical) UID to different user names. These users have equal access to all files owned by that UID, but every user can have his own password. You should not actually use this (or if you do, use it only with great circumspection).

2.1.2 Users and Groups

To work with a Linux computer you need to log in first. This allows the system to recognise you and to assign you the correct access rights (of which more later). Everything you do during your session (from logging in to logging out) happens under your user account. In addition, every user has a **home directory**, where only they can store and manage their own files, and where other users often have no read permission and very emphatically no write permission. (Only the system administrator—root—may read and write all files.)

home directory

Depending on which Linux distribution you use (cue: Ubuntu) it may be possible that you do not have to log into the system explicitly. This is because the computer "knows" that it will usually be you and simply assumes that this is going to be the case. You are trading security for convenience; this particular deal probably makes sense only where you can stipulate with reasonable certainty that nobody except you will switch on your computer—and hence should be restricted *by rights* to the computer in your single-person household without a cleaner. We told you so.

Several users who want to share access to certain system resources or files can form a **group**. Linux identifies group members either fixedly by name or transiently by a login procedure similar to that for users. Groups have no "home directories" like users do, but as the administrator you can of course create arbitrary directories meant for certain groups and having appropriate access rights.

group

Groups, too, are identified internally using numerical identifiers ("group IDs" or GIDs).

Group names relate to GIDs as user names to UIDs: The Linux kernel only knows about the former and stores only the former in process data or the file system.

Every user belongs to a *primary group* and possibly several *secondary* or *additional groups*. In a corporate setting it would, for example, be possible to introduce project-specific groups and to assign the people collaborating on those projects to the appropriate group in order to allow them to manage common data in a directory only accessible to group members.

[1] Which of course is slightly more dangerous again, since programs runninig on the same screen can communicate with one another

For the purposes of access control, all groups carry equivalent weight—every user always enjoys all rights deriving from all the groups that he is a member of. The only difference between the primary and secondary groups is that files newly created by a user are usually[2] assigned to his primary group.

Up to (and including) version 2.4 of the Linux kernel, a user could be a member of at most 32 additional groups; since Linux 2.6 the number of secondary groups is unlimited.

You can find out a user account's UID, the primary and secondary groups and the corresponding GIDs by means of the `id` program:

```
$ id
uid=1000(joe) gid=1000(joe) groups=24(cdrom),29(audio),44(video),▷
 ◁ 1000(joe)
$ id root
uid=0(root) gid=0(root) groups=0(root)
```

With the options `-u`, `-g`, and `-G`, `id` lets itself be persuaded to output just the account's UID, the GID of the primary group, or the GIDs of the secondary groups. (These options cannot be combined.) With the additional option `-n` you get names instead of numbers:

```
$ id -G
1000 24 29 44
$ id -Gn
joe cdrom audio video
```

The `groups` command yields the same result as the ”‘`id -Gn`”’ command.

`last` You can use the `last` command to find who logged into your computer and when (and, in the case of logins via the network, from where):

```
$ last
joe      pts/1        pcjoe.example.c  Wed Feb 29 10:51   still logged in
bigboss  pts/0        pc01.example.c   Wed Feb 29 08:44   still logged in
joe      pts/2        pcjoe.example.c  Wed Feb 29 01:17 - 08:44  (07:27)
sue      pts/0        :0               Tue Feb 28 17:28 - 18:11  (00:43)
◁◁◁◁◁
reboot   system boot 3.2.0-1-amd64    Fri Feb  3 17:43 - 13:25 (4+19:42)
◁◁◁◁◁
```

For network-based sessions, the third column specifies the name of the `ssh` client computer. “`:0`” denotes the graphical screen (the first X server, to be exact—there might be more than one).

Do also note the `reboot` entry, which tells you that the computer was started. The third column contains the version number of the Linux operating system kernel as provided by “`uname -r`”.

With a user name, `last` provides information about a particular user:

```
$ last
joe      pts/1        pcjoe.example.c  Wed Feb 29 10:51   still logged in
joe      pts/2        pcjoe.example.c  Wed Feb 29 01:17 - 08:44  (07:27)
◁◁◁◁◁
```

[2] The exception occurs where the owner of a directory has decreed that new files and subdirectories within this directory are to be assigned to the same group as the directory itself. We mention this strictly for completeness.

You might be bothered (and rightfully so!) by the fact that this somewhat sensitive information is apparently made available on a casual basis to arbitrary system users. If you (as the administrator) want to protect your users' privacy somewhat better than you Linux distribution does by default, you can use the

```
# chmod o-r /var/log/wtmp
```

command to remove general read permissions from the file that `last` consults for the telltale data. Users without administrator privileges then get to see something like

```
$ last
last: /var/log/wtmp: Permission denied
```

2.1.3 People and Pseudo-Users

pseudo-users

Besides "natural" persons—the system's human users—the user and group concept is also used to allocate access rights to certain parts of the system. This means that, in addition to the personal accounts of the "real" users like you, there are further accounts that do not correspond to actual human users but are assigned to administrative functions internally. They define functional "roles" with their own accounts and groups.

After installing Linux, you will find several such pseudo-users and groups in the `/etc/passwd` and `/etc/group` files. The most important role is that of the `root` user (which you know) and its eponymous group. The UID and GID of `root` are 0 (zero).

`root`'s privileges are tied to UID 0; GID 0 does not confer any additional access privileges.

pseudo-users for privileges

Further pseudo-users belong to certain software systems (e. g., `news` for Usenet news using INN, or `postfix` for the Postfix mail server) or certain components or devices (such as printers, tape or floppy drives). You can access these accounts, if necessary, like other user accounts via the `su` command. These pseudo-users are helpful as file or directory owners, in order to fit the access rights tied to file ownership to special requirements without having to use the `root` account. The same appkies to groups; the members of the `disk` group, for example, have block-level access to the system's disks.

Exercises

2.1 [1] How does the operating system kernel differentiate between various users and groups?

2.2 [2] What happens if a UID is assigned to two different user names? Is that allowed?

2.3 [1] What is a pseudo-user? Give examples!

2.4 [2] (On the second reading.) Is it acceptable to assign a user to group `disk` who you would not want to trust with the `root` password? Why (not)?

2.2 User and Group Information

2.2.1 The `/etc/passwd` File

The `/etc/passwd` file is the system user database. There is an entry in this file for every user on the system—a line consisting of attributes like the Linux user name, "real" name, etc. After the system is first installed, the file contains entries for most pseudo-users.

The entries in `/etc/passwd` have the following format:

```
⟨user name⟩:⟨password⟩:⟨UID⟩:⟨GID⟩:⟨GECOS⟩:⟨home directory⟩:⟨shell⟩
```

⟨user name⟩ This name should consist of lowercase letters and digits; the first character should be a letter. Unix systems often consider only the first eight characters—Linux does not have this limitation but in heterogeneous networks you should take it into account.

⚠ Resist the temptation to use umlauts, punctuation and similar special characters in user names, even if the system lets you do so—not all tools that create new user accounts are picky, and you could of course edit `/etc/passwd` by hand. What seems to work splendidly at first glance may lead to problems elsewhere later.

You should also stay away from user names consisting of only uppercase letters or only digits. The former may give their owners trouble logging in (see exercise 2.6), the latter can lead to confusion, especially if the numerical user name does not equal the account's numerical UID. Commands such as "'`ls -l`"' will display the UID if there is no corresponding entry for it in `/etc/passwd`, and it is not exactly straightforward to tell UIDs from purely numerical user names in `ls` output.

⟨password⟩ Traditionally, this field contains the user's encrypted password. Today, most Linux distributions use "shadow passwords"; instead of storing the password in the publically readable `/etc/passwd` file, it is stored in `/etc/shadow` which can only be accessed by the administrator and some privileged programs. In `/etc/passwd`, a "`x`" calls attention to this circumstance. Every user can avail himself of the `passwd` program to change his password.

⟨UID⟩ The numerical user identifier—a number between 0 and $2^{32} - 1$. By convention, UIDs from 0 to 99 (inclusive) are reserved for the system, UIDs from 100 to 499 are for use by software packages if they need pseudo-user accounts. With most popular distributions, "real" users' UIDs start from 500 (or 1000).

Precisely because the system differentiates between users not by name but by UID, the kernel treats two accounts as completely identical if they contain different user names but the same UID—at least as far as the access privileges are concerned. Commands that display a user name (e. g., "'`ls -l`"' or `id`) show the one used when the user logged in.

primary group

⟨GID⟩ The GID of the user's **primary group** after logging in.

The Novell/SUSE distributions (among others) assign a single group such as `users` as the shared primary group of all users. This method is quite established as well as easy to understand.

Many distributions, such as those by Red Hat or Debian GNU/Linux, create a new group whenever a new account is created, with the GID equalling the account's UID. The idea behind this is to allow more sophisticated assignments of rights than with the approach that puts all users into the same group `users`. Consider the following situation: Jim (user name `jim`) is the personal assistant of CEO Sue (user name `sue`). In this capacity he sometimes needs to access files stored inside Sue's home directory that other users should not be able to get at. The method used by Red Hat, Debian & co., "one group per user", makes it straightforward to put user `jim` into group `sue` and to arrange for Sue's files to be readable for all group members (the default case) but not others. With the "one group for everyone" approach it would have been necessary to introduce a new group completely from scratch, and to reconfigure the `jim` and `sue` accounts accordingly.

By virtue of the assignment in `/etc/passwd`, every user must be a member of at least one group.

The user's secondary groups (if applicable) are determined from entries in the `/etc/group` file.

⟨*GECOS*⟩ This is the comment field, also known as the "GECOS field".

GECOS stands for "General Electric Comprehensive Operating System" and has nothing whatever to do with Linux, except that in the early days of Unix this field was added to `/etc/passwd` in order to keep compatibility data for a GECOS remote job entry service.

This field contains various bits of information about the user, in particular his "real" name and optional data such as the office number or telephone number. This information is used by programs such as `mail` or `finger`. The full name is often included in the sender's address by news and mail software.

Theoretically there is a program called `chfn` that lets you (as a user) change the content of your GECOS field. Whether that works in any particular case is a different question, since at least in a corporate setting one does not necessarily want to allow people to change their names at a whim.

⟨*home directory*⟩ This directory is that user's personal area for storing his own files. A newly created home directory is by no means empty, since a new user normally receives a number of "profile" files as his basic equipment. When a user logs in, his shell uses his home directory as its current directory, i. e., immediately after logging in the user is deposited there.

⟨*shell*⟩ The name of the program to be started by `login` after successful authentication—this is usually a shell. The seventh field extends through the end of the line.

The user can change this entry by means of the `chsh` program. The eligible programs (shells) are listed in the `/etc/shells` file. If a user is not supposed to have an interactive shell, an arbitrary program, with arguments, can be entered here (a common candidate is `/bin/true`). This field may also remain empty, in which case the standard shell `/bin/sh` will be started.

If you log in to a graphical environment, various programs will be started on your behalf, but not necessarily an interactive shell. The shell entry in `/etc/passwd` comes into its own, however, when you invoke a terminal emulator such as `xterm` or `konsole`, since these programs usually check it to identify your preferred shell.

Some of the fields shown here may be empty. Absolutely necessary are only the user name, UID, GID and home directory. For most user accounts, all the fields will be filled in, but pseudo-users might use only part of the fields.

home directories

The home directories are usually located below `/home` and take their name from their owner's user name. In general this is a fairly sensible convention which makes a given user's home directory easy to find. In theory, a home directory might be placed anywhere in the file system under a completely arbitrary name.

On large systems it is common to introduce one or more additional levels of directories between `/home` and the "user name" directory, such as

```
/home/hr/joe                                Joe from Human Resources
/home/devel/sue                                 Sue from Development
/home/exec/bob                                           Bob the CEO
```

There are several reasons for this. On the one hand this makes it easier to keep one department's home directory on a server within that department, while still making it available to other client computers. On the other hand, Unix (and some Linux) file systems used to be slow dealing with directories containing very many files, which would have had an unfortunate impact on a `/home` with several thousand entries. However, with current Linux file systems (`ext3` with `dir_index` and similar) this is no longer an issue.

tools Note that as an administrator you should not really be editing `/etc/passwd` by hand. There is a number of programs that will help you create and maintain user accounts.

In principle it is also possible to store the user database elsewhere than in `/etc/passwd`. On systems with very many users (thousands), storing user data in a relational database is preferable, while in heterogeneous networks a shared multi-platform user database, e. g., based on an LDAP directory, might recommend itself. The details of this, however, are beyond the scope of this course.

2.2.2 The `/etc/shadow` File

For security, nearly all current Linux distributions store encrypted user passwords in the `/etc/shadow` file ("shadow passwords"). This file is unreadable for normal users; only `root` may write to it, while members of the `shadow` group may read it in addition to `root`. If you try to display the file as a normal user an error occurs.

Use of `/etc/shadow` is not mandatory but highly recommended. However there may be system configurations where the additional security afforded by shadow passwords is nullified, for example if NIS is used to export user data to other hosts (especially in heterogeneous Unix environments).

format Again, this file contains one line for each user, with the following format:

```
⟨user name⟩:⟨password⟩:⟨change⟩:⟨min⟩:⟨max⟩▷
 ◁:⟨warn⟩:⟨grace⟩:⟨lock⟩:⟨reserved⟩
```

For example:

```
root:gaY2L19jxzHj5:10816:0:10000::::
daemon:*:8902:0:10000::::
joe:GodY6c5pZk1xs:10816:0:10000::::
```

Here is the meaning of the individual fields:

⟨user name⟩ This must correspond to an entry in the `/etc/passwd` file. This field "joins" the two files.

⟨password⟩ The user's encrypted password. An empty field generally means that the user can log in without a password. An asterisk or an exclamation point prevent the user in question from logging in. It is common to lock user's accounts without deleting them entirely by placing an asterisk or exclamation point at the beginning of the corresponding password.

⟨change⟩ The date of the last password change, in days since 1 January 1970.

⟨*min*⟩ The minimal number of days that must have passed since the last password change before the password may be changed again.

⟨*max*⟩ The maximal number of days that a password remains valid without having to be changed. After this time has elapsed the user must change his password.

⟨*warn*⟩ The number of days before the expiry of the ⟨*max*⟩ period that the user will be warned about having to change his password. Generally, the warning appears when logging in.

⟨*grace*⟩ The number of days, counting from the expiry of the ⟨*max*⟩ period, after which the account will be locked if the user does not change his password. (During the time from the expiry of the ⟨*max*⟩ period and the expiry of this grace period the user may log in but must immediately change his password.)

⟨*lock*⟩ The date on which the account will be definitively locked, again in days since 1 January 1970.

password encryption

Some brief remarks concerning password encryption are in order. You might think that if passwords are encrypted they can also be *decrypted* again. This would open all of the system's accounts to a clever cracker who manages to obtain a copy of `/etc/shadow`. However, in reality this is not the case, since password "encryption" is a one-way street. It is impossible to recover the decrypted representation of a Linux password from the "encrypted" form because the method used for encryption prevents this. The only way to "crack" the encryption is by encrypting likely passwords and checking whether they match what is in `/etc/shadow`.

Let's assume you select the characters of your password from the 95 visible ASCII characters (uppercase and lowercase letters are distinguished). This means that there are 95 different one-character passwords, $95^2 = 9025$ two-character passwords, and so on. With eight characters you are already up to 6.6 quadrillion ($6.6 \cdot 10^{15}$) possibilities. Stipulating that you can trial-encrypt 10 million passwords per second (not entirely unrealistic on current hardware), this means you would require approximately 21 years to work through all possible passwords. If you are in the fortunate position of owning a modern graphics card, another acceleration by a factor of 50–100 is quite feasible, which makes that about two months. And then of course there are handy services like Amazon's EC2, which will provide you (or random crackers) with almost arbitrary CPU power, or the friendly neighbourhood Russian bot net … so don't feel too safe.

There are a few other problems. The traditional method (usually called "crypt" or "DES"—the latter because it is based on, but not identical to, the eponymous encryption method[3]) should no longer be used if you can avoid it. It has the unpleasant property of only looking at the first eight characters of the entered password, and clever crackers can nowadays buy enough disk space to build a pre-encrypted cache of the 50 million (or so) most common passwords. To "crack" a password they only need to search their cache for the encrypted password, which can be done extremely quickly, and read off the corresponding clear-text password.

To make things even more laborious, when a newly entered password is encrypted the system traditionally adds a random element (the so-called

[3] If you must know exactly: The clear-text password is used as the key (!) to encrypt a constant string (typically a sequence of zero bytes). A DES key is 56 bits, which just happens to be 8 characters of 7 bits each (as the leftmost bit in each character is ignored). This process is repeated for a total of 25 rounds, with the previous round's output serving as the new input. Strictly speaking the encryption scheme used isn't quite DES but changed in a few places, to make it less feasible to construct a special password-cracking computer from commercially available DES encryption chips.

"salt") which selects one of 4096 different possibilities for the encrypted password. The main purpose of the salt is to avoid random hits resulting from user X, for some reason or other, getting a peek at the content of `/etc/shadow` and noting that his encrypted password looks just like that of user Y (hence letting him log into user Y's account using his own *cleartext* password). For a pleasant side effect, the disk space required for the cracker's pre-encrypted dictionary from the previous paragraph is blown up by a factor of 4096.

Nowadays, password encryption is commonly based on the MD5 algorithm, allows for passwords of arbitrary length and uses a 48-bit salt instead of the traditional 12 bits. Kindly enough, the encryption works much more slowly than "crypt", which is irrelevant for the usual purpose (checking a password upon login—you can still encrypt several hundred passwords per second) but does encumber clever crackers to a certain extent. (You should not let yourself be bothered by the fact that cryptographers poo-poo the MD5 scheme as such due to its insecurity. As far as password encryption is concerned, this is fairly meaningless.)

You should not expect too much of the various password administration parameters. They are being used by the text console login process, but whether other parts of the system (such as the graphical login screen) pay them any notice depends on your setup. Nor is there usually an advantage in forcing new passwords on users at short intervals—this usually results in a sequence like `bob1`, `bob2`, `bob3`, …, or users alternate between two passwords. A *minimal interval* that must pass before a user is allowed to change their password again is outright dangerous, since it may give a cracker a "window" for illicit access even though the user knows their password has been compromised.

The problem you need to cope with as a system administrator is usually not people trying to crack your system's passwords by "brute force". It is much more promising, as a rule, to use "social engineering". To guess your password, the clever cracker does not start at `a`, `b`, and so on, but with your spouse's first name, your kids' first names, your car's plate number, your dog's birthday et cetera. (We do not in any way mean to imply that *you* would use such a stupid password. No, no, not *you* by any means. However, we are not quite so positive about your boss …) And then there is of course the time-honoured phone call approach: "Hi, this is the IT department. We're doing a security systems test and urgently require your user name and password."

There are diverse ways of making Linux passwords more secure. Apart from the improved encryption scheme mentioned above, which by now is used by default by most Linux distributions, these include complaining about (too) weak passwords when they are first set up, or proactively running software that will try to identify weak encrypted passwords, just like clever crackers would (*Caution:* Do this in your workplace only with written (!) pre-approval from your boss!). Other methods avoid passwords completely in favour of constantly changing magic numbers (as in SecurID) or smart cards. All of this is beyond the scope of this manual, and therefore we refer you to the Linup Front manual *Linux Security*.

2.2.3 The `/etc/group` File

group database

By default, Linux keeps group information in the `/etc/group` file. This file contains one-line entry for each group in the system, which like the entries in `/etc/passwd` consists of fields separated by colons (`:`). More precisely, `/etc/group` contains four fields per line.

⟨*group name*⟩`:`⟨*password*⟩`:`⟨*GID*⟩`:`⟨*members*⟩

Their meaning is as follows:

⟨*group name*⟩ The name of the group, for use in directory listings, etc.

⟨*password*⟩ An optional password for this group. This lets users who are not members of the group via /etc/shadow or /etc/group assume membership of the group using newgrp. A "*" as an invalid character prevents normal users from changing to the group in question. A "x" refers to the separate password file /etc/gshadow.

⟨*GID*⟩ The group's numerical group identifier.

⟨*Members*⟩ A comma-separated list of user names. This list contains all users who have this group as a secondary group, i. e., who are members of this group but have a different value in the GID field of their /etc/passwd entry. (Users with this group as their primary group may also be listed here but that is unnecessary.)

A /etc/group file could, for example, look like this:

```
root:x:0:root
bin:x:1:root,daemon
users:x:100:
project1:x:101:joe,sue
project2:x:102:bob
```

administrative groups

The entries for the root and bin groups are entries for administrative groups, similar to the system's pseudo-user accounts. Many files are assigned to groups like this. The other groups contain user accounts.

GID values

Like UIDs, GIDs are counted from a specific value, typically 100. For a valid entry, at least the first and third field (group name and GID) must be filled in. Such an entry assigns a GID (which might occur in a user's primary GID field in /etc/passwd) a textual name.

membership list

group password

The password and/or membership fields must only be filled in for groups that are assigned to users as secondary groups. The users listed in the membership list are not asked for a password when they want to change GIDs using the newgrp command. If an encrypted password is given, users without an entry in the membership list can authenticate using the password to assume membership of the group.

> In practice, group passwords are hardly if ever used, as the administrative overhead barely justifies the benefits to be derived from them. On the one hand it is more convenient to assign the group directly to the users in question (since, from version 2.6 of the Linux kernel on, there is no limit to the number of secondary groups a user can join), and on the other hand a *single* password that must be known by *all* group members does not exactly make for bullet-proof security.

> If you want to be safe, ensure that there is an asterisk ("*") in every group password slot.

2.2.4 The /etc/gshadow File

As for the user database, there is a shadow password extension for the group database. The group passwords, which would otherwise be encrypted but readable for anyone in /etc/group (similar to /etc/passwd), are stored in the separate file /etc/gshadow. This also contains additional information about the group, for example the names of the group administrators who are entitled to add or remove members from the group.

2.2.5 The `getent` Command

Of course you can read and process the `/etc/passwd`, `/etc/shadow`, and `/etc/group` files, like all other text files, using programs such as `cat`, `less` or `grep` (OK, OK, you need to be `root` to get at `/etc/shadow`). There are, however, some practical problems:

- You may not be able to see the whole truth: Your user database (or parts of it) might be stored on an LDAP server, SQL database, or a Windows domain controller, and there really may not be much of interest in `/etc/passwd`.
- If you want to look for a specific user's entry, it is slightly inconvenient to type this using `grep` if you want to avoid "false positives".

The `getent` command makes it possible to query the various databases for user and group information directly. With

```
$ getent passwd
```

you will be shown something that looks like `/etc/passwd`, but has been assembled from all sources of user information that are currently configured on your computer. With

```
$ getent passwd hugo
```

you can obtain user `hugo`'s entry, no matter where it is actually stored. Instead of `passwd`, you may also specify `shadow`, `group`, or `gshadow` to consult the respective database. (Naturally, even with `getent` you can only access `shadow` and `gshadow` as user `root`.)

The term "database" is understood as "totality of all sources from where the C library can obtain information on that topic (such as users)". If you want to know exactly where that information comes from (or might come from), then read `nsswitch.conf`(5) and examine the `/etc/nsswitch.conf` file on your system.

You may also specify several user or group names. In that case, information on all the named users or groups will be output:

```
$ getent passwd hugo susie fritz
```

Exercises

2.5 [1] Which value will you find in the second column of the `/etc/passwd` file? Why do you find that value there?

2.6 [2] Switch to a text console (using, e. g., Alt+F1) and try logging in but enter your user name in uppercase letters. What happens?

2.7 [2] How can you check that there is an entry in the `shadow` database for every entry in the `passwd` database? (`pwconv` only considers the `/etc/passwd` and `/etc/shadow` *files*, and also rewrites the `/etc/shadow` file, which we don't want.

2.3 Managing User Accounts and Group Information

After a new Linux distribution has been installed, there is often just the `root` account for the system administrator and the pseudo-users' accounts. Any other user accounts must be created first (and most distributions today will gently but firmly nudge the installing person to create at least *one* "normal" user account).

As the administrator, it is your job to create and manage the accounts for all required users (real and pseudo). To facilitate this, Linux comes with several tools for user management. With them, this is mostly a straightforward task, but it is important that you understand the background.

tools for user management

2.3.1 Creating User Accounts

The procedure for creating a new user account is always the same (in principle) and consists of the following steps:

1. You must create entries in the `/etc/passwd` (and possibly `/etc/shadow`) files.
2. If necessary, an entry (or several) in the `/etc/group` file is necessary.
3. You must create the home directory, copy a basic set of files into it, and transfer ownership of the lot to the new user.
4. If necessary, you must enter the user in further databases, e. g., for disk quotas (section 7.4), database access privilege tables and special applications.

All files involved in adding a new account are plain text files. You can perform each step manually using a text editor. However, as this is a job that is as tedious as it is elaborate, it behooves you to let the system help you, by means of the `useradd` program.

`useradd`

In the simplest case, you pass `useradd` merely the new user's user name. Optionally, you can enter various other user parameters; for unspecified parameters (typically the UID), "reasonable" default values will be chosen automatically. On request, the user's home directory will be created and endowed with a basic set of files that the program takes from the `/etc/skel` directory. The `useradd` command's syntax is:

```
useradd [⟨options⟩] ⟨user name⟩
```

The following options (among others) are available:

`-c` ⟨*comment*⟩ GECOS field entry

`-d` ⟨*home directory*⟩ If this option is missing, `/home/`⟨*user name*⟩ is assumed

`-e` ⟨*date*⟩ On this date the account will be deactivated automatically (format "YYYY-MM-DD")

`-g` ⟨*group*⟩ The new user's primary group (name or GID). This group must exist.

`-G` ⟨*group*⟩[,⟨*group*⟩]… Supplementary groups (names or GIDs). These groups must also exist.

`-s` ⟨*shell*⟩ The new user's login shell

`-u` ⟨*UID*⟩ The new user's numerical UID. This UID must not be already in use, unless the "`-o`" option is given

`-m` Creates the home directory and copies the basic set of files to it. These files come from `/etc/skel`, unless a different directory was named using "`-k` ⟨*directory*⟩".

For instance, the

```
# useradd -c "Joe Smith" -m -d /home/joe -g devel \
>    -k /etc/skel.devel
```

command creates an account by the name of `joe` for a user called Joe Smith, and assigns it to the `devel` group. `joe`'s home directory is created as `/home/joe`, and the files from `/etc/skel.devel` are being copied into it.

With the `-D` option (on SUSE distributions, `--show-defaults`) you may set default values for some of the properties of new user accounts. Without additional options, the default values are displayed:

```
# useradd -D
GROUP=100
HOME=/home
INACTIVE=-1
EXPIRE=
SHELL=/bin/sh
SKEL=/etc/skel
CREATE_MAIL_SPOOL=no
```

You can change these values using the `-g`, `-b`, `-f`, `-e`, and `-s` options, respectively:

```
# useradd -D -s /usr/bin/zsh                    zsh as the default shell
```

The final two values in the list cannot be changed.

`useradd` is a fairly low-level tool. In real life, you as an experienced administrator will likely not be adding new user accounts by means of `useradd`, but through a shell script that incorporates your local policies (just so you don't have to remember them all the time). Unfortunately you will have to come up with this shell script by yourself—at least unless you are using Debian GNU/Linux or one of its derivatives (see below).

Watch out: Even though every serious Linux distribution comes with a program called `useradd`, the implementations differ in their details.

The Red Hat distributions include a fairly run-of-the-mill version of `useradd`, without bells and whistles, which provides the features discussed above.

The SUSE distributions' `useradd` is geared towards optionally adding users to a LDAP directory rather than the `/etc/passwd` file. (This is why the `-D` option cannot be used to query or set default values like it can elsewhere—it is already spoken for to do LDAPy things.) The details are beyond the scope of this manual.

On Debian GNU/Linux and Ubuntu, `useradd` does exist but the recommended method to create new user accounts is a program called `adduser` (thankfully this is not confusing). The advantage of `adduser` is that it plays according to Debian GNU Linux's rules, and furthermore makes it possible to execute arbitrary other actions for a new account besides creating the actual account. For example, one might create a directory in a web server's document tree so that the new user (and nobody else) can publish files there, or the user could automatically be authorised to access a database server. You can find the details in `adduser`(8) and `adduser.conf`(5).

password

After it has been created using `useradd`, the new account is not yet accessible; the system administrator must first set up a password. We shall be explaining this presently.

2.3.2 The `passwd` Command

The `passwd` command is used to set up passwords for users. If you are logged in as `root`, then

```
# passwd joe
```

asks for a new password for user `john` (You must enter it twice as it will not be echoed to the screen).

The `passwd` command is also available to normal users, to let them change their own passwords (changing other users' passwords is `root`'s prerogative):

```
$ passwd
Changing password for joe.
(current) UNIX password: secret123
Enter new UNIX password: 321terces
Retype new UNIX password: 321terces
passwd: password updated successfully
```

Normal users must enter their own password correctly once before being allowed to set a new one. This is supposed to make life difficult for practical jokers that play around on your computer if you had to step out very urgently and didn't have time to engage the screen lock.

On the side, `passwd` serves to manage various settings in `/etc/shadow`. For example, you can look at a user's "password state" by calling the `passwd` command with the `-S` option:

```
# passwd -S bob
bob LK 10/15/99 0 99999 7 0
```

The first field in the output is (once more) the user name, followed by the password state: "`PS`" or "`P`" if a password is set, "`LK`" or "`L`" for a locked account, and "`NP`" for an account with no password at all. The other fields are, respectively, the date of the last password change, the minimum and maximum interval for changing the password, the expiry warning interval and the "grace period" before the account is locked completely after the password has expired. (See also Section 2.2.2.)

You can change some of these settings by means of `passwd` options. Here are a few examples:

```
# passwd -l joe                                          Lock the account
# passwd -u joe                                        Unlock the account
# passwd -n 7 joe                      Password change at most every 7 days
# passwd -x 30 joe                   Password change at least every 30 days
# passwd -w 3 joe                   3 days grace period before password expires
```

Locking and unlocking accounts by means of `-l` and `-u` works by putting a "`!`" in front of the encrypted password in `/etc/shadow`. Since "`!`" cannot result from password encryption, it is impossible to enter something upon login that matches the "encrypted password" in the user database—hence access via the usual login procedure is prevented. Once the "`!`" is removed, the original password is back in force. (Astute, innit?) However, you should keep in mind that users may be able to gain access to the system by other means that do not refer to the encrypted password in the user database, such as the secure shell with public-key authentication.

Changing the remaining settings in `/etc/shadow` requires the `chage` command:

```
# chage -E 2009-12-01 joe                        Lock account from 1 Dec 2009
# chage -E -1 joe                                          Cancel expiry date
# chage -I 7 joe                         Grace period 1 week from password expiry
# chage -m 7 joe                                         Like passwd -n (Grr.)
# chage -M 7 joe                                    Like passwd -x (Grr, grr.)
# chage -W 3 joe                               Like passwd -w (Grr, grr, grr.)
```

(`chage` can change all settings that `passwd` can change, and then some.)

If you cannot remember the option names, invoke `chage` with the name of a user account only. The program will present you with a sequence of the current values to change or confirm.

You cannot retrieve a clear-text password even if you are the administrator. Even checking `/etc/shadow` doesn't help, since this file stores all passwords already encrypted. If a user forgets their password, it is usually sufficient to reset their password using the `passwd` command.

Should you have forgotten the `root` password and not be logged in as `root` by any chance, your last option is to boot Linux to a shell, or boot from a rescue disk or CD. (See Chapter 8.) After that, you can use an editor to clear the ⟨*password*⟩ field of the `root` entry in `/etc/passwd`.

Exercises

2.8 [3] Change user `joe`'s password. How does the `/etc/shadow` file change? Query that account's password state.

2.9 [!2] The user `dumbo` has forgotten his password. How can you help him?

2.10 [!3] Adjust the settings for user `joe`'s password such that he can change his password after at least a week, and must change it after at most two weeks. There should be a warning two days before the two weeks are up. Check the settings afterwards.

2.3.3 Deleting User Accounts

To delete a user account, you need to remove the user's entries from `/etc/passwd` and `/etc/shadow`, delete all references to that user in `/etc/group`, and remove the user's home directory as well as all other files created or owned by that user. If the user has, e. g., a mail box for incoming messages in `/var/mail`, that should also be removed.

`userdel` Again there is a suitable command to automate these steps. The `userdel` command removes a user account completely. Its syntax:

```
userdel [-r] ⟨user name⟩
```

The `-r` option ensures that the user's home directory (including its content) and his mail box in `/var/mail` will be removed; other files belonging to the user—e. g., `crontab` files—must be delete manually. A quick way to locate and remove files belonging to a certain user is the

```
find / -uid ⟨UID⟩ -delete
```

command. Without the `-r`option, only the user information is removed from the user database; the home directory remains in place.

2.3.4 Changing User Accounts and Group Assignment

User accounts and group assignments are traditionally changed by editing the `/etc/passwd` and `/etc/group` files. However, many systems contain commands like `usermod` and `groupmod` for the same purpose, and you should prefer these since they are safer and—mostly—more convenient to use.

`usermod` The `usermod` program accepts mostly the same options as `useradd`, but changes existing user accounts instead of creating new ones. For example, with

```
usermod -g ⟨group⟩ ⟨user name⟩
```

you could change a user's primary group.

Changing UIDs Caution! If you want to change an existing user account's UID, you could edit the ⟨*UID*⟩ field in `/etc/passwd` directly. However, you should at the same time transfer that user's files to the new UID using `chown`: "`chown -R tux /home/tux`" re-confers ownership of all files below user tux's home directory to user tux, after you have changed the UID for that account. If "ls -l" displays a numerical UID instead of a textual name, this implies that there is no user name for the UID of these files. You can fix this using chown.

2.3.5 Changing User Information Directly—vipw

The vipw command invokes an editor (vi or a different one) to edit /etc/passwd directly. At the same time, the file in question is locked in order to keep other users from simultaneously changing the file using, e. g., passwd (which changes would be lost). With the -s option, /etc/shadow can be edited.

The actual editor that is invoked is determined by the value of the VISUAL environment variable, alternatively that of the EDITOR environment variable; if neither exists, vi will be launched.

Exercises

2.11 [!2] Create a user called test. Change to the test account and create a few files using touch, including a few in a different directory than the home directory (say, /tmp). Change back to root and change test's UID. What do you see when listing user test's files?

2.12 [!2] Create a user called test1 using your distribution's graphical tool (if available), test2 by means of the useradd command, and another, test3, manually. Look at the configuration files. Can you work without problems using any of these three accounts? Create a file using each of the new accounts.

2.13 [!2] Delete user test2's account and ensure that there are no files left on the system that belong to that user.

2.14 [2] Change user test1's UID. What else do you need to do?

2.15 [2] Change user test1's home directory from /home/test1 to /home/user/test1.

2.3.6 Creating, Changing and Deleting Groups

Like user accounts, you can create groups using any of several methods. The "manual" method is much less tedious here than when creating new user accounts: Since groups do not have home directories, it is usually sufficient to edit the /etc/group file using any text editor, and to add a suitable new line (see below for vigr). When group passwords are used, another entry must be added to /etc/gshadow.

Incidentally, there is nothing wrong with creating directories for groups. Group members can place the fruits of their collective labour there. The approach is similar to creating user home directories, although no basic set of configuration files needs to be copied.

For group management, there are, by analogy to useradd, usermod, and userdel, the groupadd, groupmod, and groupdel programs that you should use in favour of editing /etc/group and /etc/gshadow directly. With groupadd you can create new groups simply by giving the correct command parameters: groupadd

```
groupadd [-g ⟨GID⟩] ⟨group name⟩
```

The -g option allows you to specify a given group number. As mentioned before, this is a positive integer. The values up to 99 are usually reserved for system groups. If -g is not specified, the next free GID is used.

You can edit existing groups with groupmod without having to write to /etc/group directly: groupmod

```
groupmod [-g ⟨GID⟩] [-n ⟨name⟩] ⟨group name⟩
```

The "`-g` ⟨*GID*⟩" option changes the group's GID. Unresolved file group assignments must be adjusted manually. The "`-n` ⟨*name*⟩" option sets a new name for the group without changing the GID; manual adjustments are not necessary.

`groupdel`

There is also a tool to remove group entries. This is unsurprisingly called `groupdel`:

```
groupdel ⟨group name⟩
```

Here, too, it makes sense to check the file system and adjust "orphaned" group assignments for files with the `chgrp` command. Users' primary groups may not be removed—the users in question must either be removed beforehand, or they must be reassigned to a different primary group.

`gpasswd`

group administrator

The `gpasswd` command is mainly used to manipulate group passwords in a way similar to the `passwd` command. The system administrator can, however, delegate the administration of a group's membership list to one or more group administrators. Group administrators also use the `gpasswd` command:

```
gpasswd -a ⟨user⟩ ⟨group⟩
```

adds the ⟨*user*⟩ to the ⟨*group*⟩, and

```
gpasswd -d ⟨user⟩ ⟨group⟩
```

removes him again. With

```
gpasswd -A ⟨user⟩,… ⟨group⟩
```

the system administrator can nominate users who are to serve as group administrators.

SUSE The SUSE distributions haven't included `gpasswd` for some time. Instead there are modified versions of the user and group administration tools that can handle an LDAP directory.

`vigr`

As the system administrator, you can change the group database directly using the `vigr` command. It works like `vipw`, by invoking an editor for "exclusive" access to `/etc/group`. Similarly, "`vigr -s`" gives you access to `/etc/gshadow`.

Exercises

2.16 [2] What are groups needed for? Give possible examples.

2.17 [1] Can you create a directory that all members of a group can access?

2.18 [!2] Create a supplementary group `test`. Only user `test1` should be a member of that group. Set a group password. Log in as user `test1` or `test2` and try to change over to the new group.

Commands in this Chapter

adduser	Convenient command to create new user accounts (Debian)	adduser(8)	34
chfn	Allows users to change the GECOS field in the user database	chfn(1)	27
getent	Gets entries from administrative databases	getent(1)	32
gpasswd	Allows a group administrator to change a group's membership and update the group password	gpasswd(1)	38
groupadd	Adds user groups to the system group database	groupadd(8)	37
groupdel	Deletes groups from the system group database	groupdel(8)	38
groupmod	Changes group entries in the system group database	groupmod(8)	37
groups	Displays the groups that a user is a member of	groups(1)	24
id	Displays a user's UID and GIDs	id(1)	24
last	List recently-logged-in users	last(1)	24
useradd	Adds new user accounts	useradd(8)	33
userdel	Removes user accounts	userdel(8)	36
usermod	Modifies the user database	usermod(8)	36
vigr	Allows editing /etc/group or /etc/gshadow with "file locking", to avoid conflicts	vipw(8)	38

Summary

- Access to the system is governed by user accounts.
- A user account has a numerical UID and (at least) one textual user name.
- Users can form groups. Groups have names and numerical GIDs.
- "Pseudo-users" and "pseudo-groups" serve to further refine access rights.
- The central user database is (normally) stored in the /etc/passwd file.
- The users' encrypted passwords are stored—together with other password parameters—in the /etc/shadow file, which is unreadable for normal users.
- Group information is stored in the /etc/group and /etc/gshadow files.
- Passwords are managed using the passwd program.
- The chage program is used to manage password parameters in /etc/shadow.
- User information is changed using vipw or—better—using the specialised tools useradd, usermod, and userdel.
- Group information can be manipulated using the groupadd, groupmod, groupdel and gpasswd programs.

3

Access Control

Contents

Goals

- Understanding the Linux access control/privilege mechanisms
- Being able to assign access permissions to files and directories
- Knowing about the “umask”, SUID, SGID and the “sticky bit”
- Knowing about file attributes in the `ext` file systems

Prerequisites

- Knowledge of Linux user and group concepts (see Chapter 2)
- Knowledge of Linux files and directories

3.1 The Linux Access Control System

access control system

Whenever several users have access to the same computer system there must be an access control system for processes, files and directories in order to ensure that user *A* cannot access user *B*'s private files just like that. To this end, Linux implements the standard system of Unix privileges.

separate privileges

access mode

In the Unix tradition, every file and directory is assigned to exactly one user (its owner) and one group. Every file supports separate privileges for its owner, the members of the group it is assigned to ("the group", for short), and all other users ("others"). Read, write and execute privileges can be enabled individually for these three sets of users. The owner may determine a file's access privileges. The group and others may only access a file if the owner confers suitable privileges to them. The sum total of a file's access permissions is also called its **access mode**.

access control

In a multi-user system which stores private or group-internal data on a generally accessible medium, the owner of a file can keep others from reading or modifying his files by instituting suitable access control. The rights to a file can be determined separately and independently for its owner, its group and the others. Access permissions allow users to map the responsibilities of a group collaborative process to the files that the group is working with.

3.2 Access Control For Files And Directories

3.2.1 The Basics

For each file and each directory in the system, Linux allows separate access rights for each of the three classes of users—owner, members of the file's group, others. These rights include read permission, write permission, and execute permission.

file permissions

As far as files are concerned, these permissions control approximately what their names suggest: Whoever has read permission may look at the file's content, whoever has write permission is allowed to change its content. Execute permission is necessary to launch the file as a process.

💡 Executing a binary "machine-language program" requires only execute permission. For files containing shell scripts or other types of "interpreted" programs, you also need read permission.

directory permissions

For directories, things look somewhat different: Read permission is required to look at a directory's content—for example, by executing the `ls` command. You need write permission to create, delete, or rename files in the directory. "Execute" permission stands for the possibility to "use" the directory in the sense that you can change into it using `cd`, or use its name in path names referring to files farther down in the directory tree.

💡 In directories where you have only read permission, you may read the file names but cannot find out anything else about the files. If you have only "execute permission" for a directory, you can access files as long as you know their names.

Usually it makes little sense to assign write and execute permission to a directory separately; however, it may be useful in certain special cases.

⚠ It is important to emphasise that write permission on a *file* is completely immaterial if the file is to be *deleted*—you need write permission to the *directory* that the file is in and nothing else! Since "deleting" a file only removes a reference to the actual file information (the inode) from the directory, this is purely a directory operation. The `rm` command does warn you if you're trying to delete a file that you do not have write permission for, but if you confirm the operation and have write permission to the directory involved, nothing will stand in the way of the operation's success. (Like any other

Unix-like system, Linux has no way of "deleting" a file outright; you can only remove all references to a file, in which case the Linux kernel decides on its own that no one will be able to access the file any longer, and gets rid of its content.)

If you do have write permission to the file but not its directory, you cannot remove the file completely. You can, however, truncate it down to 0 bytes and thereby remove its *content*, even though the file itself still exists in principle.

For each user, Linux determines the "most appropriate" access rights. For example, if the members of a file's group do not have read permission for the file but "others" do, then the group members may not read the file. The (admittedly enticing) rationale that, if all others may look at the file, then the group members, who are in some sense also part of "all others", should be allowed to read it as well, does not apply.

3.2.2 Inspecting and Changing Access Permissions

information

You can obtain information about the rights, user and group assignment that apply to a file using "`ls -l`":

```
$ ls -l
-rw-r--r--  1  joe  users   4711  Oct 4 11:11 datei.txt
drwxr-x---  2  joe  group2  4096  Oct 4 11:12 testdir
```

The string of characters in the first column of the table details the access permissions for the owner, the file's group, and others (the very first character is just the file type and has nothing to do with permissions). The third column gives the owner's user name, and the fourth that of the file's group.

In the permissions string, "`r`", "`w`", and "`x`" signify existing read, write, and execute permission, respectively. If there is just a "`-`" in the list, then the corresponding category does not enjoy the corresponding privilege. Thus, "`rw-r--r--`" stands for "read and write permission for the owner, but read permission only for group members and others".

chmod command

As the file owner, you may set access permissions for a file using the `chmod` command (from "change mode"). You can specify the three categories by means of the abbreviations "`u`" (user) for the owner (yourself), "`g`" (group) for the file's group's members, and "`o`" (others) for everyone else. The permissions themselves are given by the already-mentioned abbreviations "`r`", "`w`", and "`x`". Using "`+`", "`-`", and "`=`", you can specify whether the permissions in question should be added to any existing permissions, "subtracted" from the existing permissions, or used to replace whatever was set before. For example:

```
$ chmod u+x file                                   Execute permission for owner
$ chmod go+w file                  sets write permission for group and others
$ chmod g+rw file                     sets read and write permission for group
$ chmod g=rw,o=r file                          sets read and write permission,
                                             removes group execute permission;
                                            sets just read permission for others
$ chmod a+w file                                           equivalent to ugo+w
```

In fact, permission specifications can be considerably more complex. Consult the `info` documentation for `chmod` to find out all the details.

A file's owner is the single user (apart from `root`) who is allowed to change a file's or directory's access permissions. This privilege is independent of the actual permissions; the owner may take away all their own permissions, but that does not keep them from giving them back later.

The general syntax of the `chmod` command is

```
chmod [⟨options⟩] ⟨permissions⟩ ⟨name⟩ …
```

You can give as many file or directory names as desired. The most important options include:

-R If a directory name is given, the permissions of files and directories inside this directory will also be changed (and so on all the way down the tree).

--reference=⟨*name*⟩ Uses the access permissions of file ⟨*name*⟩. In this case no ⟨*permissions*⟩ must be given with the command.

You may also specify a file's access mode "numerically" instead of "symbolically" (what we just discussed). In practice this is very common for setting all permissions of a file or directory at once, and works like this: The three permission triples are represented as a three-digit octal number—the first digit describes the owner's rights, the second those of the file's group, and the third those that apply to "others". Each of these digits derives from the sum of the individual permissions, where read permission has value 4, write permission 2, and execute permission 1. Here are a few examples for common access modes in "`ls -l`" and octal form:

`rw-r--r--`	644
`r--------`	400
`rwxr-xr-x`	755

Using numerical access modes, you can only set all permissions at once—there is no way of setting or removing individual rights while leaving the others alone, like you can do with the "`+`" and "`-`" operators of the symbolic representation. Hence, the command

```
$ chmod 644 file
```

is equivalent to the symbolic

```
$ chmod u=rw,go=r file
```

3.2.3 Specifying File Owners and Groups—`chown` and `chgrp`

The `chown` command lets you set the owner and group of a file or directory. This command takes the desired owner's user name and/or group name and the file or directory name the change should apply to. It is called like

```
chown ⟨user name⟩[:][⟨group name⟩] ⟨name⟩ …
chown :⟨group name⟩ ⟨name⟩ …
```

If both a user and group name are given, both are changed; if just a user name is given, the group remains as it was; if a user name followed by a colon is given, then the file is assigned to the user's primary group. If just a group name is given (with the colon in front), the owner remains unchanged. For example:

```
# chown joe:devel letter.txt
# chown www-data foo.html                              new user www-data
# chown :devel /home/devel                            new group devel
```

`chown` also supports an obsolete syntax where a dot is used in place of the colon.

To "give away" files to other users or arbitrary groups you need to be `root`. The main reason for this is that normal users could otherwise annoy one another if the system uses quotas (i.e., every user can only use a certain amount of storage space).

Using the `chgrp` command, you can change a file's group even as a normal user—as long as you own the file and are a member of the *new* group:

```
chgrp ⟨group name⟩ ⟨name⟩ …
```

Changing a file's owner or group does not change the access permissions for the various categories.

`chown` and `chgrp` also support the `-R` option to apply changes recursively to part of the directory hierarchy.

Of course you can also change a file's permissions, group, and owner using most of the popular file browsers (such as Konqueror or Nautilus).

Exercises

3.1 [!2] Create a new file. What is that file's group? Use `chgrp` to assign the file to one of your secondary groups. What happens if you try to assign the file to a group that you are not a member of?

3.2 [4] Compare the mechanisms that various file browsers (like Konqueror, Nautilus, …) offer for setting a file's permissions, owner, group, … Are there notable differences?

3.2.4 The umask

New files are usually created using the (octal) access mode 666 (read and write permission for everyone). New directories are assigned the access mode 777. Since this is not always what is desired, Linux offers a mechanism to remove certain rights from these access modes. This is called "umask".

Nobody knows exactly where this name comes from—even though there are a few theories that all sound fairly implausible.

umask interpretation

The umask is an octal number whose complement is ANDed bitwise to the standard access mode—666 or 777—to arrive at the new file's or directory's actual access mode. In other words: You can consider the umask an access mode containing exactly those rights that the new file should *not* have. Here's an example—let the umask be 027:

1.	Umask value:	027	`----w-rwx`
2.	Complement of umask value:	750	`rwxr-x---`
3.	A new file's access mode:	666	`rw-rw-rw-`
4.	Result (2 and 3 ANDed together):	640	`rw-r-----`

The third column shows the octal value, the fourth a symbolic representation. The AND operation in step 4 can also be read off the fourth column of the second and third lines: In the fourth line ther e is a letter in each position that had a letter in the second *and* the third line—if there is just one dash ("`-`"), the result will be a dash.

If you'd rather not bother with the complement and AND, you can simply imagine that each digit of the umask is subtracted from the corresponding digit of the actual access mode and negative results are considered as zero (so no "borrowing" from the place to the left). For our example—access mode 666 and umask 027—this means something like

$$666 \ominus 027 = 640,$$

since $6 \ominus 0 = 6$, $6 \ominus 4 = 2$, and $6 \ominus 7 = 0$.

umask shell command

The umask is set using the umask shell command, either by invoking it directly or via a shell startup file—typically `~/.profile`, `~/.bash_profile`, or `~/.bashrc`.

process attribute

The umask is a process attribute similar to the current directory or the process environment, i. e., it is passed to child processes, but changes in a child process do not modify the parent process's settings.

syntax

The `umask` command takes a parameter specifying the desired umask:

```
umask [-S|⟨umask⟩]
```

symbolic representation

The umask may be given as an octal number or in a symbolic representation similar to that used by `chmod`—deviously enough, the symbolic form contains the permissions that should be *left* (rather than those to be taken away):

```
$ umask 027                                   … is equivalent to …
$ umask u=rwx,g=rx,o=
```

This means that in the symbolic form you must give the exact complement of the value that you would specify in the octal form—exactly those rights that do *not* occur in the octal specification.

If you specify no value at all, the current umask is displayed. If the `-S` option is given, the current umask is displayed in symbolic form (where, again, the remaining permissions are set):

```
$ umask
0027
$ umask -S
u=rwx,g=rx,o=
```

execute permission?

Note that you can only *remove* permissions using the umask. There is no way of making files executable by default.

umask and `chmod`

Incidentally, the umask also influences the `chmod` command. If you invoke `chmod` with a "`+`" mode (e. g., "`chmod +w file`") without referring to the owner, group or others, this is treated like "`a+`", but the permissions set in the umask are not modified. Consider the following example:

```
$ umask 027
$ touch file
$ chmod +x file
$ ls -l file
-rwxr-x---   1 tux   users        0 May 25 14:30 file
```

The "`chmod +x`" sets execute permission for the user and group, but not the others, since the umask contains the execute bit for "others". Thus with the umask you can take precautions against giving overly excessive permissions to files.

Theoretically, this also works for the `chmod` operators "`-`" and "`=`", but this does not make a lot of sense in practice.

Exercises

3.3 [!1] State a numerical umask that leaves the user all permissions, but removes all permissions from group members and others? What is the corresponding symbolic umask?

3.4 [2] Convince yourself that the "`chmod +x`" and "`chmod a+x`" commands indeed differ from each other as advertised.

3.3 Access Control Lists (ACLs)

As mentioned above, Linux allows you to assign permissions for a file's owner, group, and all others separately. For some applications, though, this three-tier system is too simple-minded, or the more sophisticated permission schemes of other operating systems must be mapped to Linux. Access control lists (ACLs) can be used for this.

On most file systems, Linux supports "POSIX ACLs" according to IEEE 1003.1e (draft 17) with some Linux-specific extensions. This lets you specify additional groups and users for files and directories, who then can be assigned read, write, and execute permissions that differ from those of the file's group and "others". Other rights, such as that to assign permissions, are still restricted to a file's owner (or `root`) and cannot be delegated even wiht ACLs. The `setfacl` and `getfacl` commands are used to set and query ACLs.

ACLs are a fairly new and rarely-used addition to Linux, and their use is subject to certain restrictions. The kernel does oversee compliance with them, but, for instance, not every program is able to copy ACLs along with a file's content—you may have to use a specially-adapted `tar` (`star`) for backups of a file system using ACLs. ACLs are supported by Samba, so Windows clients get to see the correct permissions, but if you export file systems to other (proprietary) Unix systems, it may be possible that your ACLs are ignored by Unix clients that do not support ACLs.

You can read up on ACLs on Linux on `http://acl.bestbits.at/` and in `acl`(5) as well as `getfacl`(1) and `setfacl`(1).

Detailed knowledge of ACLs is not required for the LPIC-1 exams.

3.4 Process Ownership

Linux considers not only the data on a storage medium as objects that can be owned. The processes on the system have owners, too.

Many commands create a process in the system's memory. During normal use, there are always several processes that the system protects from each other. Every process together with all data within its virtual address space is assigned to a user, its owner. This is most often the user who started the process—but processes created using administrator privileges may change their ownership, and the SUID mechanism (section 3.5) can also have a hand in this.

Processes have owners

The owners of processes are displayed by the `ps` program if it is invoked using the `-u` option.

```
# ps -u
USER   PID %CPU  %MEM SIZE      RSS TTY STAT  START  TIME COMMAND
bin     89  0.0   1.0  788      328  ?    S  13:27  0:00 rpc.portmap
test1  190  0.0   2.0 1100       28  3    S  13:27  0:00 bash
test1  613  0.0   1.3  968       24  3    S  15:05  0:00 vi XF86.tex
nobody 167  0.0   1.4  932       44  ?    S  13:27  0:00 httpd
root     1  0.0   1.0  776       16  ?    S  13:27  0:03 init [3]
root     2  0.0   0.0    0        0  ?   SW  13:27  0:00 (kflushd)
```

3.5 Special Permissions for Executable Files

When listing files using the "`ls -l`" command, you may sometimes encounter permission sets that differ from the usual `rwx`, such as

```
-rwsr-xr-x   1 root shadow  32916 Dec 11 20:47 /usr/bin/passwd
```

What does that mean? We have to digress here for a bit:

Assume that the passwd program carries the usual access mode:

```
-rwxr-xr-x   1 root shadow  32916 Dec 11 20:47 /usr/bin/passwd
```

A normal (unprivileged) user, say joe, wants to change his password and invokes the passwd program. Next, he receives the message "permission denied". What is the reason? The passwd process (which uses joe's privileges) tries to open the /etc/shadow file for writing and fails, since only root may write to that file—this cannot be different since otherwise, everybody would be able to manipulate passwords arbitrarily and, for example, change the root password.

SUID bit

By means of the **set-UID bit** (frequently called "SUID bit", for short) a program can be caused to run not with the invoker's privileges but those of the file owner—here, root. In the case of passwd, the *process* executing passwd has write permission to /etc/shadow, even though the invoking user, not being a system administrator, generally doesn't. It is the responsibility of the author of the passwd program to ensure that no monkey business goes on, e. g., by exploiting programming errors to change arbitrary files except /etc/shadow, or entries in /etc/shadow except the password field of the invoking user. On Linux, by the way, the set-UID mechanism works only for binary programs, not shell or other interpreter scripts.

Bell Labs used to hold a patent on the SUID mechanism, which was invented by Dennis Ritchie [SUID]. Originally, AT&T distributed Unix with the caveat that license fees would be levied after the patent had been granted; however, due to the logistical difficulties of charging hundreds of Unix installations small amounts of money retroactively, the patent was released into the public domain.

SGID bit

By analogy to the set-UID bit there is a SGID bit, which causes a process to be executed with the program file's group and the corresponding privileges (usually to access other files assigned to that group) rather than the invoker's group setting.

chmod syntax

The SUID and SGID modes, like all other access modes, can be changed using the chmod program, by giving symbolic permissions such as u+s (sets the SUID bit) or g-s (deletes the SGID bit). You can also set these bits in octal access modes by adding a fourth digit at the very left: The SUID bit has the value 4, the SGID bit the value 2—thus you can assign the access mode 4755 to a file to make it readable and executable to all users (the owner may also write to it) and to set the SUID bit.

ls output

You can recognise set-UID and set-GID programs in the output of "ls -l" by the symbolic abbreviations "s" in place of "x" for executable files.

3.6 Special Permissions for Directories

SGID for directories

There is another exception from the principle of assigning file ownership according to the identity of its creator: a directory's owner can decree that files created in that directory should belong to the same group as the directory itself. This can be specified by setting the SGID bit on the directory. (As directories cannot be executed, the SGID bit is available to be used for such things.)

A directory's access permissions are not changed via the SGID bit. To create a file in such a directory, a user must have write permission in the category (owner, group, others) that applies to him. If, for example, a user is neither the owner of a directory nor a member of the directory's group, the directory must be writable for "others" for him to be able to create files there. A file created in a SGID directory then belongs to that directory's group, even if the user is not a member of that group at all.

The typical application for the SGID bit on a directory is a directory that is used as file storage for a "project group". (Only) the members of the project group are supposed to be able to read and write all files in the directory, and to create new files. This means that you need to put all users collaborating on the project into a project group (a secondary group will suffice):

```
# groupadd project                                 Create new group
# usermod -a -G project joe                        joe into the group
# usermod -a -G project sue                                   sue too
◁◁◁◁◁
```

Now you can create the directory and assign it to the new group. The owner and group are given all permissions, the others none; you also set the SGID bit:

```
# cd /home/project
# chgrp project /home/project
# chmod u=rwx,g=srwx /home/project
```

Now, if user `hugo` creates a file in `/home/project`, that file should be assigned to group `project`:

```
$ id
uid=1000(joe) gid=1000(joe) groups=101(project),1000(joe)
$ touch /tmp/joe.txt                              Test: standard directory
$ ls -l /tmp/joe.txt
-rw-r--r-- 1 joe joe 0 Jan  6 17:23 /tmp/joe.txt
$ touch /home/project/joe.txt                            project directory
$ ls -l /home/project/joe.txt
-rw-r--r-- 1 joe project 0 Jan  6 17:24 /home/project/joe.txt
```

There is just a little fly in the ointment, which you will be able to discern by looking closely at the final line in the example: The file does belong to the correct group, but other members of group `project` may nevertheless only read it. If you want all members of group `project` to be able to write to it as well, you must either apply `chmod` after the fact (a nuisance) or else set the umask such that group write permission is retained (see Exercise 3.6).

The SGID mode only changes the system's behaviour when new files are created. Existing files work just the same as everywhere else. This means, for instance, that a file created outside the SGID directory keeps its existing group assignment when moved into it (whereas on copying, the new copy would be put into the directory's group).

The `chgrp` program works as always in SGID directories, too: the owner of a file can assign it to any group he is a member of. Is the owner not a member of the directory's group, he cannot put the file into that group using `chgrp`—he must create it afresh within the directory.

It is possible to set the SUID bit on a directory—this permission does not signify anything, though.

Linux supports another special mode for directories, where only a file's owner may delete or remove files within that directory:

```
drwxrwxrwt   7 root   root   1024 Apr  7 10:07 /tmp
```

This `t` mode, the "sticky bit", can be used to counter a problem which arises when public directories are in shared use: Write permission to a directory lets a user delete other users' files, regardless of their access mode and owner! For example, the `/tmp` directories are common ground, and many programs create their temporary files there. To do so, all users have write permission to that directory. This implies that any user has permission to delete files there.

Table 3.1: The most important file attributes

Attribute	Meaning
`A`	*atime* is not updated (interesting for mobile computers)
`a`	(*append-only*) The file can only be appended to
`c`	The file's content is compressed transparently (not implemented)
`d`	The file will not be backed up by `dump`
`i`	(*immutable*) The file cannot be changed at all
`j`	Write operations to the file's content are passed through the journal (`ext3` only)
`s`	File data will be overwritten with zeroes on deletion (not implemented)
`S`	Write operations to the file are performed "synchronously", i. e., without buffering them internally
`u`	The file may be "undeleted" after deletion (not implemented)

Usually, when deleting or renaming a file, the system does not consider that file's access permissions. If the "sticky bit" is set on a directory, a file in that directory can subsequently be deleted only by its owner, the directory's owner, or `root`. The "sticky bit" can be set or removed by specifying the symbolic `+t` and `-t` modes; in the octal representation it has value 1 in the same digit as SUID and SGID.

The "sticky bit" derives its name from an additional meaning it used to have in earlier Unix systems: At that time, programs were copied to swap space in their entirety when started, and removed completely after having terminated. Program files with the sticky bit set would be left in swap space instead of being removed. This would accelerate subsequent invocations of those programs since no copy would have to be done. Like most current Unix systems, Linux uses demand paging, i. e., it fetches only those parts of the code from the program's executable file that are really required, and does not copy anything to swap space at all; on Linux, the sticky bit never had its original meaning.

Exercises

3.5 [2] What does the special "`s`" privilege mean? Where do you find it? Can you set this privilege on a file that you created yourself?

3.6 [!1] Which `umask` invocation can be used to set up a umask that would, in the project directory example above, allow all members of the `project` group to read *and* write files in the project directory?

3.7 [2] What does the special "`t`" privilege mean? Where do you find it?

3.8 [4] (For programmers.) Write a C program that invokes a suitable command (such as `id`). Set this program SUID `root` (or SGID `root`) and observe what happens when you execute it.

3.9 [I]f you leave them alone for a few minutes with a `root` shell, clever users might try to stash a SUID `root` shell somewhere in the system, in order to assume administrator privileges when desired. Does that work with `bash`? With other shells?

3.7 File Attributes

file attributes

Besides the access permissions, the `ext2` and `ext3` file systems support further **file**

attributes enabling access to special file system features. The most important file attributes are summarised in Table 3.1.

a and i attributes

Most interesting are perhaps the "append-only" and "immutable" attributes, which you can use to protect log files and configuration files from modification; only root may set or reset these attributes, and once set they also apply to processes running as root.

In principle, an attacker who has gained root privileges may reset these attributes. However, attackers do not necessarily consider that they might be set.

A attribute

The A attribute may also be useful; you can use it on mobile computers to ensure that the disk isn't always running, in order to save power. Usually, whenever a file is read, its "atime"—the time of last access—is updated, which of course entails an inode write operation. Certain files are very frequently looked at in the background, such that the disk never gets to rest, and you can help here by judiciously applying the A attribute.

The c, s and u attributes sound very nice in theory, but are not (yet) supported by "normal" kernels. There are some more or less experimental enhancements making use of these attributes, and in part they are still pipe dreams.

chattr

You can set or reset attributes using the chattr command. This works rather like chmod: A preceding "+" sets one or more attributes, "-" deletes one or more attributes, and "=" causes the named attributes to be the only enabled ones. The -R option, as in chmod, lets chattr operate on all files in any subdirectories passed as arguments and their nested subdirectories. Symbolic links are ignored in the process.

```
# chattr +a /var/log/messages                              Append only
# chattr -R +j /data/important                        Data journaling …
# chattr -j /data/important/notso                       … with exception
```

lsattr

With the lsattr command, you can review the attributes set on a file. The program behaves similar to "ls -l":

```
# lsattr /var/log/messages
-----a----------- /var/log/messages
```

Every dash stands for a possible attribute. lsattr supports various options such as -R, -a, and -d, which generally behave like the eponymous options to ls.

Exercises

3.10 [!2] Convince yourself that the a and i attributes work as advertised.

3.11 [2] Can you make *all* dashes disappear in the lsattr output for a given file?

Commands in this Chapter

chattr	Sets file attributes for `ext2` and `ext3` file systems	`chattr`(1)	51
chgrp	Sets the assigned group of a file or directory	`chgrp`(1)	44
chmod	Sets access modes for files and directories	`chmod`(1)	43
chown	Sets the owner and/or assigned group of a file or directory	`chown`(1)	44
getfacl	Displays ACL data	`getfacl`(1)	47
lsattr	Displays file attributes on `ext2` and `ext3` file systems	`lsattr`(1)	51
setfacl	Enables ACL manipulation	`setfacl`(1)	47
star	POSIX-compatible tape archive with ACL support	`star`(1)	47

Summary

- Linux supports file read, write and execute permissions, where these permissions can be set separately for a file's owner, the members of the file's group and "all others".
- The sum total of a file's permissions is also called its access mode.
- Every file (and directory) has an owner and a group. Access rights—read, write, and execute permission—are assigned to these two categories and "others" separately. Only the owner is allowed to set access rights.
- Access rights do not apply to the system administrator (`root`). He may read or write all files.
- File permissions can be manipulated using the `chmod` command.
- Using `chown`, the system administrator can change the user and group assignment of arbitrary files.
- Normal users can use `chgrp` to assign their files to different groups.
- The umask can be used to limit the standard permissions when files and directories are being created.
- The SUID and SGID bits allow the execution of programs with the privileges of the file owner or file group instead of those of the invoker.
- The SGID bit on a directory causes new files in that directory to be assigned the directory's group (instead of the primary group of the creating user).
- The "sticky bit" on a directory lets only the owner (and the system administrator) delete files.
- The `ext` file systems support special additional file attributes.

Bibliography

SUID Dennis M. Ritchie. "Protection of data file contents". US patent 4,135,240.

4

Process Management

Contents

Goals

- Knowing the Linux process concept
- Using the most important commands to query process information
- Knowing how to signal and stop processes
- Being able to influence process priorities

Prerequisites

- Linux commands

4.1 What Is A Process?

A process is, in effect, a "running program". Processes have code that is executed, and data on which the code operates, but also various attributes the operating uses to manage them, such as:

process number

- The unique process number (PID or "process identity") serves to identify the process and can only be assigned to a single process at a time.

parent process number

- All processes know their parent process number, or PPID. Every process can spawn others ("children") that then contain a reference to their procreator. The only process that does not have a parent process is the "pseudo" process with PID 0, which is generated during system startup and creates the "init" process with a PID of 1, which in turn is the ancestor of all other processes in the system.

user
groups

- Every process is assigned to a user and a set of groups. These are important to determine the access the process has to files, devices, etc. (See Section 3.4.) Besides, the user the process is assigned to may stop, terminate, or otherwise influence the process. The owner and group assignments are passed on to child processes.

priority

- The system splits the CPU time into little chunks ("time slices"), each of which lasts only for a fraction of a second. The current process is entitled to such a time slice, and afterwards the system decides which process should be allowed to execute during the next time slice. This decision is made by the appropriate "scheduler" based on the priority of a process.

  In multi-processor systems, Linux also takes into account the particular topology of the computer in question when assigning CPU time to processes—it is simple to run a process on any of the different cores of a multi-core CPU which share the same memory, while the "migration" of a process to a different processor with separate memory entails a noticeable administrative overhead and is therefore less often worthwhile.

other attributes

- A process has other attributes—a current directory, a process environment, …—which are also passed on to child processes.

process file system

You can consult the `/proc` file system for this type of information. This process file system is used to make available data from the system kernel which is collected at run time and presented by means of directories and files. In particular, there are various directories using numbers as names; every such directory corresponds to one process and its name to the PID of that process. For example:

```
dr-xr-xr-x  3   root  root      0  Oct 16 11:11 1
dr-xr-xr-x  3   root  root      0  Oct 16 11:11 125
dr-xr-xr-x  3   root  root      0  Oct 16 11:11 80
```

In the directory of a process, there are various "files" containing process information. Details may be found in the `proc`(5) man page.

job control

The job control available in many shells is also a form of process management—a "job" is a process whose parent process is a shell. From the corresponding shell, its jobs can be controlled using commands like `jobs`, `bg`, and `fg`, as well as the key combinations Ctrl+z and Ctrl+c (among others). More information is available from the manual page of the shell in question, or from the Linup Front training manual, *Introduction to Linux for Users and Administrators*.

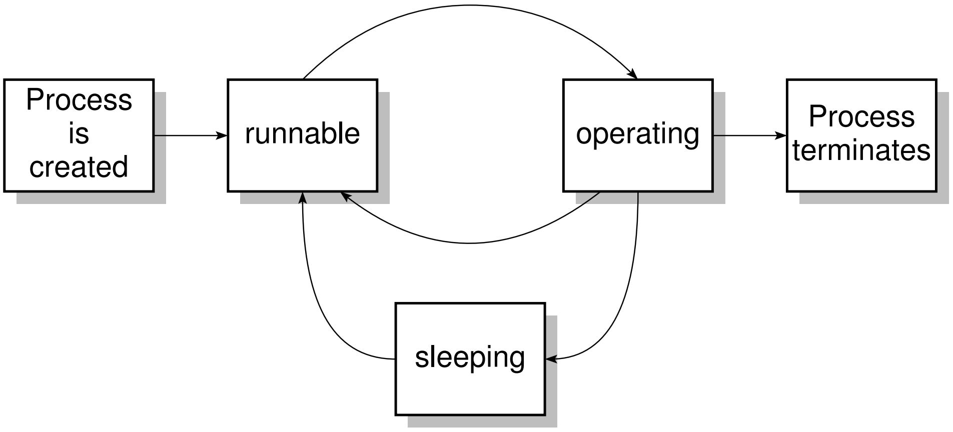


Figure 4.1: The relationship between various process states

Exercises

4.1 [3] How can you view the environment variables of any of your processes? (*Hint:* `/proc` file system.)

4.2 [2] (For programmers.) What is the maximum possible PID? What happens when this limit is reached? (*Hint:* Look for the string "`PID_MAX`" in the files below `/usr/include/linux`.)

4.2 Process States

process state

pre-emptive multitasking

Another important property of a process is its **process state**. A process in memory waits to be executed by the CPU. This state is called "runnable". Linux uses **pre-emptive multitasking**, i. e., a scheduler distributes the available CPU time to waiting processes in pieces called "time slices". If a process is actually executing on the CPU, this state is called "operating", and after its time slice is over the process reverts to the "runnable" state.

From an external point of view, Linux does not distinguish between these two process states; the process in question is always marked "runnable".

return code

It is quite possible that a process requires further input or needs to wait for peripheral device operations to complete; such a process cannot be assigned CPU time, and its state is considered to be "sleeping". Processes that have been stopped by means of Ctrl+z using the shell's job control facility are in state "stopped". Once the execution of a process is over, it terminates itself and makes a **return code** available, which it can use to signal, for example, whether it completed successfully or not (for a suitable definition of "success").

zombies

Once in a while processes appear who are marked as **zombies** using the "Z" state. These "living dead" usually exist only for a brief instant. A process becomes a zombie when it finishes and dies for good once its parent process has queried its return code. If a zombie does not disappear from the process table this means that its parent should really have picked up the zombie's return code but didn't. A zombie cannot be removed from the process table. Because the original process no longer exists and cannot take up neither RAM nor CPU time, a zombie has no impact on the system except for an unattractive entry in the system state. Persistent or very numerous zombies usually indicate programming errors in the parent process; when the parent process terminates they should do so as well.

Zombies disappear when their parent process disappears because "orphaned" processes are "adopted" by the init process. Since the init process spends most of its time waiting for other processes to terminate so that it can collect their return code, the zombies are then disposed of fairly quickly.

Of course, zombies take up room in the process table that might be required for other processes. If that proves a problem, look at the parent process.

Exercises

4.3 [2] Start a `xclock` process in the background. In the `$!` shell variable you will find the PID of that process (it always contains the PID of the most recently launched background process). Check the state of that process by means of the "`grep ^State: /proc/$!/status`" command. Stop the `xclock` by moving it to the foreground and stopping it using Ctrl+z. What is the process state now? (Alternatively, you may use any other long-running program in place of `xclock`.)

4.4 [4] (When going over this manual for the second time.) Can you create a zombie process on purpose?

4.3 Process Information—`ps`

You would normally not access the process information in `/proc` directly but use the appropriate commands to query it.

The `ps` ("process status") command is available on every Unix-like system. Without any otions, all processes running on the current terminal are output. The resulting list contains the process number `PID`, the terminal `TTY`, the process state `STAT`, the CPU time used so far `TIME` and the command being executed.

```
$ ps
  PID TTY STAT TIME COMMAND
  997   1 S    0:00 -bash
 1005   1 R    0:00 ps
$ _
```

There are two processes currently executing on the `tty1` terminal: Apart from the `bash` with PID 997, which is currently sleeping (state "`S`"), a `ps` command is executed using PID 1005 (state "`R`"). The "operating" state mentioned above is not being displayed in `ps` output.

The syntax of `ps` is fairly confusing. Besides Unix98-style options (like `-l`) and GNU-style long options (such as `--help`), it also allows BSD-style options without a leading dash. Here is a selection out of all possible parameters:

`a` ("all") displays all processes with a terminal

`--forest` displays the process hierarchy

`l` ("long") outputs extra information such as the priority

`r` ("running") displays only runnable processes

`T` ("terminal") displays all processes on the current terminal

`U` ⟨*name*⟩ ("user") displays processes owned by user ⟨*name*⟩

`x` also displays processes without a terminal

The unusual syntax of `ps` derives from the fact that AT&T's `ps` traditionally used leading dashes on options while BSD's didn't (and the same option can have quite different results in both flavours). When the big reunification came in System V Release 4, one could hang on to most options with their customary meaning.

If you give ps a PID, only information pertaining to the process in question will be displayed (if it exists):

```
$ ps 1
  PID TTY      STAT   TIME COMMAND
    1 ?        Ss     0:00 init [2]
```

With the -C option, ps displays information about the process (or processes) based on a particular command:

```
$ ps -C konsole
    PID TTY          TIME CMD
   4472 ?        00:00:10 konsole
  13720 ?        00:00:00 konsole
  14045 ?        00:00:14 konsole
```

(Alternatively, grep would help here as well.)

Exercises

4.5 [!2] What does the information obtainable with the ps command mean? Invoke ps without an option, then with the a option, and finally with the ax option. What does the x option do?

4.6 [3] The ps command allows you to determine the output format yourself by means of the -o option. Study the ps(1) manual page and specify a ps command line that will output the PID, PPID, the process state and the command.

4.4 Processes in a Tree—pstree

If you do not want to obtain every bit of information about a process but are rather interested in the relationships between processes, the pstree command is helpful. pstree displays a process tree in which the child processes are shown as depending on their parent process. The processes are displayed by name:

pstree

```
$ pstree
init-+-apache---7*[apache]
     |-apmd
     |-atd
     |-cannaserver
     |-cardmgr
     |-chronyd
     |-cron
     |-cupsd
     |-dbus-daemon-1
     |-events/0-+-aio/0
     |          |-kblockd/0
     |          `-2*[pdflush]
     |-6*[getty]
     |-ifd
     |-inetd
     |-kapmd
     |-kdeinit-+-6*[kdeinit]
     |         |-kdeinit-+-bash---bash
     |         |         |-2*[bash]
     |         |         |-bash---less
     |         |         |-bash-+-pstree
     |         |         |      `-xdvi---xdvi.bin---gs
     |         |         `-bash---emacs---emacsserver
     |         |-kdeinit---3*[bash]
     |         |-kteatime
     |         `-tclsh
     |-10*[kdeinit]
     |-kdeinit---kdeinit
◁◁◁◁◁
```

Identical processes are collected in brackets and a count and "*" are displayed. The most important options of `pstree` include:

`-p` displays PIDs along with process names

`-u` displays process owners' user name

`-G` makes the display prettier by using terminal graphics characters—whether this is in fact an improvement depends on your terminal

You can also obtain an approximated tree structure using "`ps --forest`". The tree structure is part of the `COMMAND` column in the output.

4.5 Controlling Processes—`kill` and `killall`

signals

The `kill` command sends **signals** to selected processes. The desired signal can be specified either numerically or by name; you must also pass the process number in question, which you can find out using `ps`:

```
$ kill -15 4711                        Send signal SIGTERM to process 4711
$ kill -TERM 4711                                               Same thing
$ kill -SIGTERM 4711                                      Same thing again
$ kill -s TERM 4711                                       Same thing again
$ kill -s SIGTERM 4711                                    Same thing again
$ kill -s 15 4711                                               Guess what
```

Here are the most important signals with their numbers and meaning:

`SIGHUP` (1, "hang up") causes the shell to terminate all of its child processes that use the same controlling terminal as itself. For background processes without a controlling terminal, this is frequently used to cause them to re-read their configuration files (see below).

`SIGINT` (2, "interrupt") Interrupts the process; equivalent to the Ctrl+c key combination.

`SIGKILL` (9, "kill") Terminates the process and cannot be ignored; the "emergency brake".

`SIGTERM` (15, "terminate") Default for `kill` and `killall`; terminates the process.

`SIGCONT` (18, "continue") Lets a process that was stopped using `SIGSTOP` continue.

`SIGSTOP` (19, "stop") Stops a process temporarily.

`SIGTSTP` (20, "terminal stop") Equivalent to the Ctrl+z key combination.

You shouldn't get hung up on the signal numbers, which are not all guaranteed to be the same on all Unix versions (or even Linux platforms). You're usually safe as far as 1, 9, or 15 are concerned, but for everything else you should rather be using the names.

Unless otherwise specified, the signal `SIGTERM` ("terminate") will be sent, which (usually) ends the process. Programs can be written such that they "trap" signals (handle them internally) or ignore them altogether. Signals that a process neither traps nor ignores usually cause it to crash hard. Some (few) signals are ignored by default.

The `SIGKILL` and `SIGSTOP` signals are not handled by the process but by the kernel and hence cannot be trapped or ignored. `SIGKILL` terminates a process without giving it a chance to object (as `SIGTERM` would), and `SIGSTOP` stops the process such that it is no longer given CPU time.

daemons

`kill` does not always stop processes. Background processes which provide system services without a controlling terminal—daemons—usually reread their configuration files without a restart if they are sent `SIGHUP` ("hang up").

You can apply `kill`, like many other Linux commands, only to processes that you actually own. Only `root` is not subject to this restriction.

Sometimes a process will not even react to `SIGKILL`. The reason for this is either that it is a zombie (which is already dead and cannot be killed again) or else blocked in a system call. The latter situation occurs, for example, if a process waits for a write or read operation on a slow device to finish.

`killall`

An alternative to the `kill` command is the `killall` command. `killall` acts just like `kill`—it sends a signal to the process. The difference is that the process must be named instead of addressed by its PID, and that all processes of the same name are signalled. If no signal is specified, it sends `SIGTERM` by default (like `kill`). `killall` outputs a warning if there was nothing to signal to under the specified name.

The most important options for `killall` include:

`-i` `killall` will query you whether it is actually supposed to signal the process in question.

`-l` outputs a list of all available signals.

`-w` waits whether the process that was signalled actually terminates. `killall` checks every second whether the process still exists, and only terminates once it is gone.

Be careful with `killall` if you get to use Solaris or BSD every now and then. On these systems, the command does exactly what its name suggests—it kills *all* processes.

Exercises

4.7 [2] Which signals are being ignored by default? (*Hint:* `signal`(7))

4.6 `pgrep` and `pkill`

As useful as `ps` and `kill` are, as difficult can it be sometimes to identify exactly the right processes of interest. Of course you can look through the output of `ps` using `grep`, but to make this "foolproof" and without allowing too many false positives is at least inconvenient, if not tricky. Nicely enough, Kjetil Torgrim Homme has taken this burden off us and developed the `pgrep` program, which enables us to search the process list conveniently. A command like

```
$ pgrep -u root sshd
```

will, for example, list the PIDs of all `sshd` processes belonging to `root`.

By default, `pgrep` restricts itself to outputting PIDs. Use the `-l` option to get it to show the command name, too. With `-a` it will list the full command line.

The `-d` option allows you to specify a separator (the default is "`\n`"):

```
$ pgrep -d, -u hugo bash
4261,11043,11601,12289
```

You can obtain more detailed information on the processes by feeding the PIDs to `ps`:

```
$ ps up $(pgrep -d, -u hugo bash)
```

(The `p` option lets you give `ps` a comma-separated list of PIDs of interest.)

`pgrep`'s parameter is really an (extended) regular expression (consider `egrep`) which is used to examine the process names. Hence something like

```
$ pgrep '^([bd]a|t?c|k|z|)sh$'
```

will look for the common shells.

Normally `pgrep` considers only the process name (the first 15 characters of the process name, to be exact). Use the `-f` option to search the whole command line.

You can add search criteria by means of options. Here is a small selection:

`-G` Consider only processes belonging to the given group(s). (Groups can be specified using names or GIDs.)

`-n` Only display the newest (most recently started) of the found processes.

`-o` Only display the oldest (least recently started) of the found processes.

`-P` Consider only processes whose parent processes have one of the given PIDs.

`-t` Consider only processes whose controlling terminal is listed. (Terminal names should be given without the leading "`/dev/`".)

`-u` Consider only processes with the given (effective) UIDs.

If you specify search criteria but no regular expression for the process name, all processes matching the search criteria will be listed. If you omit both you will get an error message.

The `pkill` command behaves like `pgrep`, except that it does not list the found processes' PIDs but sends them a signal directly (by default, `SIGTERM`). As in `kill` you can specify another signal:

```
# pkill -HUP syslogd
```

The `--signal` option would also work:

```
# pkill --signal HUP syslogd
```

The advantage of `pkill` compared to `killall` is that `pkill` can be much more specific.

Exercises

4.8 [!1] Use `pgrep` to determine the PIDs of all processes belonging to user `hugo`. (If you don't have a user `hugo`, then specify some other user instead.)

4.9 [2] Use two separate terminal windows (or text consoles) to start one "`sleep 60`" command each. Use `pkill` to terminate (a) the first command started, (b) the second command started, (c) the command started in one of the two terminal windows.

4.7 Process Priorities—`nice` and `renice`

In a multi-tasking operating system such as Linux, CPU time must be shared among various processes. This is the scheduler's job. There is normally more than one runnable process, and the scheduler must allot CPU time to runnable processes according to certain rules. The deciding factor for this is the **priority** of a process. The priority of a process changes dynamically according to its prior behaviour—"interactive" processes, i. e, ones that do I/O, are favoured over those that just consume CPU time.

priority

As a user (or administrator) you cannot set process priorities directly. You can merely ask the kernel to prefer or penalise processes. The "nice value" quantifies the degree of favouritism exhibited towards a process, and is passed along to child processes.

A new process's nice value can be specified with the `nice` command. Its syntax is

`nice`

```
nice [-⟨nice value⟩] ⟨command⟩ ⟨parameter⟩ …
```

(`nice` is used as a "prefix" for another command).

The possible nice values are numbers between −20 and +19. A negative nice value increases the priority, a positive value decreases it (the higher the value, the "nicer" you are towards the system's other users by giving your own processes a lower priority). If no nice value is specified, the default value of +10 is assumed. Only `root` may start processes with a negative nice value (negative nice value are not generally nice for other users).

possible nice values

The priority of a running process can be influenced using the `renice` command. You call `renice` with the desired new nice value and the PID (or PIDs) of the process(es) in question:

`renice`

```
renice [-⟨nice value⟩] ⟨PID⟩ …
```

Again, only the system administrator may assign arbitrary nice values. Normal users may only increase the nice value of their own processes using `renice`—for example, it is impossible to revert a process started with nice value 5 back to nice value 0, while it is absolutely all right to change its nice value to 10. (Think of a ratchet.)

Exercises

4.10 [2] Try to give a process a higher priority. This may possibly not work—why? Check the process priority using `ps`.

4.8 Further Process Management Commands—`nohup` and `top`

When you invoke a command using `nohup`, that command will ignore a `SIGHUP` signal and thus survive the demise of its parent process:

Ignoring `SIGHUP`

```
nohup ⟨command⟩ …
```

The process is not automatically put into the background but must be placed there by appending a `&` to the command line. If the program's standard output is a terminal and the user has not specified anything else, the program's output together with its standard error output will be redirected to the `nohup.out` file. If the current directory is not writable for the user, the file is created in the user's home directory instead.

top

top unifies the functions of many process management commands in a single program. It also provides a process table which is constantly being updated. You can interactively execute various operations; an overview is available using h. For example, it is possible to sort the list according to several criteria, send signals to processes (k), or change the nice value of a process (r).

Commands in this Chapter

kill	Terminates a background process	bash(1), kill(1)	58
killall	Sends a signal to all processes matching the given name	killall(1)	59
nice	Starts programs with a different *nice* value	nice(1)	61
nohup	Starts a program such that it is immune to SIGHUP signals	nohup(1)	61
pgrep	Searches processes according to their name or other criteria	pgrep(1)	59
pkill	Signals to processes according to their name or other criteria	pkill(1)	60
ps	Outputs process status information	ps(1)	56
pstree	Outputs the process tree	pstree(1)	57
renice	Changes the *nice* value of running processes	renice(8)	61
top	Screen-oriented tool for process monitoring and control	top(1)	61

Summary

- A process is a program that is being executed.
- Besides a program text and the corresponding data, a process has attributes such as a process number (PID), parent process number (PPID), owner, groups, priority, environment, current directory, …
- All processes derive from the init process (PID 1).
- ps can be used to query process information.
- The pstree command shows the process hierarchy as a tree.
- Processes can be controlled using signals.
- The kill and killall commands send signals to processes.
- The nice and renice commands are used to influence process priorities. ulimit limits the resource usage of a process.
- top is a convenient user interface for process management.

5

Hardware

Contents

Goals

- Knowing the basics of Linux hardware support on PCs
- Being familiar with terms like BIOS, UEFI, PCI, USB and others
- Mastering the basics of dynamic peripheral support via `udev`

Prerequisites

- Basic Linux knowledge
- A basic familiarity with PC hardware and its support by other operating systems is helpful

5.1 Fundamentals

Linux supports a very broad spectrum of system architectures and platforms: The Linux kernel is available for all of today's common microprocessors, and Linux runs on computers from modest PDAs to the largest mainframes. Linux is also an important part of many devices that one would not associate with it at first sight—digital cameras and camcorders, routers, television sets and set-top boxes, GPS navigation devices and many more—and that sometimes use "unusual" hardware. The most common Linux systems by far that are used as "computers" in the proper sense of the word are based on the x86 PC architecture founded by IBM and Intel.

The x86 PC architecture by IBM and Intel is also the only one relevant to LPIC-1 certification, although it should be said that architecture-specific questions form a very small part of the exam today (they used to be much more prevalent).

Architecture

Computers today possess one or more processors (CPUs), memory for running programs and data (RAM), and secondary storage for files (disks, SSDs). Added to this are various peripheral devices for input (keyboard, mouse, graphics tablet, webcam, …) and output (graphics "card"—today often part of the CPU—, audio, …) and much more. These peripherals are either part of the "motherboard" or are added via various interfaces (PCIe, USB, SATA, SCSI, …). Ethernet or WLAN are used for networking. To be able to boot, a computer also needs "firmware" in read-only memory (ROM)[1]

On PCs, the firmware is either called "BIOS" (on older systems) or "UEFI". In former times—during the age of MS-DOS—the BIOS served as the interface between the operating system proper (DOS) and the computer's hardware. Today it is only used to initialise the computer at boot and to find and launch a "real" operating system. Modern operating systems tend to disregard the BIOS completely.

UEFI is a more modern implementation of the same idea, which does away with some nasty shortcomings of the BIOS—which is after all a remnant of the 1980s—and offers more extensive options, e. g., when dealing with several operating systems (or operating system versions) on the same computer or attempting to secure the system's software against malicious manipulation.

New computers are based on UEFI, but can frequently pretend to have a traditional BIOS if desired, so that older operating systems can be supported.

While in former times extensive BIOS configuration might have been required before a computer would run Linux efficiently, this is usually no longer a problem today. You may only have to perform very few settings, e. g., to set the date and time.

hardware clock

You will need to decide whether the hardware clock should be set to "zone time" (such as Pacific Standard Time for the west coast of North America) or "universal time" (UTC). Linux—which consults the hardware clock only when starting—can handle both, but must know where it is at. UTC is preferable on machines running only Linux, while zone time may be useful if the computer is occasionally running other alternative operating systems.

Hard disks

On BIOS-based computers, the BIOS must know about the hard disk the system is to be booted from (at minimum). You can generally tell the BIOS about the properties of the various disk drives in the system, and whether

[1] Technically speaking, most computers today use "flash ROM", which is rewritable using appropriate tools, instead of "true" ROM. This doesn't change the principle.

the BIOS should see them at all—these settings have no bearing on the later use of the disk(s) by Linux, hence it is possible to trick old-fashioned BIOS implementations which cannot handle modern large disk drives, simply by deactivating the disk drive in the BIOS. Linux can access it later, but you won't be able to boot from that disk.

disk geometry

Within the BIOS setup, you can enter a hard disks>geometrydisk geometry for each disk, i. e., the number of read/write heads, cylinders per head, and sectors per cylinder of the disk. Today's disks have more sectors on the outside cylinders than on those near the centre of the platters, and no longer bother with the "cylinder/head/sector" or "CHS" model of addressing disk sectors but simply count sectors off sequentially from 0 to …—the LBA mode ("linear block access"). Even so, every disk still claims a completely fabricated geometry matching the disk capacity in order to make older BIOSes and DOS happy. With modern BIOS implementations, you can put disks into LBA mode explicitly, but as long as the system boots using the default values this is not required.

LBA mode

peripherals

The BIOS can also be used to enable or disable various types of peripherals:

- Assignment of serial ports from the operating system's point of view (usually `COM1:` or some such) to existing serial ports, IrDA ports, etc.
- Support for USB, in particular USB keyboards and mice
- Internal graphics and sound support
- Energy and status management (APM, ACPI)

Within the scope of this course it is not feasible (and, fortunately, not necessary for the purposes of LPIC-1) to give more detailed advice. Keep in mind that these settings exist and that, if some device persistently refuses to get to work on Linux, you should check what the BIOS has to say on the matter. It is possible for the device in question to be disabled at the BIOS level, or for a BIOS-enabled device to conflict with the device that is really wanted.

5.2 Linux and PCI (Express)

During the 30 years or more since the first IBM PC was marketed, the "internals" of the hardware have changed radically in almost all aspects. Not least the bus system connecting the CPU and memory with the most important peripherals has undergone several metamorphoses since the original "ISA" bus; the current standard is called **PCI Express** (PCIe) and has all but ousted its predecessors.

PCI Express

PCIe is vaguely based on PCI and has taken over many PCI concepts (such as device codes for hardware detection, see below), but its electrical properties are completely different.

Unlike its predecessors, PCIe is conceptually a serial bus which allows independent point-to-point connections between different devices. Formerly, all devices shared a parallel bus, which put a cap on the maximum speeds that could be achieved.

In the interest of efficiency, PCIe allows the transfer of data packets over several connections ("lanes"), but that does not take away from the general principle.

hardware detection

A major advantage of PCIe is that automatic hardware detection is possible. Every device connected to the bus reports a code specifying its type, manufacturer, and model. You can query this information using the `lspci` command (Figure 5.1). At the beginning of each line there is the "PCI ID" of each device, giving its position on the PCI bus.

```
# lspci
00:00.0 Host bridge: Intel Corp Core Processor DRAM Controller (rev 12)
00:01.0 PCI bridge: Intel Corp Core Processor PCI Express x16 Root Port
00:16.0 Communication controller: Intel Corporation 5 Series/3400▷
 ◁ Series Chipset HECI Controller (rev 06)
00:16.3 Serial controller: Intel Corp 5 Series/3400 Series Chipset KT ▷
 ◁ Controller (rev 06)
00:19.0 Ethernet controller: Intel Corporation 82577LM Gigabit Network▷
 ◁ Connection (rev 06)
00:1a.0 USB controller: Intel Corporation 5 Series/3400 Series▷
 ◁ Chipset USB2 Enhanced Host Controller (rev 06)
00:1b.0 Audio device: Intel Corporation 5 Series/3400 Series Chipset▷
High Definition Audio (rev 06)
◁◁◁◁◁
01:00.0 VGA compatible controller: NVIDIA Corporation GT218M▷
 ◁ [NVS 3100M] (rev a2)
01:00.1 Audio device: NVIDIA Corporation High Definition Audio▷
 ◁ Controller (rev a1)
◁◁◁◁◁
```

Figure 5.1: Output of lspci on a typical x86-based PC

Linux uses the device codes to select and configure drivers for the various peripheral devices it detects. The kernel and the udev infrastructure collaborate on this. We shall look at this in more detail later.

In former times you would have had to guess, but that could cause problems including system crashes, if a driver poked the "right" wrong device. In cases of doubt, the system administrator (you) would have to perform tedious manual interventions!

lspci supports some interesting options: "lspci -v" provides more verbose output:

```
# lspci -v
00:00.0 Host bridge: Intel Corp Core Processor DRAM Controller (rev 12)
        Subsystem: Hewlett-Packard Company Device 172b
        Flags: bus master, fast devsel, latency 0
        Capabilities: [e0] Vendor Specific Information: Len=0c <?>

00:01.0 PCI bridge: Intel Corporation Core Processor PCI Express x16▷
 ◁ Root Port (rev 12) (prog-if 00 [Normal decode])
        Flags: bus master, fast devsel, latency 0
        Bus: primary=00, secondary=01, subordinate=01, sec-latency=0
        I/O behind bridge: 00005000-00005fff
        Memory behind bridge: d2000000-d30fffff
◁◁◁◁◁
```

In former times, similar information used to be available from the /proc/pci "file", which is no longer provided by default by newer kernels. The official method of obtaining PCI data is lspci.

"lspci -t" gives a tree-like representation of the connections between the various components:

```
# lspci -t
-+-[0000:ff]-+-00.0
```

```
 |          +-00.1
 |          +-02.0
 |          +-02.1
 |          +-02.2
 |          \-02.3
 \-[0000:00]-+-00.0
            +-01.0-[01]--+-00.0
            |            \-00.1
            +-16.0
            +-16.3
            +-19.0
◁◁◁◁◁
```

This tells you, for example, that the chipset's "PCI bridge" (device `0000:00:01.0`) provides the connection to the "VGA compatible controller) (device 0000:01:00.0) and the "audio device" (`0000:01:00.1`, more precisely the graphic card's HDMI audio output). The Ethernet adapter (device `0000:01:19.0`) is also connected via PCIe.

device codes

Finally, "`lspci -n`" outputs the device codes directly, without looking up their meaning in the PCI database:

```
# lspci -n
00:00.0 0600: 8086:0044 (rev 12)
00:01.0 0604: 8086:0045 (rev 12)
00:16.0 0780: 8086:3b64 (rev 06)
00:16.3 0700: 8086:3b67 (rev 06)
00:19.0 0200: 8086:10ea (rev 06)
00:1a.0 0c03: 8086:3b3c (rev 06)
00:1b.0 0403: 8086:3b56 (rev 06)
◁◁◁◁◁
```

Exercises

5.1 [!2] Examine the hardware structure of your system using commands such as "`lspci -v`" or "`lspci -t`". What can you find out?

5.2.1 USB

The "Universal Serial Bus", or USB, is nowadays used to connect practically all external peripheral devices that do not use wireless technology (such as WLAN or Bluetooth). The goal is a "legacy-free" computer that can do without the erstwhile multitude of device-specific connectors for the keyboard, mouse, printer, modem, …

USB traditionally uses a "foolproof" cabling concept with different sockets for computers ("A") and peripherals ("B") and appropriate plugs at the end of the cables, in order to avoid mistakes. In addition, USB allows unplugging and re-plugging of devices when powered up (even though this is generally not a great idea when dealing with hard disks or USB thumbdrives).

The newest fad (2015) are "USB C" sockets, which are identical between computers and peripherals and do not care which way the plug is inserted (which is a constant nuisance especially with USB "A" plugs—Pro tip, the USB logo on the plug is always supposed to point towards you). USB C connections aren't just useful for USB, but also many other things (like powerful charging currents, graphics signals, and what else one might think of), but the system is still fairly new.

Table 5.1: USB standards

Version	since	Max speed	Devices
1.1	1998	1,5 MiBit/s	Keyboards, mice, modems, audio devices, …
		12 MiBit/s	10-MiBit ethernet, disks, …
2.0	2000	480 MiBit/s	disks, 100-MiBit ethernet, video …
3.0	2008	5 GiBit/s	disks, graphics, …
3.1	2013	10 GiBit/s	PCIe

USB controllers

USB is now part of all new PCs. This means that every PC contains one or more USB controllers, each of which can manage up to 127 USB devices.

USB hubs

Since PCs do not feature that many USB sockets, USB allows the tree-like connection of devices via USB hubs. You can therefore use one USB socket in your computer to connect a hub with several additional sockets; the devices connected to the hub will share the connection to your computer. You can even connect further hubs to a hub (although this may not always be a bright idea in actual practice).

Since its introduction in 1996, the USB standard has continuously been improved, and newer versions pushed the maximum possible transmission speed ever higher. The current incarnation (2015) is version 3.1, which basically makes it possible to drive external devices at PCIe bus speeds (which is nice, e. g., for external graphics cards).

Newer USB implementations are, in principle, backwards-compatible. You can connect USB-2.0 devices to a USB-1.1 bus and operate them at USB-1.1 speeds. Conversely, you can connect USB-1.1 devices to a USB-2.0 hub or controller, but they won't be any faster—but they will not slow down "real" USB-2.0 devices, either.

USB-3.*x* devices actually support USB 2.0 and USB 3.*x* simultaneously on separate leads in the cable and separate contacts in the connectors. You can plug a USB-2.0 plug into a USB-3.*x* socket on your computer, and that will yield a USB-2.0 connection; conversely, you can plug a USB-3.*x* plug into a USB-2.0 socket, which will also lead to a USB-2.0 connection. For "genuine" USB 3.*x*, your computer must support USB 3.*x*, and you need USB-3.*x* devices and matching cables.

Enumeration
descriptor
class
subclass

When a USB device is connected to the bus, it is assigned a number between 1 and 127 as its device number, and the computer reads the device's "descriptor". The descriptor specifies the type of device, the type of USB port it supports, and so on. In particular, each device belongs to a class of devices that are handled in a similar fashion by the computer—such as the "human interface devices", i. e., devices that serve as input devices for people: keyboards, mice, joysticks, but also switches, dials, phone keyboards, steering wheels, data gloves and so on. Classes may have subclasses that narrow the type of device down even more.

You can check out which USB devices are connected to your system by using the `lsusb` command. Like `lspci`, it outputs the device numbers and names:

```
$ lsusb
Bus 002 Device 002: ID 8087:0020 Intel Corp. Integrated Rate ▷
 ◁ Matching Hub
Bus 002 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 001 Device 006: ID 04f2:b15e Chicony Electronics Co., Ltd
Bus 001 Device 008: ID 046d:0807 Logitech, Inc. Webcam B500
Bus 001 Device 010: ID 046a:0023 Cherry GmbH CyMotion Master Linux▷
```

```
 ◁  Keyboard G230
Bus 001 Device 009: ID 046d:c52b Logitech, Inc. Unifying Receiver
Bus 001 Device 007: ID 05e3:0608 Genesys Logic, Inc. USB-2.0 4-Port HUB
◁◁◁◁◁
```

Using the `-v` option, the program becomes noticeable more “chatty”:

```
# lsusb -v
Bus 002 Device 002: ID 8087:0020 Intel Corp. Integrated Rate ▷
 ◁ Matching Hub
Device Descriptor:
  bLength                18
  bDescriptorType         1
  bcdUSB               2.00
  bDeviceClass            9 Hub
  bDeviceSubClass         0 Unused
  bDeviceProtocol         1 Single TT
  bMaxPacketSize0        64
  idVendor           0x8087 Intel Corp.
  idProduct          0x0020 Integrated Rate Matching Hub
  bcdDevice            0.00
  iManufacturer           0
  iProduct                0
  iSerial                 0
  bNumConfigurations      1
  Configuration Descriptor:
    bLength                 9
    bDescriptorType         2
    wTotalLength           25
    bNumInterfaces          1
    bConfigurationValue     1
    iConfiguration          0
    bmAttributes         0xe0
      Self Powered
      Remote Wakeup
    MaxPower                0mA
    Interface Descriptor:
      bLength                 9
      bDescriptorType         4
      bInterfaceNumber        0
      bAlternateSetting       0
◁◁◁◁◁
```

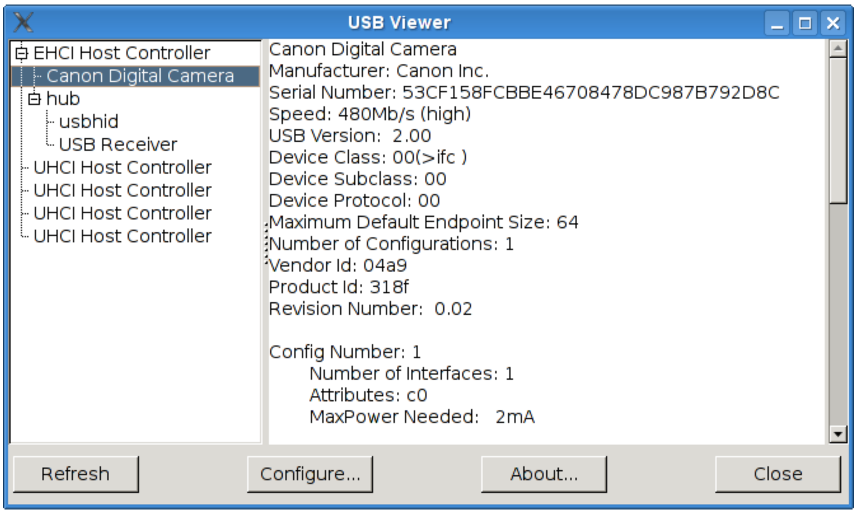
The output of “`lsusb -v`” is quite extensive but also fairly unreadable. The `usbview` program presents the data rather more nicely (Figure 5.2).

If you would like to know where `lsusb` gets its encyclopedic knowledge of USB peripherals: Take a look at the `/var/lib/usbutils/usb.ids` file.

5.3 Peripherals

5.3.1 Overview

The interesting question that remains is how Linux can handle almost arbitrary peripheral devices—both those built in to the computer as well as those that can come and go while the machine is running.



```
Canon Digital Camera
Manufacturer: Canon Inc.
Serial Number: 53CF158FCBBE46708478DC987B792D8C
Speed: 480Mb/s (high)
USB Version:  2.00
Device Class: 00(>ifc )
Device Subclass: 00
Device Protocol: 00
Maximum Default Endpoint Size: 64
Number of Configurations: 1
Vendor Id: 04a9
Product Id: 318f
Revision Number:  0.02

Config Number: 1
      Number of Interfaces: 1
      Attributes: c0
      MaxPower Needed:   2mA
```

Figure 5.2: The `usbview` program

Talking to peripheral devices is the Linux kernel's natural business. Its job is to find and initialise connected peripherals and to allow userspace programs controlled access to them. This of course includes appropriate rights assignment, such that only suitably privileged users can access peripherals directly.

In real life, the Linux kernel likes to share the device management work with userspace programs. This makes sense, because it both keeps the kernel lean and also avoids the hard-wiring of too much functionality inside the kernel if it is better placed in userspace programs, where it is easier to program and customise.

By way of an example: Simple printers are nowadays connected to the computer via USB. If a new printer is connected, the Linux kernel finds out about this and can use the USB manufacturer and model codes to figure out that the newly recognised peripheral is a printer, and which model of which manufacturer it is. The result of this initialisation process is a device file below `/dev` which userspace programs can use to contact the printer.—Which doesn't mean that arbitrary user applications get to talk to the printer directly. Direct access to its device file is restricted, via suitable access rights, to the printer subsystem (CUPS). The CUPS configuration routine can access the Linux kernel's information on the printer's make and model, and select an appropriate printer driver. User applications—such as LibreOffice—print documents by passing them to CUPS in a suitable format (PostScript or PDF). CUPS then converts the job into a format that the printer understands, and passes it via the device file to the Linux kernel, which then routes it to the actual printer via USB.

5.3.2 Devices and Drivers

loadable modules

Linux supports the wide variety of peripheral devices available for PCs by means of drivers which are usually furnished as loadable modules. These loadable modules can be found in subdirectories of `/lib/modules` and can be loaded into the kernel manually using the `modprobe` command:

```
# modprobe foo
```

for example locates and loads the `foo.ko`. Should the desired module depend on the functionality of other modules which are not currently loaded, `modprobe` arranges for them to be loaded as well.

Linux uses loadable modules not just to implement device drivers in the proper sense of the word, but also for network protocols, the packet filter infrastructure, and many more system features that are not required on every system or at all times. Modules also make kernel development easier and are therefore popular with programmers.

The `lsmod` command gives an overview of the currently-loaded modules.

```
$ lsmod
Module                  Size  Used by
nfs                   213896  1
lockd                  54248  1 nfs
nfs_acl                 2912  1 nfs
sunrpc                162144  10 nfs,lockd,nfs_acl
tun                     8292  1
michael_mic             2304  4
arc4                    1824  4
◁◁◁◁◁
```

You can also try to remove a loaded module using “`modprobe -r`”:

```
# modprobe -r foo
```

(Whether this actually works depends on your kernel.) If the module to be removed is the only one depending on another module, `modprobe` tries to remove the other module, too.

Automatic loading on demand

You will only rarely have to run `modprobe` manually, as the kernel can mostly take care of loading modules on demand. The basis for this is the `/etc/modprobe.conf` file (or the files in the `/etc/modprobe.d` directory), which contains entries like

```
alias block-major-3-*  ide_generic                         Block device
alias char-major-10-1 psmouse                  Character-oriented device
alias net-pf-16-proto-8 scsi_transport_iscsi            Network protocol
```

(and various others). These entries connect device files in `/dev` such as

```
$ ls -l /dev/hda1 /dev/psaux
brw-rw---- 1 root disk  3, 1 Jan 22 02:03 /dev/hda1
crw-rw---- 1 root root 10, 1 Jan 22 02:03 /dev/psaux
```

to the corresponding drivers—`block-major-3-*` means nothing other that for block-oriented device files with a major device number of 3 and arbitrary minor device number (such as `/dev/hda1` in our example), the `ide_generic` driver module is appropriate, while access to `/dev/psaux`, a character-oriented device file with numbers (10, 1) is managed through the driver named by `char-major-10-1`, namely `psmouse`. Whenever the device file in question is first accessed, the Linux kernel tries to locate and enable the corresponding driver module.

By now you are surely asking yourself how a driver like `ide_generic` can be read on demand from an IDE disk, if Linux requires it to talk to the disk in the first place. The answer to that is that this driver, when booting from IDE disk, is not really read from `/lib/modules`, but that the kernel obtained it earlier on from the “initial RAM disk”, which the boot loader fetched from disk together with the kernel itself (by means of the BIOS, hence without involving Linux). (More detail is in Chapter 8).

With `modprobe` and the corresponding configuration files, you can do many other strange and wonderful things that are far beyond the scope of this discussion. You will find more about these topics from the Linup Front training manual, *Linux System Adaptation*.

5.3.3 The /sys Directory

sysfs

Up to and including Linux 2.4, the /proc directory represented the only way to access details of the kernel and system configuration. However, the kernel developers disliked the uncontrolled growth of entries under /proc, in particular those whose purpose did not have anything whatsoever to do with processes (the original intent of the directory). For this reason, the kernel developers decided to move, in the medium to long term, those aspects of /proc that didn't have anything to do with process management to a new pseudo file system, sysfs, where stricter rules should apply. The sysfs is usually mounted on /sys, and is available from the Linux kernel version 2.6 onwards.

connection type

In particular, /sys/bus allows accessing devices depending on their connection type ("bus"; pci, usb, scsi, …). File /sys/devices/ also allows accessing devices, only the sort order is dfferent (device type, e. g., PCI bus address). This redundancy is implemented by means of symbolic links:

```
$ ls -ld /sys/devices/pci0000:00/0000:00:07.2/usb1
drwxr-xr-x  6 root root 0 Jun 27 19:35 ▷
 ◁ /sys/devices/pci0000:00/0000:00:1a.0/usb1
$ ls -ld /sys/bus/usb/devices/usb1
lrwxrwxrwx  1 root root 0 Jun 27 19:35 ▷
 ◁ /sys/bus/usb/devices/usb1
```

In the example you see the directory allowing access to the data of the first USB connector. This is, in fact, the same directory. Such a directory contains various files with information about the device in question:

```
$ ls /sys/bus/usb/devices/usb1
1-0:1.0              bmAttributes        devpath      remove
1-1                  bMaxPacketSize0     driver       serial
authorized           bMaxPower           ep_00        speed
authorized_default   bNumConfigurations  idProduct    subsystem
avoid_reset_quirk    bNumInterfaces      idVendor     uevent
bcdDevice            busnum              manufacturer urbnum
bConfigurationValue  configuration       maxchild     version
bDeviceClass         descriptors         power
bDeviceProtocol      dev                 product
bDeviceSubClass      devnum              quirks
$ cat /sys/bus/usb/devices/usb1/product
EHCI Host Controller
```

assignment of interfaces

One disadvantage of the former kernel concept was the uncertainty about the assignment of interfaces to devices, such as the device files. The "numbering" of devices (as in, sda, sdb, sdc, …, or eth0, eth1, …) was not easy to reproduce. sysfs, on the other hand, lets us assign interfaces to devices in an unambiguous fashion.

You will find the interfaces in the directories /sys/block (block devices) and /sys/class (character devices, more or less):

```
$ ls /sys/block/
dm-0  dm-2  dm-4  dm-6   loop1  loop3  loop5  loop7  sr0
dm-1  dm-3  dm-5  loop0  loop2  loop4  loop6  sda
$ ls /sys/class/
ata_device  dmi          mem            regulator    tty
ata_link    firmware     misc           rfkill       vc
ata_port    graphics     mmc_host       rtc          video4linux
backlight   hidraw       net            scsi_device  vtconsole
bdi         hwmon        pci_bus        scsi_disk    wmi
block       i2c-adapter  pcmcia_socket  scsi_host
bluetooth   ieee80211    power_supply   sound
```

```
bsg         input       ppdev        spi_master
dma         leds        printer      thermal
```

An advantage: This lists only the devices that are actually on the system.

Interfaces are assigned to block devices by means of symbolic links; there is a file called `device`, like

```
$ ls -l /sys/block/sda/device
lrwxrwxrwx  1 root root 0 Jun 27 19:45 /sys/block/sda/device▷
 ◁ -> ../../0:0:0:0
```

5.3.4 udev

In former times, Linux distributors used to create, as part of the installation process, a `/dev` directory that was filled with device files for all the peripherals under the sun. In the meantime, this approach is deprecated in favour of the idea of dynamically creating device files only for those devices that are actually available.

The Linux infrastructure for this is called `udev`, short for “userspace `/dev`”. It was implemented by Greg Kroah-Hartman and Kay Sievers (among others) and is available in its current form since kernel 2.6.13. `udev` consists mainly of a library called `namedev`, which deals with assigning names to devices, and a daemon called `udevd` which does (or delegates) the actual work.

`udevd`

With `udev`, the `/dev` directory on disk only contains the essential entries. Fairly soon after the system is started, a `tmpfs` file system (i. e., a RAM disk, see Section 7.1.6) is mounted over `/dev`, in which `udev` places the appropriate device files. To this end, the `udevd` is started, which listens to “uevents”—event reports by the kernel concerning devices that have been added or removed. Each such report is compared to a set of rules which decides under which name the device should appear in the `/dev` directory managed by `udev`, and which can also execute additional actions, like uploading “firmware” available as a binary file to the device in order to initialise it. These rules can be found in `/etc/udev`.

uevents

firmware

There is nothing wrong with having a device appear in `/dev` under several names. Your digital camera, for example, might be assigned the name `/dev/camera` in addition to `/dev/sd`“whatever”. It all comes down to a question of rules.

`udev` can also deal with devices that do not correspond to device files in `/dev`, such as network cards.

An interesting side effect of this approach is that it was originally designed to deal with devices that come and go at runtime—to be exact, at some point after `udevd` was started (“hotplugging”). The same method, however, is also used under the guise of “coldplugging” when the system is originally started, to initialise the devices that are already available at that time. Of course the uevents the kernel sends to register these devices cannot be processed before `udevd` is running. Still, the kernel makes it possible to re-send these uevents on demand, and exactly that happens after `udevd` has become available, in order to add the devices to the dynamic `/dev` directory. Accordingly, there is no difference between device initialisation using “hotplugging” and “coldplugging”.

To allow this, there is a device called `uevent` in each device’s entry below `/sys`. You just need to write the string `add` to that file to trigger the corresponding uevent.

`udev` isn’t the first attempt to include such an infrastructure in Linux. The `devfs` proposed by Richard Gooch did a similar thing but inside the kernel rather than outside (like `udev`). `devfs` never gathered a large following in the community, since various matters of policy concerning, e. g., device naming had been hard-wired into `devfs` (and hence in the kernel) and not all developers agreed with that. `udev` exposes its rules by means of editable configuration files and is therefore much more flexible. There used to be an infrastructure called `hotplug` to handle dynamic registration and de-registration of devices, but this has been superseded by `udev`.

5.3.5 Device Integration and D-Bus

application programs

The final piece of the puzzle brings in application programs. How do you set things up such that plugging in your digital camera will start your photo management program to download any new pictures? Or how does the icon for your USB disk get placed on the backdrop of your graphical desktop environment? One possible answer to this is "udisks".

Formerly this used an infrastructure called HAL (short for "hardware abstraction layer"). HAL, however, proved too complicated and limited and is no longer being developed further.

Udisks is one of the successor projects, and deals exclusively with storage devices such as external disks and USB thumb drives; there are other projects for other types of device. Large parts of HAL have also been assimilated into `udev`.

HAL

The idea behind HAL is to aggregate information from various sources: The kernel (via `udev`) reports that a new device is available and which one. The device itself can be asked, as can default settings for the device that the user once made in their graphical environment.

This aggregation is necessary because the kernel may not actually know everything worthwhile about a device. Many digital cameras, for example, register with the system as "hard disks", and the information that they should really be treated as cameras must come from elsewhere.

D-Bus

Udisks and similar project are based on D-Bus, a simple system for interprocess communication that is supported by most desktop environments. Programs can tell D-Bus about services that they want to offer to other programs. Programs can also wait for events. (It doesn't matter if several programs wait for the same event; all interested parties are notified.) D-Bus has two major application areas:

- Communication between programs in the same graphical environment. A possible application might be for your telephony program to automatically mute your music player when a call comes in.
- Communication between the operating system and your graphical environment.

Udisks consists of two components, a background service (`udisksd`) and a user-accessible tool (`udisksctl`) which can communicate with the background service. The background service is accessible through the D-Bus and will be started on demand whenever a program issues a D-Bus query to udisks. This way programs can find out which storage devices are available as well as execute operations such as mounting or unmounting devices. This includes appropriate privilege checks; users may, for example, be allowed to plug USB thumb drives into their own computers but will be denied access to USB thumb drives on other "seats".

Here is an example of a `udisksctl` query about a USB thumb drive:

```
$ udisksctl status
MODEL                     REVISION  SERIAL               DEVICE
---------------------------------------------------------------------
ST9320423AS               0002SDM1  5VH5TBTC             sda
```

```
hp      DVDRAM GT30L      mP04      KZKA2LB3306         sr0
TOSHIBA TransMemory       1.00      7427EAB351F9CE110FE8E23E sdb
$ udisksctl info -b /dev/sdb
/org/freedesktop/UDisks2/block_devices/sdb:
  org.freedesktop.UDisks2.Block:
    Configuration:              []
    CryptoBackingDevice:        '/'
    Device:                     /dev/sdb
    DeviceNumber:               2064
    Drive:                      '/org/freedesktop/UDisks2/drives/▷
       ◁ TOSHIBA_TransMemory_7427EAB351F9CE110FE8E23E'
    HintAuto:                   true
    HintIconName:
    HintIgnore:                 false
    HintName:
    HintPartitionable:          true
```

Many of these data really derive from the `udev` database and may be influenced by the administrator by way of suitable rules.

In fairness we should also add that udisks represents a minimal solution. The common graphical desktop environments like KDE or GNOME include similar infrastructure, which also takes care of suitably representing storage devices in the graphical environment.

Commands in this Chapter

lsmod	Lists loaded kernel modules	lsmod(8)	71
lspci	Displays information about devices on the PCI bus	lspci(8)	65
lsusb	Lists all devices connected to the USB	lsusb(8)	68
modprobe	Loads kernel modules, taking dependencies into account	modprobe(8)	70
udevd	Kernel uevent management daemon	udevd(8)	73

Summary

- The firmware (BIOS/UEFI) organises the early startup of a Linux system, but will subsequently not be used (apart from exceptional cases).
- Typical BIOS settings relevant to Linux include, for example, the date and time, disk parameters, boot order, and the assignment of some system interfaces to actual peripherals.
- The `lspci` command sheds light on a system's PCI bus and the devices connected to it.

Bibliography

Hardware-HOWTO04 Steven Pritchard. "Linux Hardware Compatibility HOWTO", January 2004. `http://www.tldp.org/HOWTO/Hardware-HOWTO/`

Linux-USB-Subsystem Brad Hards. "The Linux USB sub-system". `http://www.linux-usb.org/USB-guide/book1.html`

PCI-HOWTO01 Michael Will. "Linux PCI-HOWTO", June 2001. `http://www.tldp.org/HOWTO/PCI-HOWTO.html`

6

Hard Disks (and Other Secondary Storage)

Contents

Goals

- Understanding how Linux deals with secondary storage devices based on ATA, SATA, and SCSI.
- Understanding MBR-based and GPT-based partitioning
- Knowing about Linux partitioning tools and how to use them
- Being able to use loop devices

Prerequisites

- Basic Linux knowledge
- Knowledge about Linux hardware support (see chapter 5)

6.1 Fundamentals

RAM is fairly cheap these days, but even so not many computers can get by without the permanent storage of programs and data on mass storage devices. These include (among others):

- Hard disks with rotating magnetic platters
- "Solid-state disks" (SSDs) that look like hard disks from the computer's point of view, but use flash memory internally
- USB thumb drives, SD cards, CF cards, and other interchangeable media based on flash memory
- RAID systems that aggregate hard disks and present them as one big storage medium
- SAN devices which provide "virtual" disk drives on the network
- File servers that offer file access at an abstract level (CIFS, NFS, …)

In this chapter we shall explain the basics of Linux support for the first three entries in the list—hard disks, SSDs and flash-based portable media like USB thumb drives. RAID sstems and SAN are discussed in the Linup Front training manual, *Linux Storage and File Systems*, file servers are discussed in *Linux Infrastructure Services*.

6.2 Bus Systems for Mass Storage

IDE

IDE, ATA and SATA Until not so long ago, hard disks and optical drives such as CD-ROM and DVD readers and writers used to be connected via a "IDE controller", of which self-respecting PCs had at least two (with two "channels" each).

"IDE" is really an abbreviation of "Integrated Drive Electronics". The "integrated drive electronics" alluded to here lets the computer see the disk as a sequence of numbered blocks without having to know anything about sectors, cylinders, and read/write heads, even though this elaborate charade is still kept up between the BIOS, disk controller, and disks. However, for a long time this description has applied to *all* hard disks, not just those with the well-known "IDE" interface, which by now is officially called "ATA", short for "AT Attachment"[1].

Serial ATA

Computers bought new these days usually still contain IDE interfaces, but the method of choice to connect hard disks and optical drives today is a serial version of the ATA standard, imaginatively named "Serial ATA" (SATA, for short). Since SATA (i. e., approximately 2003), traditional ATA (or "IDE") is commonly called "P-ATA", an abbreviation of "parallel ATA". The difference applies to the cable, which for traditional ATA is an inconvenient-to-place and electrically not 100% fortunate 40- or 80-lead ribbon cable which can transfer 16 data bits in parallel, and which links several devices at once to the controller. SATA, on the other hand, utilises narrow flat seven-lead cables for serial transmission, one per device (usually produced in a cheerful orange colour).

SATA cables and connectors are much more sensitive mechanically than the corresponding P-ATA hardware, but they make up for that through other advantages: They are less of an impediment to air flow in a PC case, cannot be installed wrongly due to their different connectors on each end, which furthermore cannot be plugged in the wrong way round. In addition, the illogical separation between 2.5-inch and 3.5-inch diskdrives, which required different connectors for P-ATA, goes away.

[1] Anyone remember the IBM PC/AT?

Interestingly, serial ATA allows considerably faster data transfers than traditional ATA, even though with the former all bits are transferred in "single file" rather than 16 at a go in parallel. This is due to the electrical properties of the interface, which uses differential transmission and a signalling voltage of only 0.5 V instead of 5 V. (This is why cables may be longer, too—1 m instead of formerly 45 cm.) Current SATA interfaces can theoretically transfer up to 16 GiBit/s (SATA 3.2) which due to encoding and other impediments comes out as approximately 2 MiB/s—rather more than single disk drives can keep up with at sustained rates, but useful for RAID systems that access multiple disk drives at the same time, and for fast SSDs. It is unlikely that SATA speeds will evolve further, since the trend with SSDs is towards connecting them directly via PCIe[2].

Besides the higher speed and more convenient cabling, SATA also offers the advantage of "hot-swapping": It is possible to disconnect a SATA disk drive and connect another one in its place, without having to shut down the computer. This of course presupposes that the computer can do without the data on the drive in question—typically because it is part of a RAID-1 or RAID-5, where the the data on the new drive can be reconstructed based on other drives in the system. With traditional ATA, this was impossible (or only possible by jumping through hoops).

eSATA

External SATA ("eSATA") is a derivative of SATA for use with external drives. It has different connectors and electrical specifications, which are much more robust mechanically and better suited for hot-swapping. In the meantime, it has been almost completely ousted from the market by USB 3.*x*, but can still be found in older hardware.

SCSI and SAS The "Small Computer System Interface" or SCSI (customary pronunciation: "SCUZ-zy") has served for more than 25 years to connect hard disks, tape drives and other mass storage devices, but also peripherals such as scanners, to "small" computers[3]. SCSI buses and connectors exist in a confusing variety, beginning with the "traditional" 8-bit bus and ranging from the fast 16-bit varieties to new, even faster serial implementations (see below). They also differ in the maximum number of devices per bus, and in physical parameters such as the maximum cable length and the allowable distances between devices on the cable. Nicely enough, most of the variants are compatible or can be made compatible (possibly with loss of efficiency!). Varieties such as *FireWire* (IEEE-1394) or *FiberChannel* are treated like SCSI by Linux.

Nowadays, most work goes into the serial SCSI implementations, most notably "Serial Attached SCSI" (SAS). As with SATA, data transfer is potentially faster (at the moment, SAS is slightly slower than the fastest parallel SCSI version, Ultra-640 SCSI) and electrically much less intricate. In particular, the fast parallel SCSI versions are plagued by clocking problems that derive from the electrical properties of the cables and termination, and that do not exist with SAS (where the pesky termination is no longer necessary at all).

SAS and SATA are fairly closely related; the most notable differences are that SAS allows things like accessing a drive via several cable paths for redundancy ("multipath I/O"; SATA requires jumping through hoops for this), supports more extensive diagnosis and logging functions, and is based on a higher signalling voltage, which allows for longer cables (up to 8 m) and physically larger servers.

[2] SATA in a strict sense allows speeds up to 6 GiBit/s; the higher speed of SATA 3.2 is already achieved by means of PCIe. This "SATA Express" specification defines an interface that can carry SATA signals as well as PCIe, such that compatible devices can be connected not only to SATA Express controllers, but also to older hosts which support "only" SATA with up to 6 GiBit/s.

[3] Nobody has ever defined the meaning of "small" in this context, but it must be something like "can be bodily lifted by at most two people"

Table 6.1: Different SCSI variants

Name	Width	Transfer rate	Devices	Explanation
SCSI-1	8 bit	⩽ 5 MiB/s	8	“Ancestor”
SCSI-2 »Fast«	8 bit	10 MiB/s	8	
SCSI-2 »Wide«	16 bit	20 MiB/s	16	
SCSI-3 »Ultra«	8 bit	20 MiB/s	8	
SCSI-3 »Ultrawide«	16 bit	40 MiB/s	16	
Ultra2 SCSI	16 bit	80 MiB/s	16	LVD bus[a]
Ultra-160 SCSI[b]	16 bit	160 MiB/s	16	LVD bus
Ultra-320 SCSI[c]	16 Bit	320 MiB/s	16	LVD bus
Ultra-640 SCSI	16 Bit	640 MiB/s	16	LVD bus

SATA and SAS are compatible to an extent where you can use SATA disk drives on a SAS backplane (but not vice-versa).

Vorkommen

“Pure-bred” SCSI, as far as PCs are concerned, is found mostly in servers; work stations and “consumer PCs” tend to use IDE or SATA for mass storage and USB (qv. section 5.2.1) for other devices. Devices based on IDE and USB are much cheaper to manufacture than SCSI-based devices—IDE disks, for example, cost about a third or a fourth of the price of comparatively large SCSI disks.

We do need to mention that SCSI disks are usually designed especially for use in servers, and are therefore optimised for high speed and longevity. SATA disks for workplace PCs do not come with the same warranties, are not supposed to rival a jet fighter for noise, and should support fairly frequent starting and stopping.

As a Linux administrator, you should know about SCSI even if you do not run any SCSI-based systems, since from the point of view of the Linux kernel, in addition to SATA many USB or FireWire devices are accessed like SCSI devices and use the same infrastrucure.

SCSI ID

Every device on a SCSI bus requires a unique “SCSI ID”. This number between 0 and 7 (15 on the wide buses) is used to address the device. Most “true” SCSI devices sport jumpers or a switch to select it; with Fibre-Channel, USB, or SATA devices that are accessed via the SCSI infrastructure, the system arranges for suitable unique SCSI IDs to be assigned.

host adapter
SCSI BIOS

To use SCSI, a PC needs at least one host adapter (or “host”). Motherboard-based and better expansion card host adapters contain a SCSI BIOS which lets the system boot from a SCSI device. You can also use this to check which SCSI IDs are available and which are used, and which SCSI device, if any, should be used for booting.

The host adapter counts as a device on the SCSI bus—apart from itself you can connect 7 (or 15) other devices.

boot order

If your BIOS can boot from SCSI devices, you can also select in the boot order whether the ATA disk `C:` should be preferred to any (potentially) bootable SCSI devices.

termination

Most important for the correct function of a parallel SCSI system is appropriate **termination** of the SCSI bus. This can either be ensured via a special plug (“terminator”) or switched on or off on individual devices. Erroneous termination is the possible origin of all sorts of SCSI problems. If you do experience difficulties with SCSI, always check first that termination is in order. SAS does not require termination.

USB With the new fast USB variants (Section 5.2.1), few if any compromises will be needed when connecting mass storage devices—reading and writing speeds are bounded by the storage device, not (as with USB 1.1 and USB 2.0) by the bus. Linux manages USB-based storage devices exactly like SCSI devices.

Exercises

6.1 [1] How many hard disks or SSDs does your computer contain? What is their capacity? How are they connected to the computer (SATA, …)?

6.3 Partitioning

6.3.1 Fundamentals

Mass storage devices such as hard disks or SSDs are commonly "partitioned", i. e., subdivided into several logical storage devices that the operating system can then access independently. This does not only make it easier to use data structures that are appropriate to the intended use—sometimes partitioning is the only way to make a very large storage medium fully accessible, if limits within the operating system preclude the use of the medium "as a whole" (even though this sort of problem tends to be rare today).

Advantages of partitioning include the following:

- Logically separate parts of the system may be separated. For example, you could put your users' data on a different partition from that used by the operating system itself. This makes it possible to reinstall the operating system from scratch without endangering your users' data. Given the often rudimentary "upgrade" functionality of even current distributions this is very important. Furthermore, if inconsistencies occur in a file system then only one partition may be impacted at first.

- The structure of the file system may be adapted to the data to be stored. Most file systems keep track of data by means of fixed-size "blocks", where every file, no matter how small, occupies at least a single block. With a 4 KiB block size this implies that a 500-byte file only occupies 1/8 of its block—the rest goes to waste. If you know that a directory will contain mostly small files (cue: mail server), it may make sense to put this directory on a partition using smaller blocks (1 or 2 KiB). This can reduce waste considerably. Some database servers, on the other hand, like to work on "raw" partitions (without any file system) that they manage themselves. An operating system must make that possible, too.

- "Runaway" processes or incautious users can use up all the space available on a file system. At least on important server systems it makes sense to allow user data (including print jobs, unread e-mail, etc.) only on partitions that may get filled up without getting the system itself in trouble, e.g., by making it impossible to append information to important log files.

There are currently two competing methods to partition hard disks for PCs. The traditional method goes back to the 1980s when the first hard disks (with awesome sizes like 5 or 10 MB) appeared. Recently a more modern method was introduced; this does away with various limitations of the traditional approach, but in some cases requires special tools.

Hard disks are virtually always partitioned, even though at times only one partition will be created. With USB thumb drives, one sometimes eschews partitioning altogether.

Table 6.2: Partition types for Linux (hexadecimal)

Type	Description
81	Linux data
82	Linux swap space
86	RAID super block (old style)
8E	Linux LVM
E8	LUKS (encrypted partition)
EE	"Protective partition" for GPT-partitioned disk
FD	RAID super block with autodetection
FE	Linux LVM (old style)

6.3.2 The Traditional Method (MBR)

primary partitions
extended partition
logical partitions

The traditional method stores partitioning information inside the "master boot record" (MBR), the first sector (number 0) of a hard disk. (Traditionally, PC hard disk sectors are 512 bytes long, but see below.) The space there—64 bytes starting at offset 446—is sufficient for four **primary partitions**. If you want to create more than four partitions, you must use one of these primary partitions as an **extended partition**. An extended partition may contain further **logical partitions**.

The details about logical partitions are not stored inside the MBR, but at the start of the partition (extended or logical) in question, i. e., they are scattered around the hard disk.

Partition entries today usually store the starting sector number of the partition on the disk as well as the length of the partition in question in sectors[4]. Since these values are 32-bit numbers, given the common 512-byte sectors this results in a maximum partition size of 2 TiB.

There are hard disks available today which are larger than 2 TiB. Such disks cannot be made fully accessible using MBR partitioning. One common ruse consists in using disks whose sectors are 4096 bytes long instead of 512. This will let you have 16-TiB disks even with MBR, but not every operating system supports such "4Kn" drives (Linux from kernel 2.6.31, Windows from 8.1 or Server 2012).

4-KiB sectors are useful on hard disks even without considering partitions. The larger sectors are more efficient for storing larger files and allow better error correction. Therefore the market offers "512e" disks which use 4-KiB sectors internally but pretend to the outside that they really have 512-byte sectors. This means that if a single 512-byte sector needs to be rewritten, the adjoining 7 sectors must be read and also rewritten (a certain, but usually bearable, loss of efficiency, since data is most often written in larger chunks). When partitioning, you will have to pay attention that the 4-KiB blocks that Linux uses internally for hard disk access coincide with the disk's internal 4-KiB sectors—if that is not the case, then to write one 4-KiB Linux block *two* 4-KiB disk sectors might have to be read *and* rewritten, and that would not be good. (Fortunately, the partitioning tools help you watch out for this.)

partition type

Besides the starting address and length of the (primary) partitions, the partition table contains a partition type which loosely describe the type of data management structure that might appear on the partition. A selection of Linux partition types appears in table 6.2.

[4] In former times, partitions used to be described in terms of the cylinder, head, and sector addresses of the sectors in question, but this has been deprecated for a very long time.

6.3.3 The Modern Method (GPT)

In the late 1990s, Intel developed a new partitioning method that should do away with the limitations of the MBR approach, namely "GUID Partition Table" or GPT.

GPT was developed hand-in-hand with UEFI and is part of the UEFI specification today. You can, however, use a BIOS-based Linux system to access GPT-partitioned disks and vice-versa.

GPT uses 64-bit sector addresses and thus allows a maximum disk size of 8 ZiB—zebibyte, in case you haven't run into that prefix. 1 ZiB are 2^{70} bytes, or, roughly speaking, about one million million tebibytes. This should last even the NSA for a while. (Disk manufactures, who prefer to talk powers of ten rather than powers of two, will naturally sell you an 8-ZiB disk as a 9.4 zettabyte disk.)

With GPT, the first sector of the disk remains reserved as a "protective MBR" which designates the whole disk as partitioned from a MBR point of view. This avoids problems if a GPT-partitioned disk is connected to a computer that can't handle GPT.

The second sector (address 1) contains the "GPT header" which stores management information for the whole disk. Partitioning information is usually contained in the third and subsequent sectors.

The GPT header points to the partitioning information, and therefore they could be stored anywhere on the disk. It is, however, reasonable to place them immediately after the GPT header. The UEFI header stipulates a minimum of 16 KiB for partitioning information (regardless of the disk's sector size).

On a disk with 512-byte sectors, with a 16 KiB space for partitioning information the first otherwise usable sector on the disk is the one at address 34. You should, however, avoid placing the disk's first partition at that address because that will get you in trouble with 512e disks. The next correctly-aligned sector is the one at address 40.

For safety reasons, GPT replicates the partitioning information at the end of the disk.

Traditionally, partition boundaries are placed at the start of a new "track" on the disk. Tracks, of course, are a relic from the hard disk paleolithic, since contemporary disks are addressed linearly (in other words, the sectors are numbered consecutively from the start of the disk to the end)—but the idea of describing a disk by means of a combination of a number of read/write heads, a number of "cylinders", and a number of sectors per "track" (a track is the concentric circle a single head describes on a given cylinder) has continued to be used for a remarkably long time. Since the maximum number of sectors per track is 63, this means that the first partition would start at block 63, and that is, of course, disastrous for 512e disks.

Since Windows Vista it is common to have the first partition start 1 MiB after the start of the disk (with 512-byte sectors, at sector 2048). This isn't a bad idea for Linux, either, since the ample free space between the partition table and the first partition can be used to store the GRUB boot loader. (The space between the MBR and sector 63 was quite sufficient earlier, too.)

Partition table entries are at least 128 bytes long and, apart from 16-byte GUIDs for the partition type and the partition itself and 8 bytes each for a starting and ending block number, contains 8 bytes for "attributes" and 72 bytes for a partition name. It is debatable whether 16-byte GUIDs are required for partition types, but on the one hand the scheme is called "GUID partition table" after all, and on the other hand this ensures that we won't run out of partition types anytime soon. A selection is displayed in table 6.3.

GUID	Description
00000000-0000-0000-0000-000000000000	Unused entry
C12A7328-F81F-11D2-BA4B-00A0C93EC93B	EFI system partition (ESP)
21686148-6449-6E6F-744E-656564454649	BIOS boot partition
0FC63DAF-8483-4772-8E79-3D69D8477DE4	Linux file system
A19D880F-05FC-4D3B-A006-743F0F84911E	Linux RAID partition
0657FD6D-A4AB-43C4-84E5-0933C84B4F4F	Linux swap space
E6D6D379-F507-44C2-A23C-238F2A3DF928	Linux LVM
933AC7E1-2EB4-4F13-B844-0E14E2AEF915	/home partition
3B8F8425-20E0-4F3B-907F-1A25A76F98E8	/srv partition
7FFEC5C9-2D00-49B7-8941-3EA10A5586B7	dm-crypt partition
CA7D7CCB-63ED-4C53-861C-1742536059CC	LUKS partition

Table 6.3: Partition type GUIDs for GPT (excerpt)

Linux can use GPT-partitioned media. This needs the “EFI GUID Partition support” option enabled in the kernel, but with current distributions this is the case. Whether the installation procedure allows you to create GPT-partitioned disks is a different question, just like the question of whether the boot loader will be able to deal with them. But that is neither here nor there.

6.4 Linux and Mass Storage

If a mass storage device is connected to a Linux computer, the Linux kernel tries to locate any partitions. It then creates block-oriented device files in /dev for the device itself and its partitions. You can subsequently access the partitions' device files and make the directory hierarchies there available as part of the computer's file system.

A new mass storage device may have no partitions at all. In this case you can use suitable tools to create partitions. This will be explained later in this chapter. The next step after partitioning consists of generating file systems on the partitions. This is explained in detail in chapter 7.

The device names for mass storage are usually /dev/sda, /dev/sdb, …, in the order the devices are recognised. Partitions are numbered, the /dev/sda device therefore contains partitions like /dev7sda1, /dev/sda2, … A maximum of 15 partitions per device is possible. If /dev/sda is partitioned according to the MBR scheme, /dev/sda1 to /dev/sda4 correspond to the primary partitions (possibly including an extended partition), while any logical partitions are numbered starting with /dev/sda5 (regardless of whether there are four primary partitions on the disk or fewer).

The “s” in /dev/sda derives from “SCSI”. Today, almost all mass storage devices in Linux are managed as SCSI devices.

For P-ATA disks there is another, more specific mechanism. This accesses the IDE controllers inside the computer directly—the two drives connected to the first controller are called /dev/hda and /dev/hdb, the ones connected to the second /dev/hdc and /dev/hdd. (These names are used independently of whether the drives actually exist or not—if you have a single hard disk and a CD-ROM drive on your system, you do well to connect the one to one controller and the other to the other so they will not be in each other's way. Therefore you will have /dev/hda for the disk and /dev/hdc for the CD-ROM drive.) Partitions on P-ATA disks are, again, called /dev/hda1, /dev/hda2 and so on. In this scheme, 63 (!) partitions are allowed.

If you still use a computer with P-ATA disks, you will notice that in the vast majority of cases the SCSI infrastructure is used for those, too (note the `/dev/sda` style device names). This is useful for convenience and consistency. Some very few P-ATA controllers are not supported by the SCSI infrastructure, and must use the old P-ATA specific infrastructure.

The migration of an existing Linux system from "traditional" P-ATA drivers to the SCSI infrastructure should be well-considered and involve changing the configuration in `/etc/fstab` such that file systems are not mounted via their device files but via volume labels or UUIDs that are independent of the partitions' device file names. (See section 7.2.3.)

architecture

The Linux kernel's mass storage subsystem uses a three-tier architecture. At the bottom there are drivers for the individual SCSI host adapters, SATA or USB controllers and so on, then there is a generic "middle layer", on top of which there are drivers for the various devices (disks, tape drives, …) that you might encounter on a SCSI bus. This includes a "generic" driver which is used to access devices without a specialised driver such as scanners or CD-ROM recorders. (If you can still find any of those anywhere.)

LUNs

Every SCSI host adapter supports one or more buses ("channels"). Up to 7 (or 15) other devices can be connected to each bus, and every device can support several "local unit numbers" (or LUNs), such as the individual CDs in a CD-ROM changer (rarely used). Every SCSI device in the system can thus be describe by a quadrupel (⟨*host*⟩, ⟨*channel*⟩, ⟨*ID*⟩, ⟨*LUN*⟩). Usually (⟨*host*⟩, ⟨*channel*⟩, ⟨*ID*⟩) are sufficient.

In former times you could find information on SCSI devices within the `/proc/scsi/scsi` directory. This is no longer available on current systems unless the kernel was compiled using "Legacy `/proc/scsi` support".

Nowadays, information about "SCSI controllers" is available in `/sys/class/scsi_host` (one directory per controller). This is unfortunately not quite as accessible as it used to be. You can still snoop around:

```
# cd /sys/class/scsi_host/host0/device
# ls
power  scsi_host  subsystem  target0:0:0  uevent
# cd target0:0:0; ls
0:0:0:0  power  subsystem  uevent
# ls 0:0:0:0/block
sda
```

A peek into `/sys/bus/scsi/devices` will also be instructive:

```
# ls /sys/bus/scsi/devices
0:0:0:0  10:0:0:0  host1   host2  host4  target0:0:0  target10:0:0
1:0:0:0  host0     host10  host3  host5  target1:0:0
```

Device names such as `/dev/sda`, `/dev/sdb`, etc. have the disadvantage of not being very illuminating. In addition, they are assigned to devices in the order of their appearance. So if today you connect first your MP3 player and then your digital camera, they might be assigned the device files `/dev/sdb` and `/dev/sdc`; if tomorrow you start with the digital camera and continue with the MP3 player, the names might be the other way round. This is of course a nuisance. Fortunately, `udev` assigns some symbolic names on top of the traditional device names. These can be found in `/dev/block`:

```
# ls -l /dev/block/8:0
lrwxrwxrwx 1 root root 6 Jul 12 14:02 /dev/block/8:0 -> ../sda
# ls -l /dev/block/8:1
lrwxrwxrwx 1 root root 6 Jul 12 14:02 /dev/block/8:1 -> ../sda1
# ls -l /dev/disk/by-id/ata-ST9320423AS_5VH5TBTC
lrwxrwxrwx 1 root root 6 Jul 12 14:02 /dev/disk/by-id/▷
 ◁ ata-ST9320423AS_5VH5TBTC -> ../../sda
# ls -l /dev/disk/by-id/ata-ST9320423AS_5VH5TBTC-part1
lrwxrwxrwx 1 root root 6 Jul 12 14:02 /dev/disk/by-id/▷
 ◁ ata-ST9320423AS_5VH5TBTC-part1 -> ../../sda1
# ls -l /dev/disk/by-path/pci-0000:00:1d.0-usb-▷
 ◁ 0:1.4:1.0-scsi-0:0:0:0
lrwxrwxrwx 1 root root 6 Jul 12 14:02 /dev/disk/by-path/▷
 ◁ pci-0000:00:1d.0-usb-0:1.4:1.0-scsi-0:0:0:0  -> ../../sdb
# ls -l /dev/disk/by-uuid/c59fbbac-9838-4f3c-830d-b47498d1cd77
lrwxrwxrwx 1 root root 10 Jul 12 14:02 /dev/disk/by-uuid/▷
 ◁ c59fbbac-9838-4f3c-830d-b47498d1cd77 -> ../../sda1
# ls -l /dev/disk/by-label/root
lrwxrwxrwx 1 root root 10 Jul 12 14:02 /dev/disk/by-label/root ◁
▷ -> ../../sda1
```

These device names are derived from data such as the (unique) serial number of the disk drive, its position on the PCIe bus, or the UUID or name of the file system, and are independent of the name of the actual device file.

Exercises

6.2 [!2] On your ssytem there are two SATA hard disks. The first disk has two primary and two logical partitions. The second disk has one primary and three logical partitions. Which are the device names for these partitions on Linux?

6.3 [!1] Examine the /dev directory on your system. Which storage media are available and what are the corresponding device files called? (Also check /dev/block and /dev/disk.)

6.4 [1] Plug a USB thumb drive into your computer. Check whether new device files have been added to /dev. If so, which ones?

6.5 Partitioning Disks

6.5.1 Fundamentals

Before you partition the (possibly sole) disk on a Linux system, you should briefly consider what a suitable partitioning scheme might look like and how big the partitions ought to be. Changes after the fact are tedious and inconvenient at best and may at worst necessitate a complete re-install of the system (which would be exceedingly tedious and inconvenient). (See section 6.7 for an alternative, much less painful approach.)

Here are a few basic suggestions for partitioning:

- Apart from the partition with the root directory /, you should provide at least one spearate partition for the file system containing the /home directory. This lets you cleanly separate the operating system from your own data, and facilitates distribution upgrades or even switching from one Linux distribution to a completely different one.

If you follow this approach, you should probably also use symbolic links to move the `/usr/local` and `/opt` directories to (for example) `/home/usr-local` and `/home/opt`. This way, these directories, which also contain data provided by you, are on "your" partition and can more easily be included in regular backups.

- It is absolutely possible to fit a basic Linux system into a 2 GiB partition, but, considering today's (low) costs per gigabyte for hard disk storage, there is little point in scrimping and saving. With something like 30 GiB, you're sure to be on the safe side and will have enough room for log files, downloaded distribution packages during a larger update, and so on.

- On server systems, it may make sense to provide separate partitions for `/tmp`, `/var`, and possibly `/srv`. The general idea is that arbitrary users can put data into these directories (besides outright files, this could include unread or unsent e-mail, queued print jobs, and so on). If these directories are on separate partitions, users cannot fill up the system in general and thereby create problems.

- You should provide swap space of approximately the same size as the computer's RAM, up to a maximum of 8 GiB or thereabouts. Much more is pointless, but on workstations and mobile computers you may want to avail yourself of the possibility to "suspend" your computer instead of shutting it down, in order to speed up a restart and end up exactly where you were before—and the infrastructures enabling this like to use the swap space to save the RAM content.

  There used to be a rule of thumb saying that the swap space should be about twice or three times the available RAM size. This rule of thumb comes from traditional Unix systems, where RAM works as "cache" for the swap space. Linux doesn't work that way, instead RAM and swap space are added—on a computer with 4 GiB of RAM and 2 GiB of swap space, you get to run processes to the tune of 6 GiB or so. With 8 GiB of RAM, providing 16 to 24 GiB of swap space would be absurd.

  You should dimension the RAM of a computer (especially a server) to be big enough that practically no swap space is necessary during normal operations.; on an 8-GiB server, you won't usually need 16 GiB of swap space, but a gigabyte or two to be on the safe side will certainly not hurt (especially considering today's prices for disk storage). That way, if RAM gets tight, the computer will slow down before processes crash outright because they cannot get memory from the operating system.

- If you have several (physical) hard disks, it can be useful to spread the system across the available disks in order to increase the access speeed to individual components.

  Traditionally, one would place the root file system (`/` with the essential subdirectories `/bin`, `/lib`, `/etc`, and so on) on one disk and the `/usr` directory with its subdirectories on a separate file system on another disk. However, the trend on Linux is decisively away from the (artificial) separation between `/bin` and `/usr/bin` or `/lib` and `/usr/lib` and towards a root file system which is created as a RAM disk on boot. Whether the traditional separation of `/` and `/usr` will gain us a lot in the future is up for debate.

  What will certainly pay off is to spread swap space across several disks. Linux always uses the least-used disk for swapping.

Provided that there is enough empty space on the medium, new partitions can be created and included (even while the system is running). This procedure consists of the following steps:

1. Backup the current boot sectors and data on the hard disk in question
2. Partition the disk using `fdisk` (or a similar program)
3. Possibly create file systems on the new partitions ("formatting")
4. Making the new file systems accessible using `mount` or `/etc/fstab`

Items 3 and 4 on this list will be considered in more detail in chapter 7.

Data and boot-sector contents can be saved using the `dd` program (among others).

```
# dd if=/dev/sda of=/dev/st0
```

will, for example, save all of the `sda` hard disk to magnetic tape.

You should be aware that the partitioning of a storage medium has nothing to do with the data stored on it. The partition table simply specifies where on the disk the Linux kernel can find the partitions and hence the file structures. Once the Linux kernel has located a partition, the content of the partition table is irrelevant until it searches again, possibly during the next system boot. This gives you—if you are courageous (or foolhardy)—far-reaching opportunities to tweak your system while it is running. For example, you can by all means enlarge partitions (if after the end of the partition there is unused space or a partition whose contents you can do without) or make them smaller (and possibly place another partition in the space thus freed up). As long as you exercise appropriate care this will be reasonably safe.

This should of course in no way discourage you from making appropriate backup copies before doing this kind of open-heart surgery.

In addition, the file systems on the disks must play along with such shenanigans (many Linux file systems can be enlarged without loss of data, and some of them even be made smaller), or you must be prepared to move the data out of the way, generate a new file system, and then fetch the data back.

6.5.2 Partitioning Disks Using `fdisk`

`fdisk` `fdisk` is an interactive program for manipulating disk partition tables. It can also create "foreign" partition types such as DOS partitions. Drives are addressed using the corresponding device files (such as `/dev/sda` for the first disk).

`fdisk` confines itself to entering a partition into the partition table and setting the correct partition type. If you create a DOS or NTFS partition using `fdisk`, this means just that the partition exists in the partition table, not that you can boot DOS or Windows NT now and write files to that partition. Before doing that, you must create a file system, i. e., write the appropriate management data structures to the disk. Using Linux-based tools you can do this for many but not all non-Linux file systems.

After invoking "`fdisk` ⟨*device*⟩", the program comes back with a succinct prompt of

```
# fdisk /dev/sdb                                              Neue (leere) Platte
Welcome to fdisk (util-linux 2.25.2).
Changes will remain in memory only, until you decide to write them.
Be careful before using the write command.

Device does not contain a recognized partition table.
```

```
Created a new DOS disklabel with disk identifier 0x68d97339.

Command (m for help): _
```

The m command will display a list of the available commands.

fdisk lets you partition hard disks according to the MBR or GPT schemes. It recognises an existing partition table and adjusts itself accordingly. On an empty (unpartitioned) disk fdisk will by default create an MBR partition table, but you can change this afterwards (we'll show you how in a little while).

You can create a new partition using the "n" command:

```
Command (m for help): n
Partition type
   p   primary (0 primary, 0 extended, 4 free)
   e   extended (container for logical partitions)
Select (default p): p
Partition number (1-4, default 1): 1
First sector (2048-2097151, default 2048): ↩
Last sector, +sectors or +sizeK,M,G,T,P (2048-2097151,▷
 ◁ default 2097151): +500M

Created a new partition 1 of type 'Linux' and of size 500 MiB.

Command (m for help): _
```

The p command displays the current partition table. This could look like this:

```
Command (m for help): p
Disk /dev/sdb: 1 GiB, 1073741824 bytes, 2097152 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0x68d97339

Device     Boot Start     End Sectors  Size Id Type
/dev/sdb1        2048 1026047 1024000  500M 83 Linux
```

You can change the partition type using the t command. You must select the desired partition and can then enter the code (as a hexadecimal number). L displays the whole list.

You can delete a partition you no longer want by means of the d command. When you're done, you can write the partition table to disk and quit the program using w. With q, you can quit the program without rewriting the partition table.

After storing the partition table, fdisk tries to get the Linux kernel to reread the new partition table; this works well with new or unused disks, but fails if a partition on the disk is currently in use (as a mounted file system, active swap space, …). This lets you repartition the disk with the / file system only with a system reboot. One of the rare occasions when a Linux system needs to be rebooted …

Options

Like all Linux commands, fdisk supports a number of command-line options. The most important of those include:

-l displays the partition table of the selected disk and then terminates the program.

-u ("units") lets you select the units of measure used when displaying partition tables. The default is "sectors"; if you specify "`-u=cylinders`", cylinders will be used instead (but there is no good reason for that today).

If you use `fdisk` in MBR mode, it tries to observe the usual conventions and arrange the partitions such that they work properly on 4Kn and 512e hard disks. You should follow the program's suggestions wherever possible, and not deviate from them unless there are very compelling reasons.

If you partition a hard disk according to the GPT standard and there is no GPT-style partition table on the disk yet, you can generate one using the `g` command (*Warning:* A possibly existing MBR partition table will be overwritten in the process):

```
Command (m for help): g
Created a new GPT disklabel (GUID: C2A556FD-7C39-474A-B383-963E09AA7269)
```

(The GUID shown here applies to the disk as a whole.) Afterwards you can use the `n` command to create partitions in the usual way, even if the dialog looks slightly different:

```
Command (m for help): n
Partition number (1-128, default 1): 1
First sector (2048-2097118, default 2048): ↩
Last sector, +sectors or +sizeK,M,G,T,P (2048-2097118, default▷
 ◁ 2097118): +32M

Created a new partition 1 of type 'Linux filesystem' and of size 32 MiB.
```

The partition type selection is different, too, because it is about GUIDs rather than two-digit hexadecimal numbers:

```
Command (m for help): t
Selected partition 1
Partition type (type L to list all types): L
  1 EFI System                    C12A7328-F81F-11D2-BA4B-00A0C93EC93B
◁◁◁◁◁
 14 Linux swap                    0657FD6D-A4AB-43C4-84E5-0933C84B4F4F
 15 Linux filesystem              0FC63DAF-8483-4772-8E79-3D69D8477DE4
 16 Linux server data             3B8F8425-20E0-4F3B-907F-1A25A76F98E8
 17 Linux root (x86)              44479540-F297-41B2-9AF7-D131D5F0458A
 18 Linux root (x86-64)           4F68BCE3-E8CD-4DB1-96E7-FBCAF984B709
◁◁◁◁◁
Partition type (type L to list all types): _
```

Exercises

6.5 [!2] Create an empty 1 GiB file using the

```
# dd if=/dev/zero of=$HOME/test.img bs=1M count=1024
```

command. Use `fdisk` to "partition" the file according to the MBR scheme: Create two Linux partitions of 256 MiB and 512 MiB, respectively, and create a swap space partitions using the balance.

6.6 [!2] Repeat the following exercise, but create a GPT partition table instead. Assume that the 512-MiB partition will contain a `/home` directory.

6.5.3 Formatting Disks using GNU parted

Another popular program for partitioning storage media is the GNU project's parted. Featurewise, it is roughly comparable with fdisk, but it has a few useful features.

Unlike fdisk, parted does not come pre-installed with most distributions, but can generally be added after the fact from the distribution's servers.

Similar to fdisk, parted must be started with the name of the medium to be partitioned as a parameter.

```
# parted /dev/sdb
GNU Parted 3.2
Using /dev/sdb
Welcome to GNU Parted! Type 'help' to view a list of commands.
(parted) _
```

You can create a new partition using mkpart. This works either interactively …

```
(parted) mkpart
Partition name?  []? Test
File system type?  [ext2]? ext4
Start? 211MB
End? 316MB
```

… or directly when the command is invoked:

```
(parted) mkpart primary ext4 211MB 316MB
```

You can abbreviate the commands down to an unambiguous prefix. Hence, mkp will work instead of mkpart (mk would collide with the mklabel command).

The file system type will only be used to guess a partition type. You will still need to manually create a file system on the partition later on.

You can examine the partition table using the print command:

```
(parted) p
Model: ATA VBOX HARDDISK (scsi)
Disk /dev/sdb: 1074MB
Sector size (logical/physical): 512B/512B
Partition Table: gpt
Disk Flags:

Number  Start   End    Size   File system  Name     Flags
 1      1049kB  106MB  105MB
 2      106MB   211MB  105MB
 3      211MB   316MB  105MB  ext4         primary

(parted) _
```

(This also shows you where the magic numbers "211MB" and "316MB" came from, earlier on.)

print has a few interesting subcommands: "print devices" lists all available block devices, "print free" displays free (unallocated) space, and "print all" outputs the partition tables of all block devices.

You can get rid of unwanted partitions using rm. Use name to give a name to a partition (only for GPT). The quit command stops the program.

Important: While `fdisk` updates the partition table on the disk only once you leave the program, `parted` does it on an ongoing basis. This means that the addition or removal of a partition takes effect on the disk immediately.

If you use `parted` on a new (unpartitioned) disk, you must first create a partition table.

```
(parted) mklabel gpt
```

creates a GPT-style partition table, and

```
(parted) mklabel msdos
```

one according to the MBR standard. There is no default value; without a partition table, `parted` will refuse to execute the `mkpart` command.

If you inadvertently delete a partition that you would rather have kept, `parted` can help you find it again. You will just need to tell it approximately where on the disk the partition used to be:

```
(parted) rm 3                                                     Oops.
(parted) rescue 200MB 350MB
Information: A ext4 primary partition was found at 211MB -> 316MB.
Do you want to add it to the partition table?

Yes/No/Cancel? yes
```

For this to work, there must be a file system on the partition, because `parted` looks for data blocks that appear to be the start of a file system.

In addition to the interactive mode, `parted` allows you to pass commands immediately on the Linux command line, like this:

```
# parted /dev/sdb mkpart primary ext4 316MB 421MB
Information: You may need to update /etc/fstab.
```

Exercises

6.7 [!2] Repeat exercise 6.5 using `parted` rather than `fdisk`, for the MBR as well as the GPT scheme.

6.8 [2] (If you have worked through chapter 7 already.) Generate Linux file systems on the partitions on the “disk” from the earlier exercises. Remove these partitions. Can you restore them using `parted`’s `rescue` command?

6.5.4 gdisk

The `gdisk` program specialises in GPT-partitioned disks and can do a few useful things the previously explained programs can’t. You may however have to install it specially.

The elementary functions of `gdisk` correspond to those of `fdisk` and `parted`, and we will not reiterate those (read the documentation and do a few experiments). A few features of `gdisk`, however, are of independent interest:

- You can use `gdisk` to convert an MBR-partitioned medium to a GPT-partitioned medium. This presupposes that there is enough space at the start and the end of the medium for GPT partition tables. With media where, according to current conventions, the first partition starts at sector 2048, the former is not a problem, but the latter may be. You will possibly have to ensure that the last 33 sectors on the medium are not assigned to a partition.

For the conversion it is usually sufficient to start gdisk with the device file name of the medium in question as a parameter. You will either receive a warning that no GPT partition table was found and disk used the MPT partition table instead (at this point you can quit the program using w and you're done), or that an intact MBR, but a damaged GPT partition table was found (then you tell gdisk to follow the MBR, and can then quit the program using w and you're done).

- The other direction is also possible. To do this, you must use the r command in gdisk to change to the menu for "recovery/transformation commands", and select the g command there ("convert GPT into MBR and exit"). Afterwards you can quit the program using w and convert the storage medium this way.

Exercises

6.9 [!2] Repeat exercise 6.5 using gdisk rather than fdisk and generate a GPT partition table.

6.10 [2] Create (e. g., using fdisk) an MBR-partitioned "disk" and use gdisk to convert it to GPT partitioning. Make sure that a correct "protective MBR" was created.

6.5.5 More Partitioning Tools

distributions

Most distributions come with alternative ways of partitioning disks. Most of them offer the cfdisk program as an alternative to fdisk. This is screen-oriented and thus somewhat more convenient to use. Even easier to use are graphical programs, such as SUSE's YaST or "DiskDruid" on Red Hat distributions.

Also worth mentioning is sfdisk, a completely non-interactive partitioning program. sfdisk takes partitioning information from an input file and is therefore useful for unattended partitioning, e. g., as part of an automated installation procedure. Besides, you can use sfdisk to create a backup copy of your partitioning information and either print it as a table or else store it on a disk or CD as a file. If the worst happens, this copy can then be restored using sfdisk.

sfdisk only works for MBR-partitioned disks. There is a corresponding program called sgdisk which does an equivalent job for GPT-partitioned disks. However, sfdisk and sgdisk are not compatible—their option structures are completely different.

6.6 Loop Devices and kpartx

Linux has the useful property of being able to treat files like storage media. This means that if you have a file you can partition it, generate file systems, and generally treat the "partitions" on that file as if they were partitions on a "real" hard disk. In real life, this can be useful if you want to access CD-ROMs or DVDs without having a suitable drive in your computer (it is also faster). For learning purposes, it means that you can perform various experiments without having to obtain extra hard disks or mess with your computer.

CD-ROM image

A CD-ROM image can be created straightforwardly from an existing CD-ROM using dd:

```
# dd if=/dev/cdrom of=cdrom.iso bs=1M
```

You can subsequently make the image directly accessible:

```
# mount -o loop,ro cdrom.iso /mnt
```

In this example, the content of the CD-ROM will appear within the /mnt directory.

You can also use the dd command to create an empty file:

```
# dd if=/dev/zero of=disk.img bs=1M count=1024
```

That file can then be "partitioned" using one of the common partitioning tools:

```
# fdisk disk.img
```

Before you can do anything with the result, you will have to ensure that there are device files available for the partitions (unlike with "real" storage media, this is not automatic for simulated storage media in files). To do so, you will first need a device file for the file as a whole. This—a so-called "loop device"—can be created using the losetup command:

```
# losetup -f disk.img
# losetup -a
/dev/loop0: [2050]:93 (/tmp/disk.img)
```

losetup uses device file names of the form "/dev/loop*n*". The "-f" option makes the program search for the first free name. "losetup -a" outputs a list of the currently active loop devices.

Once you have assigned your disk image to a loop device, you can create device files for its partitions. This is done using the kpartx command.

You may have to install kpartx first. On Debian and Ubuntu, the package is called kpartx.

The command to create device files for the partitions on /dev/loop0 is

```
# kpartx -av /dev/loop0
add map loop0p1 (254:0): 0 20480 linear /dev/loop0 2048
add map loop0p2 (254:1): 0 102400 linear /dev/loop0 22528
```

(without the "-v" command, kpartx keeps quiet). The device files then appear in the /dev/mapper directory:

```
# ls /dev/mapper
control  loop0p1  loop0p2
```

Now nothing prevents you from, e. g., creating file systems on these "partitions" and mounting them into your computer's directory structure. See chapter 7.

When you no longer need the device files for the partitions, you can remove them again using the

```
# kpartx -dv /dev/loop0
del devmap : loop0p2
del devmap : loop0p1
```

command. An unused loop device can be released using

```
# losetup -d /dev/loop0
```

The

```
# losetup -D
```

command releases all loop devices.

Exercises

6.11 [!2] Use the test "disk" from exercise 6.5. Assign it a loop device using `losetup` and make its partitions accessible with `kpartx`. Convince yourself that the correct device files show up in `/dev/mapper`. Then release the partitions and the loop device again.

6.7 The Logical Volume Manager (LVM)

Partitioning a disk and creating file systems on it seems like a simple and obvious thing to do. On the other hand, you are committing yourself: The partition scheme of a hard disk can only be changed with great difficulty and, if the disk in question contains the root file system, may even involve the use of a "rescue system". In addition, there is no compelling reason why you should be constrained in your system architecture by trivialities such as the limited capacity of hard disks and the fact that file system can be at most as large as the partitions they are sitting on.

One method to transcend these limitations is the use of the "Logical Volume Manager" (LVM). LVM provides an abstraction layer between disks (and disk partitions) and file systems—instead of creating file systems directly on partitions, you can contribute partitions (or whole disks) to a "pool" of disk space and then allocate space from that pool to create file systems. Single file systems can freely use space which is located on more than one physical disk.

In LVM terminology, disks and disk partitions are considered "physical volumes" (PV) which are made part of a "volume group" (VG). There may be more than one VG on the same computer. The space within a VG is available for creating "logical volumes" (LV), which can then hold arbitrary file systems or swap space.

When creating LVs, you can cause their storage space to be spread deviously across several physical disks ("striping") or multiple copies of their data to be stored in several places within the VG at the same time ("mirroring"). The former is supposed to decrease retrieval time (even if there is a danger of losing data whenever *any* of the disks in the volume group fail), the latter is supposed to reduce the risk of losing data (even if you are paying for this with increased access times). In real life, you will probably prefer to rely on (hardware or software) RAID instead of using LVM's features.

One of the nicer properties of LVM is that LVs can be changed in size while the system is running. If a file system is running out of space, you can first enlarge the underlying LV (as long as your VG has unused space available—otherwise you would first need to install another disk and add it to the VG). Afterwards you can enlarge the file system on the LV in question.

This presumes that the file system in question enables size changes after the fact. With the popular file systems, e. g., `ext3` or `ext4`, this is the case. They even allow their size to be increased while the file system is mounted. (You will need to unmount the file system to reduce the size.)

If you use a file system that does not let itself be enlarged, you will have to bite the bullet, copy the data elsewhere, recreate the file system with the new size, and copy the data back.

If a disk within your VG should start acting up, you can migrate the LVs from that disk to another within the VG (if you still have or can make enough space). After that, you can withdraw the flaky disk from the VG, install a new disk, add that to the VG and migrate the LVs back.

You can do that, too, while the system is running and with your users none the wiser—at least as long as you have invested enough loose change into making your hard disks "hot-swappable".

“snapshots”

Also nice are “snapshots”, which you can use for backup copies without having to take your system offline for hours (which would otherwise be necessary to ensure that nothing changes while the backup is being performed). You can “freeze” the current state of an LV on another (new) LV—which takes a couple of seconds at most—and then make a copy of that new LV in your own time while normal operations continue on the old LV.

The “snapshot” LV only needs to be big enough to hold the amount of changes to the original LV you expect while the backup is being made (plus a sensible safety margin), since only the changes are being stored inside the new LV. Hence, nobody prevents you from making a snapshot of your 10 TB file system even if you don’t have another 10 TB of free disk space: If you only expect 10 GB of data to be changed while you’re writing the copy to tape, a snapshot LV of 20–30 GB should be fairly safe.

As a matter of fact it is now possible to create writable snapshots. This is useful, for example, if you are working with “virtual machines” that share a basic OS installation but differ in details. Writable snapshots make it possible to make the basic installation in one LV for all virtual machines and then store the configuration specific to each virtual machine in one LV with a writable snapshot each. (You shouldn’t overstretch this approach, though; if you change the LV with the basic installation the virtual machines won’t notice.)

On Linux, LVM is a special application of the “device mapper”, a system component enabling the flexible use of block devices. The device mapper also provides other useful features such as encrypted disks or space-saving storage provisioning for “virtual servers”. Unfortunately we do not have room in this training manual to do LVM and the device mapper justice, and refer you to the manual, *Linux Storage and File Systems* (STOR).

Commands in this Chapter

cfdisk	Character-screen based disk partitioner	cfdisk(8)	93
gdisk	Partitioning tool for GPT disks	gdisk(8)	92
kpartx	Creates block device maps from partition tables	kpartx(8)	94
losetup	Creates and maintains loop devices	losetup(8)	94
sfdisk	Non-interactive hard disk partitioner	sfdisk(8)	93
sgdisk	Non-interactive hard disk partitioning tool for GPT disks	sgdisk(8)	93

Summary

- Linux supports all notable types of mass storage device—magnetic hard disks (SATA, P-ATA, SCSI, SAS, Fibre Channel, USB, …), SSDs, USB thumb drives, SD cards, …
- Storage media such as hard disks may be partitioned. Partitions allow the independent management of parts of a hard disk, e. g., with different file systems or operating systems.
- Linux can deal with storage media partitioned according to the MBR and GPT schemes.
- Linux manages most storage media like SCSI devices. There is an older infrastructure for P-ATA disks which is only rarely used.
- Linux offers various tools for partitioning such as `fdisk`, `parted`, `gdisk`, `cfdisk`, or `sfdisk`. Various distributions also provide their own tools.
- Loop devices make block-oriented devices from files. Partitions on loop devices can be made accessible using `kpartx`.
- The Logical Volume Manager (LVM) decouples physical storage space on media from logical storage structures. It enables the flexible management of mass storage, e. g., to create file systems which are larger than a single physical storage medium. Snapshots help create backup copies and provision storage space for virtual machines.

Bibliography

SCSI-2.4-HOWTO03 Douglas Gilbert. "The Linux 2.4 SCSI subsystem HOWTO", May 2003. `http://www.tldp.org/HOWTO/SCSI-2.4-HOWTO/`

7

File Systems: Care and Feeding

Contents

Goals

- Knowing the most important file systems for Linux and their properties
- Being able to generate file systems on partitions and storage media
- Knowing about file system maintenance tools
- Being able to manage swap space
- Being able to mount local file systems
- Knowing how to set up disk quotas for users and groups

Prerequisites

- Competent use of the commands to handle files and directories
- Knowledge of mass storage on Linux and partitioning (Chapter 6)
- Existing knowledge about the structure of PC disk storage and file systems is helpful

adm1-dateisysteme.tex (33e55eeadba676a3)

7.1 Creating a Linux File System

7.1.1 Overview

After having created a new partition, you must "format" that partition, i. e., write the data structures necessary to manage files and directories onto it. What these data structures look like in detail depends on the "file system" in question.

Unfortunately, the term "file system" is overloaded on Linux. It means, for example:

1. A method to arrange data and management information on a medium ("the `ext3` file system")
2. Part of the file hierarchy of a Linux system which sits on a particular medium or partition ("the root file system", "the `/var` file system")
3. All of the file hierarchy, across media boundaries

The file systems (meaning 1 above) common on Linux may differ considerably. On the one hand, there are file systems that were developed specifically for Linux, such as the "`ext` filesystems" or the Reiser file system, and on the other hand there are file systems that originally belonged to other operating systems but that Linux supports (to a greater or lesser degree) for compatibility. This includes the file systems of DOS, Windows, OS X, and various Unix variants as well as "network file systems" such as NFS or SMB which allow access to file servers via the local network.

Many file systems "native" to Linux are part of the tradition of file systems common on Unix, like the Berkeley Fast Filesystem (FFS), and their data structures are based on those. However, development did not stop there; more modern influences are constantly being integrated in order to keep Linux current with the state of the art.

Btrfs (pronounced "butter eff ess") by Chris Mason (Fusion-IO) is widely considered the answer to the famous ZFS of Solaris. (The source code for ZFS is available but cannot be integrated in the Linux directly, due to licensing considerations.) Its focus is on "fault tolerance, repairs and simple administration". By now it seems to be mostly usable, at least some distributions rely on it.

superblock

With Linux file systems it is common to have a **superblock** at the beginning of the file system. This contains information pertaining to the file system as a whole—such as when it was last mounted or unmounted, whether it was unmounted "cleanly" or because of a system crash, and so on. The superblock normally also points to other parts of the management data structures, like where the inodes or free/occupied block lists are to be found and which parts of the medium are available for data.

It is usual to keep spare copies of the superblock elsewhere on the file system, in case something happens to the original. This is what the `ext` file systems do.

On disk, there is usually a "boot sector" in front of the superblock, into which you can theoretically put a boot loader (chapter 8). This makes it possible to, e. g., install Linux on a computer alongside Windows and use the Windows boot manager to start the system.

`mkfs`

On Linux, file systems (meaning 2 above) are created using the `mkfs` command. `mkfs` is independent of the actual file system (meaning 1) desired; it invokes the real routine specific to the file system (meaning 1) in question, `mkfs.`⟨*file system name*⟩. You can select a file system type by means of the `-t` option—with "`mkfs -t ext2`", for example, the `mkfs.ext2` program would be started (another name for `mke2fs`).

inconsistent state

When the computer has been switched off inadvertently or crashed, you have to consider that the file system might be in an inconsistent state (even though this happens very rarely in real life, even on crashes). File system errors can occur because write operations are cached inside the computer's RAM and may be lost if the system is switched off before they could be written to disk. Other errors can come up when the system gives up the ghost in the middle of an unbuffered write operation.

structural errors

Besides data loss, problems can include errors within the file system management structure. These can be located and repaired using suitable programs and include

- Erroneous directory entries
- Erroneous inode entries
- Files that do not occur in any directory
- Data blocks belonging to several different files

Most but not all such problems can be repaired automatically without loss of data; generally, the file system can be brought back to a consistent state.

valid flag

On boot, the system will find out whether it has not been shut down correctly by checking a file system's state. During a regular shutdown, the file systems are unmounted and the "valid flag" in every file system's super block will be set. On boot, this super block information may be used to automatically check these possibly-erroneous file systems and repair them if necessary—before the system tries to mount a file system whose valid flag is not set, it tries to do a file system check.

With all current Linux distributions, the system initialisation scripts executed by `init` after booting contain all necessary commands to perform a file system check.

file system check

If you want to check the consistency of a file system you do not need to wait for the next reboot. You can launch a file system check at any time. Should a file contain errors, however, it can only be repaired if it is not currently mounted. This restriction is necessary so that the kernel and the repair program do not "collide". This is another argument in favour of the automatic file system checks during booting.

Actual consistency checks are performed using the `fsck` command. Like `mkfs`, depending on the type of the file system to be checked this command uses a specific sub-command called `fsck.`⟨*type*⟩—e.g., `fsck.ext2` for `ext2`. `fsck` identifies the required sub-command by examining the file system in question. Using the

```
# fsck /dev/sdb1
```

command, for example, you can check the file system on `/dev/sdb1`.

The simple command

```
# fsck
```

checks all file systems listed in `/etc/fstab` with a non-zero value in the sixth (last) column in sequence. (If several different values exist, the file systems are checked in ascending order.) `/etc/fstab` is explained in more detail in section 7.2.2.

`fsck` supports a `-t` option which at first sight resembles `mkfs` but has a different meaning: A command like

```
# fsck -t ext3
```

checks all file systems in `/etc/fstab` that are marked as type `ext3` there.

options

The most important options of `fsck` include:

`-A` (All) causes `fsck` to check all file systems mentioned in `/etc/fstab`.

This obeys the checking order in the sixth column of the file. If several file systems share the same value in that column, they are checked in parallel if they are located on different physical disks.

`-R` With `-A`, the root file system is not checked (which is useful if it is already mounted for writing).

`-V` Outputs verbose messages about the check run.

`-N` Displays what `fsck` would do without actually doing it.

`-s` Inhibits parallel checking of multiple file systems. The "`fsck`" command without any parameters is equivalent to "`fsck -A -s`".

Besides its own options, you can pass additional options to `fsck` which it will forward to the specific checking program. These must occur after the name of the file system(s) to be checked and possibly a "`--`" separator. The `-a`, `-f`, `-p` and `-v` options are supported by most such programs. Details may be found within the documentation for the respective programs. The

```
# fsck /dev/sdb1 /dev/sdb2 -pv
```

for example would check the file systems on the `/dev/sdb1` and `/dev/sdb2` partitions automatically, fix any errors without manual intervention and report verbosely on its progress.

At program termination, `fsck` passes information about the file system state to the shell:

`0` No error was found in the file system

`1` Errors were found and corrected

`2` Severe errors were found and corrected. The system should be rebooted

`4` Errors were found but not corrected

`8` An error occurred while the program was executed

`16` Usage error (e. g., bad command line)

`128` Error in a shared library function

It is conceivable to analyse these return values in an init script and determine how to continue with the system boot. If several file systems are being checked (using the `-A` option), the return value of `fsck` is the logical OR of the return values of the individual checking programs.

7.1.2 The `ext` File Systems

History and Properties The original "extended file system" for Linux was implemented in April, 1992, by Rémy Card. It was the first file system designed specifically for Linux (although it did take a lot of inspiration from general Unix file systems) and did away with various limitations of the previously popular Minix file system.

The Minix file system had various nasty limits such as a maximum file system size of 64 MiB and file names of at most 14 characters. (To be fair, Minix was introduced when the IBM PC XT was considered a hot computer and 64 MiB, for PCs, amounted to an unimaginably vast amount of disk storage. By 1990, that assumption had begun to crumble.) `ext` allowed file systems of up to 2 GiB—quite useful at the time, but naturally somewhat ridiculous today.

The arrival of the `ext` file system marks another important improvement to the Linux kernel, namely the introduction of the "virtual file system switch", or VFS. The VFS abstracts file system operations such as the opening and closing of files or the reading and writing of data, and as such enables the coexistence of different file system implementations in Linux.

The original `ext` file system is no longer used today. From here on, when we talk about "the `ext` file systems", we refer to `ext2` and everything newer than that.

The subsequent version, `ext2` (the "second extended file system"), which was begun by Rémy Card in January, 1993, amounted to a considerable rework of the original "extended file system". The development of `ext2` made use of many ideas from the BSD "Berkeley Fast Filesystem". `ext2` is still being maintained and makes eminent sense for certain applications.

Compared to `ext`, `ext2` pushes various size limits—with the 4 KiB block size typical for Intel-based Linux systems, file systems can be 16 TiB and single files 2 TiB in size. Another important improvement in `ext2` was the introduction of separate timestamps for the last access, last content modification and last inode modification, which achieved compatibility to "traditional" Unix in this respect.

From the beginning, `ext2` was geared towards continued development and improvement: Most data structures contained surplus space which was later used for important extensions. These include ACLs and "extended attributes".

Since the end of the 1990s, Stephen Tweedie worked on a successor to `ext2`, which was made part of the Linux kernel at the end of 2001 under the name of `ext3`. (That was Linux 2.4.15.) The most important differences between `ext2` and `ext3` include:

- `ext3` supports Journaling.
- `ext3` allows enlarging file systems while they are mounted.
- `ext3` supports more efficient internal data structures for directories with many entries.

Even so it is largely compatible with `ext2`. It is usually possible to access `ext3` file systems as `ext2` file systems (which implies that the new features cannot be used) and vice-versa.

"Journaling" solves a problem that can be very tedious with the increasing size of file systems, namely that an unforeseen system crash makes it necessary to do a complete consistency check of the file system. The Linux kernel does not perform write operations immediately, but buffers the data in RAM and writes them to disk when that is convenient (e. g., when the read/write head of the disk drive is in the appropriate place). In addition, many write operations involve writing data to various places on the disk, e. g., one or more data blocks, the inode table, and the list of available blocks on the disk. If the power fails in the right (or wrong) moment, such an operation can remain only half-done—the file system is "inconsistent" in the sense that a data block can be assigned to a file in the inode, but not marked used in the free-block list. This can lead to serious problems later on.

A journaling file system like `ext3` considers every write access to the disk as a "transaction" which must be performed completely or not at all. By definition, the file system is consistent before and after a transaction is performed. Every transaction is first written into a special area of the file system called the *journal*. If it has been entirely written, it is marked "complete" and, as such, it is official. The Linux kernel can do the actual write operations later.—If the system crashes, a journaling file system does not need to undergo a complete file system check, which with today's file system sizes could take hours or even days. Instead, the journal is considered and any transactions marked "complete" are transferred to the actual file system. Transactions not marked "complete" are thrown out.

Most journaling file systems use the journal to log changes to the file system's "metadata", i. e., directories, inodes, etc. For efficiency, the actual file data are normally not written to the journal. This means that after a crash and reboot you will have a consistent file system without having to spend hours or days on a complete consistency check. However, your file contents may have been scrambled—for example, a file might contain obsolete data blocks because the updated ones couldn't be written before the crash. This problem can be mitigated by writing the data blocks to disk first and then the metadata to the journal, but even that is not without risk. `ext3` gives you the choice between three operating modes—writing everything to the journal (mount option `data=journal`), writing data blocks directly and then metadata to the journal (`data=ordered`), or no restrictions (`data=writeback`). The default is `data=ordered`.

Writing metadata or even file data twice—once to the journal, and then later to the actual file system—involves a certain loss of performance compared to file systems like `ext2`, which ignore the problem. One approach to fix this consists of *log-structured file systems*, in which the journal makes up the actual file system. Within the Linux community, this approach has so far not prevailed. Another approach is exemplified by "copy-on-write filesystems" like Btrfs.

Using a journaling file system like `ext3` does not absolve you from having to perform complete consistency checks every so often. Errors in a file system's data structures might arise through disk hardware errors, cabling problems, or the dreaded cosmic rays (don't laugh) and might otherwise remain unnoticed until they wreak havoc. For this reason, the `ext` file systems force a file system check every so often when the system is booted (usually when you can least afford it). You will see how to tweak this later in this chapter.

With server systems that are rarely rebooted and that you cannot simply take offline for a few hours or days for a prophylactic file system check, you may have a big problem. We shall also come back to this.

The apex of `ext` file system evolution is currently represented by `ext4`, which has been developed since 2006 under the guidance of Theodore Ts'o. This has been considered stable since 2008 (Kernel version 2.6.28). Like `ext3` and `ext2`, backward compatibility was an important goal: `ext2` and `ext3` file systems can be mounted as `ext4` file systems and will profit from some internal improvements in `ext4`. On the other hand, the `ext4` code introduces some changes that result in file systems no longer being accessible as `ext2` and `ext3`. Here are the most important improvements in `ext4` as compared to `ext3`:

- Instead of maintaining the data blocks of individual files as lists of block numbers, `ext4` uses "extents", i. e., groups of physically contiguous blocks on disk. This leads to a considerable simplification of space management and to greater efficiency, but makes file systems using extents incompatible to `ext3`. It also avoids fragmentation, or the wild scattering of blocks belonging to the same file across the whole file system.

- When data is written, actual blocks on the disk are assigned as late as possible. This also helps prevent fragmentation.
- User programs can advise the operating system how large a file is going to be. Again, this can be used to assign contiguous file space and mitigate fragmentation.
- `Ext4` uses checksums to safeguard the journal. This increases reliability and avoids some hairy problems when the journal is replayed after a system crash.
- Various optimisations of internal data structures increase the speed of consistency checks.
- Timestamps now carry nanosecond resolution and roll over in 2242 (rather than 2038).
- Some size limits have been increased—directories may now contain 64,000 or more subdirectories (previously 32,000), files can be as large as 16 TiB, and file systems as large as 1 EiB.

In spite of these useful improvements, according to Ted Ts'o `ext4` is not to be considered an innovation, but rather a stopgap until even better file systems like Btrfs become available.

All `ext` file systems include powerful tools for consistency checks and file system repairs. This is very important for practical use.

Creating `ext` file systems To create a `ext2` or `ext3` file system, it is easiest to use the `mkfs` command with a suitable `-t` option:

```
# mkfs -t ext2 /dev/sdb1                                    ext2 file system
# mkfs -t ext3 /dev/sdb1                                    ext3 file system
# mkfs -t ext4 /dev/sdb1                                    ext4 file system
```

After the `-t` option and its parameter, you can specify further parameters which will be passed to the program performing the actual operation—in the case of the `ext` file systems, the `mke2fs` program. (In spite of the `e2` in its name, it can also create `ext3` and `ext4` file systems.)

The following commands would also work:

```
# mkfs.ext2 /dev/sdb1                                 ext2 file system
# mkfs.ext3 /dev/sdb1                                 ext3 file system
# mkfs.ext4 /dev/sdb1                                 ext4 file system
```

These are exactly the commands that `mkfs` would invoke. All three commands are really symbolic links referring to `mke2fs`; `mke2fs` looks at the name used to call it and behaves accordingly.

You can even call the `mke2fs` command directly: `mke2fs`

```
# mke2fs /dev/sdb1
```

(Passing no options will get you a `ext2` file system.)

The following options for `mke2fs` are useful (and possibly important for the exam):

`-b` ⟨*size*⟩ determines the block size. Typical values are 1024, 2048, or 4096. On partitions of interesting size, the default is 4096.

-c checks the partition for damaged blocks and marks them as unusable.

Current hard disks can notice "bad blocks" and replace them by blocks from a "secret reserve" without the operating system even noticing (at least as long as you don't ask the disk directly). While this is going on, "mke2fs -c") does not provide an advantage. The command will only find bad blocks when the secret reserve is exhausted, and at that point you would do well to replace the disk, anyway. (A completely new hard disk would at this point be a warranty case. Old chestnuts are only fit for the garbage.)

-i ⟨*count*⟩ determines the "inode density"; an inode is created for every ⟨*count*⟩ bytes of space on the disk. The value must be a multiple of the block size (option b); there is no point in selecting a ⟨*count*⟩ that is less than the block size. The minimum value is 1024, the default is the current block size.

-m ⟨*percentage*⟩ sets the percentage of data blocks reserved for root (default: 5%)

-S causes mke2fs to rewrite just the super blocks and group descriptors and leave the inodes intact

-j creates a journal and, hence, an ext3 or ext4 file system.

It is best to create an ext4 file system using one of the precooked calls like "mkfs -t ext4", since mke2fs then knows what it is suppsed to do. If you must absolutely do it manually, use something like

```
# mke2fs -j -O extents,uninit_bg,dir_index /dev/sdb1
```

internal fragmentation

The ext file systems (still) need at least one complete data block for every file, no matter how small. Thus, if you create an ext file system on which you intend to store many small files (cue: mail or Usenet server), you may want to select a smaller block size in order to avoid internal fragmentation. (On the other hand, disk space is really quite cheap today.)

The inode density (-i option) determines how many files you can create on the file system—since every file requires an inode, there can be no more files than there are inodes. The default, creating an inode for every single data block on the disk, is very conservative, but from the point of view of the developers, the danger of not being able to create new files for lack of inodes seems to be more of a problem than wasted space due to unused inodes.

Various file system objects require inodes but no data blocks—such as device files, FIFOs or short symbolic links. Even if you create as many inodes as data blocks, you can still run out of inodes before running out of data blocks.

Using the mke2fs -F option, you can "format" file system objects that are not block device files. For example, you can create CD-ROMs containing an ext2 file system by executing the command sequence

```
# dd if=/dev/zero of=cdrom.img bs=1M count=650
# mke2fs -F cdrom.img
# mount -o loop cdrom.img /mnt
#                                                   … copy stuff to /mnt …
# umount /mnt
# cdrecord -data cdrom.img
```

(/dev/zero is a "device" that produces arbitrarily many zero bytes.) The resulting CD-ROMs contain "genuine" ext2 file systems with all permissions, attributes, ACLs etc., and can be mounted using

```
# mount -t ext2 -o ro /dev/scd0 /media/cdrom
```

(or some such command); you should replace `/dev/scd0` by the device name of your optical drive. (It is best to avoid using an `ext3` file system here, since the journal would be an utter waste of space. An `ext4` file system, though, can be created without a journal.)

e2fsck

Repairing `ext` file systems `e2fsck` is the consistency checker for `ext` file systems. There are usually symbolic links such as `fsck.ext2` so it can be invoked from `fsck`.

Like `mke2fs`, `e2fsck` also works for `ext3` and `ext4` file systems.

You can of course invoke the program directly, which might save you a little typing when passing options. On the other hand, you can only specify the name of one single partition (strictly speaking, one single block device).

options

The most important options for `e2fsck` include:

`-b` ⟨*number*⟩ reads the super block from block ⟨*number*⟩ of the partition (rather than the first super block)

`-B` ⟨*size*⟩ gives the size of a block group between two copies of the super block; with the `ext` file systems, backup copies of the super block are usually placed every 8192 blocks, on larger disks every 32768 blocks. (You can query this using the `tune2fs` command explained below; look for "blocks per group" in the output of "`tune2fs -l`".)

`-f` forces a file system to be checked even if its super block claims that it is clean

`-l` ⟨*file*⟩ reads the list of bad blocks from the ⟨*file*⟩ and marks these blocks as "used"

`-c` ("check") searches the file system for bad blocks

`-p` ("preen") causes errors to be repaired automatically with no further user interaction

`-v` ("verbose") outputs information about the program's execution status and the file system while the program is running

The device file specifies the partition whose file system is to be checked. If that partition does not contain an `ext` file system, the command aborts. `e2fsck` performs the following steps:

steps

1. The command line arguments are checked
2. The program checks whether the file system in question is mounted
3. The file system is opened
4. The super block is checked for readability
5. The data blocks are checked for errors
6. The super block information on inodes, blocks and sizes are compared with the current system state
7. Directory entries are checked against inodes
8. Every data block that is marked "used" is checked for existence and whether it is referred to exactly once by some inode
9. The number of links within directories is checked with the inode link counters (must match)

10. The total number of blocks must equal the number of free blocks plus the number of used blocks

exit code

e2fsck returns an exit code with the same meaning as the standard fsck exit codes..

It is impossible to list all the file system errors that e2fsck can handle. Here are a few important examples:

- Files whose inodes are not referenced from any directory are placed in the file system's lost+found directory using the inode number as the file name and can be moved elsewhere from there. This type of error can occur, e. g., if the system crashes after a file has been created but before the directory entry could be written.
- An inode's link counter is greater than the number of links pointing to this inode from directories. e2fsck corrects the inode's link counter.
- e2fsck finds free blocks that are marked used (this can occur, e. g., when the system crashes after a file has been deleted but before the block count and bitmaps could be updated).
- The total number of blocks is incorrect (free and used blocks together are different from the total number of blocks).

complicated errors

Not all errors are straightforward to repair. What to do if the super block is unreadable? Then the file system can no longer be mounted, and e2fsck often fails as well. You can then use a copy of the super block, one of which is included with every block group on the partition. In this case you should boot a rescue system and invoke fsck from there. With the -b option, e2fsck can be forced to consider a particular block as the super block. The command in question then becomes, for example:

```
# e2fsck -f -b 8193 /dev/sda2
```

If the file system cannot be automatically repaired using fsck, it is possible to modify the file system directly. However, this requires very detailed knowledge of file system structures which is beyond the scope of this course.—There are two useful tools to aid with this. First, the dumpe2fs program makes visible the internal management data structures of a ext file system. The interpretation of its output requires the aforementioned detailed knowledge. An ext file system may be repaired using the debugfs file system debugger.

You should keep your hands off programs like debugfs unless you know exactly what you are doing. While debugfs enables you to manipulate the file system's data structures on a very low level, it is easy to damage a file system even more by using it injudiciously. Now that we have appropriately warned you, we may tell you that

```
# debugfs /dev/sda1
```

will open the ext file system on /dev/sda1 for inspection (debugfs, reasonably, enables writing to the file system only if it was called with the -w option). debugfs displays a prompt; "help" gets you a list of available commands. These are also listed in the documentation, which is in debugfs(8).

Querying and Changing `ext` File System Parameters If you have created a partition and put an `ext` file system on it, you can change some formatting parameters after the fact. This is done using the `tune2fs` command, which should be used with utmost caution and should never be applied on a file system mounted for writing:

changing format parameters

```
tune2fs [⟨options⟩] ⟨device⟩
```

The following options are important:

`-c` ⟨*count*⟩ sets the maximum number of times the file system may be mounted between two routine file system checks. The default value set by `mke2fs` is a random number somewhere around 30 (so that not all file systems are preemptively checked at the same time). The value 0 means "infinitely many".

`-C` ⟨*count*⟩ sets the current "mount count". You can use this to cheat `fsck` or (by setting it to a larger value than the current maximum set up using `-c`) force a file system check during the next system boot.

`-e` ⟨*behaviour*⟩ determines the behaviour of the system in case of errors. The following possibilities exist:

`continue` Go on as normal

`remount-ro` Disallow further writing to the file system

`panic` Force a kernel panic

In every case, a file system consistency check will be performed during the next reboot.

`-i` ⟨*interval*⟩⟨*unit*⟩ sets the maximum time between two routine file system checks. ⟨*interval*⟩ is an integer; the ⟨*unit*⟩ is `d` for days, `w` for weeks and `m` for months. The value 0 means "infinitely long".

`-l` displays super block information.

`-m` ⟨*percent*⟩ sets the percentage of data blocks reserved for `root` (or the user specified using the `-u` option). The default value is 5%.

`-L` ⟨*name*⟩ sets a partition name (up to 16 characters). Commands like `mount` and `fsck` make it possible to refer to partitions by their names rather than the names of their device files.

To upgrade an existing `ext3` file system to an `ext4` file system, you need to execute the commands

```
# tune2fs -O extents,uninit_bg,dir_index /dev/sdb1
# e2fsck -fDp /dev/sdb1
```

(stipulating that the file system in question is on `/dev/sdb1`). Make sure to change `/etc/fstab` such that the file system is mounted as `ext4` later on (see section 7.2).

Do note, though, that all existing files will still be using `ext3` structures—improvements like extents will only be used for files created later. The `e4defrag` defragmentation tool is supposed to convert older files but is not quite ready yet.

If you have the wherewithal, you should not upgrade a file system "in place" but instead backup its content, recreate the file system as `ext4`, and the restore the content. The performance of `ext4` is considerably better on "native" `ext4` file systems than on converted `ext3` file systems—this can amount to a factor of 2.

If you have ext2 file systems lying around that you would like to convert into ext3 file systems: This is easily done by creating a journal. tune2fs will do that for you, too:

```
# tune2fs -j /dev/sdb1
```

Again, you will have to adjust /etc/fstab if necessary.

Exercises

7.1 [!2] Generate an ext4 file system on a suitable medium (hard disk partition, USB thumb drive, file created using dd).

7.2 [2] Change the maximum mount count of the filesystem created in exercise 7.1 to 30. In addition, 30% of the space available on the file system should be reserved for user test.

7.1.3 ReiserFS

Overview ReiserFS is a Linux file system meant for general use. It was developed by a team under the direction of Hans Reiser and debuted in Linux 2.4.1 (that was in 2001). This made it the first journaling file system available for Linux. ReiserFS also contained some other innovations that the most popular Linux file system at the time, ext2, did not offer:

- Using a special tool, ReiserFS file systems could be changed in size. Enlargement was even possible while the file system was mounted.
- Small files and the ends of larger files could be packed together to avoid "internal fragmentation" which arises in file systems like ext2 because space on disk is allocated based on the block size (usually 4 KiB). With ext2 and friends, even a 1-byte file requires a full 4-KiB block, which could be considered wasteful (a 4097-byte file requires two data blocks, and that is almost as bad). With ReiserFS, several such files could share one data block.

  There is nothing in principle that would keep the ext developers to add this "tail packing" feature to the ext file systems. This was discussed and the consensus was that by now, disk space is cheap enough that the added complexity would be worth the trouble.

- Inodes aren't pregenerated when the file system is created, but are allocated on demand. This avoids a pathological problem possible with the ext file systems, where there are blocks available in the file system but all inodes are occupied and no new files can be generated.

  The ext file systems mitigate this problem by allocating one inode per data block per default (the inode density corresponds to the block size). This makes it difficult to provoke the problem.

- ReiserFS uses trees instead of lists (like ext2) for its internal management data structures. This makes it more efficient for directories with many files.

  Ext3 and in particular ext4 can by now do that too.

  As a matter of fact, ReiserFS uses the same tree structure not just for directory entries, but also for inodes, file metadata and file block lists, which leads to a performance increase in places but to a decrease in others.

For a long time, ReiserFS used to be the default file system for the SUSE distributions (and SUSE contributed to the project's funding). Since 2006, Novell/SUSE has moved from ReiserFS to ext3; very new SLES versions use Btrfs for their root file system.

In real life you should give the Reiser file system (and its designated successor, Reiser4) a wide berth unless you need to manage older systems using it. This is less to do with the fact that Hans Reiser was convicted of his wife's murder (which of course does not speak in his favour as a human being, but things like these do happen not just among Linux kernel developers), but more with the fact that the Reiser file system does have its good points but is built on a fairly brittle base. For example, certain directory operations in ReiserFS break basic assumptions that are otherwise universally valid for Unix-like file systems. This means, for instance, that mail servers storing mailboxes on a ReiserFS file system are less resilient against system crashes than ones using different file systems. Another grave problem, which we will talk about briefly later on, is the existence of technical flaws in the file system repair program. Finally—and that may be the most serious problem—nobody seems to maintain the code any longer.

`mkreiserfs`
`dumpreiserfs`
`resize_reiserfs`

Creating ReiserFS file systems `mkreiserfs` serves to create a ReiserFS file system. The possible specification of a logical block size is currently ignored, the size is always 4 KiB. With `dumpreiserfs` you can determine information about ReiserFS file systems on your disk. `resize_reiserfs` makes it possible to change the size of currently-unused ReiserFS partitions. Mounted partitions may be resized using a command like "`mount -o remount,resize=`⟨*block count*⟩ ⟨*mount point*⟩".

Reiser file system

Consistency Checks for ReiserFS For the Reiser file system, too, there is a checking and repair program, namely `reiserfsck`.

`reiserfsck` performs a consistency check and tries to repair any errors found, much like `e2fsck`. This program is only necessary if the file system is really damaged. Should a Reiser file system merely have been unmounted uncleanly, the kernel will automatically try to restore it according to the journal.

`reiserfsck` has some serious issues. One is that when the tree structure needs to be reconstructed (which may happen in certain situations) it gets completely mixed up if data files (!) contain blocks that might be misconstrued as another ReiserFS file system's superblock. This will occur if you have an image of a ReiserFS file system in a file used as a ReiserFS-formatted "virtual" hard disk for a virtualisation environment such as VirtualBox or VMware. This effectively disqualifies the ReiserFS file system for serious work. You have been warned.

Exercises

7.3 [!1] What is the command to create a Reiser file system on the first logical partition of the second disk?

7.1.4 XFS

XFS

The XFS file system was donated to Linux by SGI (the erstwhile Silicon Graphics, Inc.); it is the file system used by SGI's Unix variant, IRIX, which is able to handle very large files efficiently. All Linux distributions of consequence offer XFS support, even though few deploy it by default; you may have to install the XFS tools separately.

In some circles, "XFS" is the abbreviation of "X11 Font Server". This can occur in distribution package names. Don't let yourself be confused.

You can create an XFS file system on an empty partition (or file) using the

```
# mkfs -t xfs /dev/sda2
```

command (insert the appropriate device name). Of course, the real work is done by a program called mkfs.xfs. You can control it using various options; consult the documentation (xfs(5) and mkfs.xfs(8)).

If performance is your goal, you can, for example, create the journal on another (physical) storge medium by using an option like "-l logdev=/dev/sdb1,size=10000b". (The actual file system should of course not be on /dev/sdb, and the partition for the journal should not otherwise be used.)

The XFS tools contain a fsck.xfs (which you can invoke using "fsck -t xfs"), but this program doesn't really do anything at all—it is merely there to give the system something to call when "all" file systems are to be checked (which is easier than putting a special exception for XFS into fsck). In actual fact, XFS file systems are checked automatically on mounting if they have not been unmounted cleanly. If you want to check the consistency of an XFS or have to repair one, use the xfs_repair(8) program—"xfs_repair -n" checks whether repairs are required; without the option, any repairs will be performed outright.

In extreme cases xfs_repair may not be able to repair the file system. In such a situation you can use xfs_metadump to create a dump of the filesystem's metadata and send that to the developers:

```
# xfs_metadump /dev/sdb1 sdb1.dump
```

(The file system must not be mounted when you do this.) The dump is a binary file that does not contain actual file data and where all file names have been obfuscated. Hence there is no risk of inadvertently passing along confidential data.

A dump that has been prepared using xfs_metadump can be written back to a file system (on a "real" storage medium or an image in a file) using xfs_mdrestore. This will not include file contents as these aren't part of the dump to begin with. Unless you are an XFS developer, this command will not be particularly interesting to you.

The xfs_info command outputs information about a (mounted) XFS file system:

```
# xfs_info /dev/sdb1
meta-data=/dev/sdb1              isize=256    agcount=4, agsize=16384 blks
         =                       sectsz=512   attr=2, projid32bit=1
         =                       crc=0        finobt=0
data     =                       bsize=4096   blocks=65536, imaxpct=25
         =                       sunit=0      swidth=0 blks
naming   =version 2              bsize=4096   ascii-ci=0 ftype=0
log      =Intern                 bsize=4096   blocks=853, version=2
         =                       sectsz=512   sunit=0 blks, lazy-count=1
realtime =keine                  extsz=4096   blocks=0, rtextents=0
```

You can see, for example, that the file system consists of 65536 blocks of 4 KiB each (bsize and blocks in the data section), while the journal occupies 853 4-KiB blocks in the same file system (Intern, bsize and blocks in the log section).

The same information is output by mkfs.xfs after creating a new XFS file system.

You should avoid copying XFS file systems using dd (or at least proceed very cautiously). This is because every XFS file system contains a unique UUID, and programs like xfsdump (which makes backup copies) can get confused if they run into two independent file systems using the same UUID. To copy XFS file systems, use xfsdump and xfsrestore or else xfs_copy instead.

7.1.5 Btrfs

Btrfs is considered the up-and-coming Linux file system for the future. It combines the properties traditionally associated with a Unix-like file system with some innovative ideas that are partly based on Solaris's ZFS. Besides some features otherwise provided by the Logical Volum Manager (LVM; Section 6.7)—such as the creation of file systems that span several physical storage media—or provided by the Linux kernel's RAID support—such as the redundant storage of data on several physical media—this includes transparent data compression, consistency checks on data blocks by means of checksums, and various others. The "killer feature" is probably snapshots that can provide views of different versions of files or complete file hierarchies simultaneously.

Btrfs is several years younger than ZFS, and its design therefore contains a few neat ideas that hadn't been invented yet when ZFS was first introduced. ZFS is currently considered the "state of the art" in file systems, but it is to be expected that some time in the not-too-distant future it will be overtaken by Btrfs.

Btrfs is based, in principle, on the idea of "copy on write". This means that if you create a snapshot of a Btrfs file system, nothing is copied at all; the system only notes that a copy exists. The data is accessible both from the original file system and the snapshot, and as long as data is just being read, the file systems can share the complete storage. Once write operations happen either in the original file system or the snapshot, only the data blocks being modified are copied. The data itself is stored in efficient data structures called B-trees.

Btrfs file systems are created with `mkfs`, as usual:

```
# mkfs -t btrfs /dev/sdb1
```

You can also mention several storage media, which will all be made part of the new file system. Btrfs stores metadata such as directory information redundantly on several media; by default, data is spread out across various disks ("striping") in order to accelerate access[1]. You can, however, request other storage arrangements:

```
# mkfs -t btrfs -L MyBtrfs -d raid1 /dev/sdb1 /dev/sdc1
```

This example generates a Btrfs file system which encompasses the `/dev/sdb1` and `/dev/sdc1` disks and is labeled "MyBtrfs". Data is stored redundantly on both disks ("`-d raid1`").

Within Btrfs file systems you can create "subvolumes", which serve as a type of partition at the file system level. Subvolumes are the units of which you will later be able to make snapshots. If your system uses Btrfs as its root file system, the command

```
# btrfs subvolume create /home
```

would, for instance, allow you to keep your own data within a separate subvolume. Subvolumes do not take a lot of space, so you should not hesitate to create more of them rather than fewer—in particular, one for every directory of which you might later want to create independent snapshots, since it is not possible to make directories into subvolumes after the fact.

[1] In other words, Btrfs uses RAID-1 for metadata and RAID-0 for data.

You can create a snapshot of a subvolume using

```
# btrfs subvolume snapshot /mnt/sub /mnt/sub-snap
```

The snapshot (here, `/mnt/sub-snap`) is at first indistinguishable from the original subvolume (here, `/mnt/sub`); both contain the same files and are writable. At first no extra storage space is being used—only if you change files in the original or snapshot or create new ones, the system copies whatever is required.

Btrfs makes on-the-fly consistency checks and tries to fix problems as they are detected. The "`btrfs scrub start`" command starts a house-cleaning operation that recalculates the checksums on all data and metadata on a Btrfs file system and repairs faulty blocks according to a different copy if required. This can, of course, take a long time; with "`btrfs scrub status`" you can query how it is getting on, with "`btrfs scrub cancel`" you can interrupt it, and restart it later with "`btrfs scrub resume`".

There is a `fsck.btrfs` program, but it does nothing beyond outputting a message that it doesn't do anything. The program is required because something needs to be there to execute when all file systems are checked for consistency during startup. To really check or repair Btrfs file systems there is the "`btrfs check`" command. By default this does only a consistency check, and if it is invoked with the "`--repair`" it tries to actually repair any problems it found.

Btrfs is very versatile and complex and we can only give you a small glimpse here. Consult the documentation (starting at `btrfs`(8)).

Exercises

7.4 [!1] Generate a Btrfs file system on an empty partition, using "`mkfs -t btrfs`".

7.5 [2] Within your Btrfs file system, create a subvolume called `sub0`. Create some files within `sub0`. Then create a snapshot called `snap0`. Convince yourself that `sub0` and `snap0` have the same content. Remove or change a few files in `sub0` and `snap0`, and make sure that the two subvolumes are independent of each other.

7.1.6 Even More File Systems

tmpfs

`tmpfs` is a flexible implementation of a "RAM disk file system", which stores files not on disk, but in the computer's virtual memory. They can thus be accessed more quickly, but seldom used files can still be moved to swap space. The size of a `tmpfs` is variable up to a set limit. There is no special program for generating a `tmpfs`, but you can create it simply by mounting it: For example, the

```
# mount -t tmpfs -o size=1G,mode=0700 tmpfs /scratch
```

command creates a `tmpfs` of at most 1 GiB under the name of `/scratch`, which can only be accessed by the owner of the `/scratch` directory. (We shall be coming back to mounting file systems in section 7.2.)

VFAT

A popular file system for older Windows PCs, USB sticks, digital cameras, MP3 players and other "storage devices" without big ideas about efficiency and flexibility is Microsoft's venerable VFAT file system. Naturally, Linux can mount, read, and write media formatted thusly, and also create such file systems, for example by

```
# mkfs -t vfat /dev/mcblk0p1
```

(insert the appropriate device name again). At this point you will no longer be surprised to hear that mkfs.vfat is just another name for the mkdosfs program, which can create all sorts of MS-DOS and Windows file systems—including the file system used by the Atari ST of blessed memory. (As there are Linux variants running on Atari computers, this is not quite as far-fetched as it may sound.)

mkdosfs supports various options allowing you to determine the type of file system created. Most of these are of no practical consequence today, and mkdosfs will do the Right Thing in most cases, anyway. We do not want to disgress into a taxonomy of FAT file system variants and restrict ourselves to pointing out that the main difference between FAT and VFAT is that file systems of the latter persuasion allow file names that do not follow the older, strict 8 + 3 scheme. The "file allocation table", the data structure that remembers which data blocks belong to which file and that gave the file system its name, also exists invarious flavours, of which mkdosfs selects the one most suitable to the medium in question—floppy disks are endowed with a 12-bit FAT, and hard disk (partitions) or (today) USB sticks of considerable capacity get 32-bit FATs; in the latter case the resulting file system is called "VFAT32".

NTFS

NTFS, the file system used by Windows NT and its successors including Windows Vista, is a bit of an exasperating topic. Obviously there is considerable interest in enabling Linux to handle NTFS partitions—everywhere but on Microsoft's part, where so far one has not deigned to explain to the general public how NTFS actually works. (It is well-known that NTFS is based on BSD's "Berkeley Fast Filesystem", which is reasonably well understood, but in the meantime Microsoft butchered it into something barely recognisable.) In the Linux community there have been several attempts to provide NTFS support by trying to understand NTFS on Windows, but complete success is still some way off. At the moment there is a kernel-based driver with good support for reading, but questionable support for writing, and another driver running in user space which according to the grapevine works well for reading *and* writing. Finally, there are the "ntfsprogs", a package of tools for managing NTFS file systems, which also allow rudimentary access to data stored on them. Further information is available from http://www.linux-ntfs.org/.

7.1.7 Swap space

swap partition

In addition to the file system partitions, you should always create a **swap partition**. Linux can use this to store part of the content of system RAM; the effective amount of working memory available to you is thus greater than the amount of RAM in your computer.

Before you can use a swap partition you must "format" it using the mkswap command:

```
# mkswap /dev/sda4
```

This writes some administrative data to the partition.

When the system is started, it is necessary to "activate" a swap partition. This corresponds to mounting a partition with a file system and is done using the swapon command:

```
# swapon /dev/sda4
```

The partition should subsequently be mentioned in the /proc/swaps file:

```
# cat /proc/swaps
Filename                Type       Size    Used   Priority
/dev/sda4               partition  2144636 380    -1
```

After use the swap partition can be deactivated using `swapoff`:

```
# swapoff /dev/sda4
```

The system usually takes care of activating and deactivating swap partitions, as long as you put them into the `/etc/fstab` file. See section 7.2.2.

You can operate up to 32 swap partitions (up to and including kernel version 2.4.10: 8) in parallel; the maximum size depends on your computer's architecture and isn't documented anywhere exactly, but "stupendously gigantic" is a reasonable approximation. It used to be just a little less than 2 GiB for most Linux platforms.

If you have several disks, you should spread your swap space across all of them, which should increase speed noticeably.

Linux can prioritise swap space. This is worth doing if the disks containing your swap space have different speeds, because Linux will prefer the faster disks. Read up on this in `swapon`(8).

Besides partitions, you can also use files as swap space. Since Linux 2.6 this isn't even any slower! This allows you to temporarily provide space for rare humongous workloads. You must initially create a swap file as a file full of zeros, for instance by using

```
# dd if=/dev/zero of=swapfile bs=1M count=256
```

before preparing it using the `mkswap` command and activating it with `swapon`. (Desist from tricks using `dd` or `cp`; a swap file may not contain "holes".)

You can find information about the currently active swap areas in the `/proc/swaps` file.

7.2 Mounting File Systems

7.2.1 Basics

To access data stored on a medium (hard disk, USB stick, floppy, …), it would in principle be possible to access the device files directly. This is in fact being done, for example when accessing tape drives. However, the well-known file management commands (`cp`, `mv`, and so on) can only access files via the directory tree. To use these commands, storage media must be made part of the directory tree ("mounted") using their device files. This is done using the `mount` command.

mount point

The place in the directory tree where a file system is to be mounted is called a **mount point**. This can be any directory; it does not even have to be empty, but you will not be able to access the original directory content while another file system is mounted "over" it.

The content reappears once the file system is unmounted using `umount`. Even so you should restrain yourself from mounting stuff on `/etc` and other important system directories …

7.2.2 The `mount` Command

The `mount` command mounts file systems into the directory tree. It can also be used to display the currently mounted file systems, simply by calling it without parameters:

```
proc       /proc         proc  defaults                  0 0
/dev/sda2  /             ext3  defaults,errors=remount-ro 0 1
/dev/sda1  none          swap  sw                        0 0
/dev/sda3  /home         ext3  defaults,relatime         0 1
/dev/sr0   /media/cdrom0 udf,iso9660 ro,user,exec,noauto  0 0
/dev/sdb1  /media/usb    auto  user,noauto               0 0
/dev/fd0   /media/floppy auto  user,noauto,sync          0 0
```

Figure 7.1: The `/etc/fstab` file (example)

```
$ mount
/dev/sda2 on / type ext3 (rw,relatime,errors=remount-ro)
tmpfs on /lib/init/rw type tmpfs (rw,nosuid,mode=0755)
proc on /proc type proc (rw,noexec,nosuid,nodev)
sysfs on /sys type sysfs (rw,noexec,nosuid,nodev)
<<<<<
```

To mount a medium, for example a hard disk partition, you must specify its device file and the desired mount point:

```
# mount -t ext2 /dev/sda5 /home
```

It is not mandatory to specify the file system type using the `-t` option, since the kernel can generally figure it out for itself. If the partition is mentioned in `/etc/fstab`, it is sufficient to give either the mount point *or* the device file:

```
# mount /dev/sda5                                              One possibility …
# mount /home                                                   … and another
```

`/etc/fstab`

Generally speaking, the `/etc/fstab` file describes the composition of the whole file system structure from various file systems that can be located on different partitions, disks etc. In addition to the device names and corresponding mount points, you can specify various options used to mount the file systems. The allowable options depend on the file system; many options are to be found in `mount`(8).

A typical `/etc/fstab` file could look similar to figure 7.1. The root partition usually occupies the first line. Besides the "normal" file systems, pseudo file systems such as `devpts` or `proc` and the swap areas are mentioned here.

type

The third field describes the type of the file system in question. Entries like `ext3` and `iso9660` speak for themselves (if `mount` cannot decide what to do with the type specification, it tries to delegate the job to a program called `/sbin/mount.`⟨*type*⟩), `swap` refers to swap space (which does not require mounting), and `auto` means that `mount` should try to determine the file system's type.

To guess, `mount` utilises the content of the `/etc/filesystems` file, or, if that file does not exist, the `/proc/filesystems` file. (`/proc/filesystems` is also read if `/etc/filesystems` ends with a line containing just an asterisk.) In any case, `mount` processes only those lines that are not marked `nodev`. For your edification, here is a snippet from a typical `/proc/filesystems` file:

```
nodev   sysfs
nodev   rootfs
<<<<<
nodev   usbfs
        ext3
nodev   nfs
        vfat
        xfs
◁◁◁◁◁
```

The kernel generates `/proc/filesystems` dynamically based on those file systems for which it actually contains drivers. `/etc/filesystems` is useful if you want to specify an order for `mount`'s guesswork that deviates from the one resulting from `/proc/filesystems` (which you cannot influence).

Before `mount` refers to `/etc/filesystems`, it tries its luck with the `libblkid` and `libvolume_id` libraries, both of which are (among other things) able to determine which type of file system exists on a medium. You can experiment with these libraries using the command line programs `blkid` and `vol_id`:

```
# blkid /dev/sdb1
/dev/sdb1: LABEL="TESTBTRFS" UUID="d38d6bd1-66c3-49c6-b272-eabdae▷
 ◁ 877368" UUID_SUB="3c093524-2a83-4af0-8290-c22f2ab44ef3" ▷
 ◁ TYPE="btrfs" PARTLABEL="Linux filesystem" ▷
 ◁ PARTUUID="ade1d2db-7412-4bc1-8eab-e42fdee9882b"
```

options The fourth field contains the options, including:

`defaults` Is not really an option, but merely a place holder for the standard options (see `mount`(8)).

`noauto` Opposite of `auto`, keeps a file system from being mounted automatically when the system is booted.

`user` In principle, only `root` can mount storage devices (normal users may only use the simple `mount` command to display information), unless the `user` option is set. In this case, normal users may say "`mount` ⟨*device*⟩" or "`mount` ⟨*mount point*⟩"; this will mount the named device on the designated mount point. The `user` option will allow the mounting user to unmount the device (`root`, too); there is a similar option `users` that allows any user to unmount the device.

`sync` Write operations are not buffered in RAM but written to the medium directly. The end of the write operation is only signaled to the application program once the data have actually been written to the medium. This is useful for floppies or USB thumb drives, which might otherwise be inadvertently removed from the drive while unwritten data is still buffered in RAM.

`ro` This file system is mounted for reading only, not writing (opposite of `rw`)

`exec` Executable files on this file system may be invoked. The opposite is `noexec`; `exec` is given here because the `user` option implies the `noexec` option (among others).

As you can see in the `/dev/sdb` entry, later options can overwrite earlier ones: `user` implies the `noexec` option, but the `exec` farther on the right of the line overwrites this default.

7.2.3 Labels and UUIDs

We showed you how to mount file systems using device names such as `/dev/hda1`. This has the disadvantage, though, that the correspondence between device files and actual devices is not necessarily fixed: As soon as you remove or repartition a disk or add another, the correspondence may change and you will have to adjust the configuration in `/etc/fstab`. With some device types, such as USB media, you cannot by design rely on anything. This is where labels and UUIDs come in.

label A **label** is a piece of arbitrary text of up to 16 characters that is placed in a file system's super block. If you have forgotten to assign a label when creating the file system, you can add one (or modify an existing one) at any time using `e2label`. The command

```
# e2label /dev/sda3 home
```

(for example) lets you refer to `/dev/sda3` as `LABEL=home`, e. g., using

```
# mount -t ext2 LABEL=home /home
```

The system will then search all available partitions for a file system containing this label.

You can do the same using the `-L` option of `tune2fs`:

```
# tune2fs -L home /dev/sda3
```

The other file systems have their ways and means to set labels, too. With Btrfs, for example, you can either specify one when the file system is generated (option "`-L`") or use

```
# btrfs filesystem label /dev/sdb1 MYLABEL
```

If you have very many disks or computers and labels do not provide the required degree of uniqueness, you can fall back to a "universally unique identifier" or **UUID**. An UUID typically looks like

UUID

```
$ uuidgen
bea6383f-22a7-453f-8ef5-a5b895c8ccb0
```

and is generated automatically and randomly when a file system is created. This ensures that no two file systems share the same UUID. Other than that, UUIDs are used much like labels, except that you now need to use `UUID=bea6383f-22a7-453f-8ef5-a5b895c8ccb0` (Gulp.) You can also set UUIDs by means of `tune2fs`, or create completely new ones using

```
# tune2fs -U random /dev/hda3
```

This should seldom prove necessary, though, for example if you replace a disk or have cloned a file system.

Incidentally, you can determine a file system's UUID using

```
# tune2fs -l /dev/hda2 | grep UUID
Filesystem UUID:            4886d1a2-a40d-4b0e-ae3c-731dd4692a77
```

With other file systems (XFS, Btrfs) you can query a file system's UUID (`blkid` is your friend) but not necessarily change it.

The

```
# lsblk -o +UUID
```

command gives you an overview of all your block devices and their UUIDs.

You can also access swap partitions using labels or UUIDs:

```
# swapon -L swap1
# swapon -U 88e5f06d-66d9-4747-bb32-e159c4b3b247
```

You can find the UUID of a swap partition using blkid or lsblk, or check the /dev/disk/by-uuid directory. If your swap partition does not have a UUID nor a label, you can use mkswap to assign one.

You can also use labels and UUIDs in the /etc/fstab file (one might indeed claim that this is the whole point of the exercise). Simply put

```
LABEL=home
```

or

```
UUID=bea6383f-22a7-453f-8ef5-a5b895c8ccb0
```

into the first field instead of the device name. Of course this also works for swap space.

Exercises

7.6 [!2] Consider the entries in files /etc/fstab and /etc/mtab. How do they differ?

7.3 The dd Command

dd is a command for copying files "by block". It is used with particular preference to create "images", that is to say complete copies of file systems—for example, when preparing for the complete restoration of the system in case of a catastrophic disk failure.

dd (short for "copy and convert"[2]) reads data block by block from an input file and writes it unchanged to an output file. The data's type is of no consequence. Neither does it matter to dd whether the files in question are regular files or device files.

Using dd, you can create a quickly-restorable backup copy of your system partition as follows:

```
# dd if=/dev/sda2 of=/data/sda2.dump
```

This saves the second partition of the first SCSI disk to a file called /data/sda2.dump—this file should of course be located on another disk. If your first disk is damaged, you can easily and very quickly restore the original state after replacing it with an identical (!) drive:

```
# dd if=/data/sda2.dump of=/dev/sda2
```

(If /dev/sda is your system disk, you must of course have booted from a rescue or live system.)

partition table

For this to work, the new disk drive's geometry must match that of the old one. In addition, the new disk drive needs a partition table that is equivalent to the old one. You can save the partition table using dd as well (at least for MBR-partitioned disks):

```
# dd if=/dev/sda of=/media/floppy/mbr_sda.dump bs=512 count=1
```

Used like this, dd does not save all of the hard disk to floppy disk, but writes everything in chunks of 512 bytes (bs=512)—one chunk (count=1), to be exact. In effect, all of the MBR is written to the floppy. This kills two birds with the same stone: the boot loader's stage 1 also ends up back on the hard disk after the MBR is restored:

[2] Seriously! The dd command is inspired by a corresponding command on IBM mainframes (hence the parameter syntax, which according to Unix standards is quite quaint), which was called CC (as in "copy and convert"), but on Unix the cc name was already spoken for by the C compiler.

```
# dd if=/media/floppy/mbr_sda.dump of=/dev/sda
```

You do not need to specify a chunk size here; the file is just written once and is (hopefully) only 512 bytes in size.

Caution: The MBR does not contain partitioning information for logical partitions! IF you use logical partitions, you should use a program like `sfdisk` to save all of the partitioning scheme—see below.

To save partitioning information for GPT-partitioned disks, use, for example, `gdisk` (the `b` command).

`dd` can also be used to make the content of CD-ROMs or DVDs permanently accessible from hard disk. The "`dd if=/dev/cdrom of=/data/cdrom1.iso`" places the content of the CD-ROM on disk. Since the file is an ISO image, hence contains a file system that the Linux kernel can interpret, it can also be mounted. After "`mount -o loop,ro /data/cdrom.iso /mnt`" you can access the image's content. You can of course make this permanent using `/etc/fstab`.

7.4 Disk Quotas

7.4.1 Basics

soft quota

hard quota

Linux makes it possible to limit the maximum number of used inodes or data blocks per user or per group. Two limits are distinguished: The **soft quota** may not be exceeded in the long term, but you can allow users an "overdraft" by setting the **hard quota** to a higher value. Users may then, for a short time, occupy space up to the hard quota, but within a certain period of time they must reduce the used space to below the soft quota again. As soon as the hard quota is reached or the grace period is over, further write operations will fail.

It is sufficient to dip below the soft quota very briefly. After that you will again be able to use space up to the hard quota, and the grace period will start from the beginning.

Quota assignment

Quotas can be assigned per file system. You could, for example, set a limit for the size of the incoming mail box in `/var/mail` without constraining the users' home directories or vice-versa. Linux supports quotas on all common Linux file systems.

As far as quotas are concerned, XFS does its own thing—but the current Linux tools for quota management can also handle XFS. (Should you be forced to deal with a fairly old Linux, it may be possible that you need to use the `xfs_quota` command instead.)

Btrfs has its own infrastructure for quotas, the so-called "quota groups" or "qgroups". In principle, you can set up quotas for individual subvolumes, which will then be managed hierarchically. The file system ensures that a file system never uses more space than the "quota group" of its subvolume, that of the enclosing subvolume, etc. specify. Snapshots do not count as long as they aren't changed, since snapshots do not take up additional space at first. Study `btrfs-qgroup`(8).

7.4.2 User Quotas (`ext` and XFS)

To set up user quotas, you must first install the quota software (included with most distributions). Then you can mark those file systems where quotas should be enforced by including the `usrquota` mount option, either *ad hoc* for a mounted file system as in

```
# mount -o remount,usrquota /home
```

or (preferably) by permanently including the option in the file system's entry in /etc/fstab:

```
/dev/hda5  /home     ext2    defaults,usrquota  0 2
```

The quota database is initialised using

```
# quotacheck -avu
```

(watch for possible warnings), *after* the file system has been mounted using the usrquota option. The quotacheck program creates the aquota.user database file in the partition's root directory, in this case /home/aquota.user. It is advisable to execute quotacheck periodically during normal system operations in order to "clean" the database.

Afterwards you should start the quota system, so that the database created using quotacheck is brought up to date with every file system operation:

```
# quotaon -avu
```

Your distribution will probably contain an init script which will perform this step during system boot. Accordingly, the quota system will be deactivated on shutdown using "quotaoff -auv".

Setting quotas

You can set quotas for various users using the edquota command. edquota starts your favourite editor (according to the EDITOR environment variable) with a "template" where you can fill in the soft and hard quotas for the file systems in question:

```
# edquota -u hugo                                         Quotas for hugo
```

The grace period in days is set using "edquota -t".

You can also refer to the quotas of a "prototype user":

```
# edquota -p tux hugo
```

sets hugo's quotas to those defined for tux. This makes it easy to assign quotas to newly created users without having to enter them manually using edquota.

Querying quotas

Users can query their quota status with the quota command. "quota -q" outputs a brief message mentioning only those file systems whose soft quota has been exceeded. This command is suitable for files such as ~/.profile.

The repquota command generates a tabular overview of the disk usage of various users, together with possibly active quotas:

```
# repquota -a
                    Block limits           File limits
User        used  soft  hard grace  used soft hard grace
root  --  166512     0     0       19562    0    0
tux   --    2304 10000 12000         806 1000 2000
hugo  --    1192  5000  6000         389  500 1000
```

The grace column contains the remaining number of days of the grace period if the soft quota has been exceeded.

7.4.3 Group Quotas (ext and XFS)

group quotas

You can also set up group quotas applying to all members of a group together. For this to become effective, you must limit *all* groups that the users in question are members of; otherwise they could circumvent their quota by changing groups.

The `mount` option for group quotas is `grpquota`, and the database file is called `aquota.group`. To enable group quotas, you must use the aforementioned commands while substituting or augmenting the `-u` option by `-g`. Hence,

```
# quotaon -auvg
```

activates all types of quota (user and group) on every file system, while

```
$ quota -vg
```

displays group quotas for those groups that you are a member of.

Exercises

7.7 [2] Set up disk quotas as described: a file system with user quotas, one with group quotas, and one with both. Watch out for the output of `quotacheck`, `quotaon`, and `quota`.

7.8 [2] Set up quotas for a user and a group (containing that user). How does `quota`'s output change when you reach the soft quota (the hard quota)? Is a warning displayed?

Commands in this Chapter

blkid	Locates and prints block device attributes	blkid(8)	118
dd	*"Copy and convert"*, copies files or file systems block by block and does simple conversions	dd(1)	120
debugfs	File system debugger for fixing badly damaged file systems. For gurus only!	debugfs(8)	108
dumpe2fs	Displays internal management data of the ext2 file system. For gurus only!	dumpe2fs(8)	108
dumpreiserfs	Displays internal management data of the Reiser file system. For gurus only!	dumpreiserfs(8)	111
e2fsck	Checks ext2 and ext3 file systems for consistency	e2fsck(8)	107
e2label	Changes the label on an ext2/3 file system	e2label(8)	118
edquota	Tool for entering and adjusting disk quotas	edquota(8)	122
fsck	Organises file system consistency checks	fsck(8)	101
lsblk	Lists available block devices	lsblk(8)	119
mkdosfs	Creates FAT-formatted file systems	mkfs.vfat(8)	114
mke2fs	Creates ext2 or ext3 file systems	mke2fs(8)	105
mkfs	Manages file system creation	mkfs(8)	100
mkfs.vfat	Creates FAT-formatted file systems	mkfs.vfat(8)	114
mkfs.xfs	Creates XFS-formatted file systems	mkfs.xfs(8)	111
mkreiserfs	Creates Reiser file systems	mkreiserfs(8)	111
mkswap	Initialises a swap partition or file	mkswap(8)	115
mount	Includes a file system in the directory tree	mount(8), mount(2)	116
quota	Reports on a user's quota status	quota(1)	122
reiserfsck	Checks a Reiser file system for consistency	reiserfsck(8)	111
repquota	Summarises filesystem usage and quota usage for many users	repquota(8)	122
resize_reiserfs	Changes the size of a Reiser file system	resize_reiserfs(8)	111
swapoff	Deactivates a swap partition or file	swapoff(8)	115
swapon	Activates a swap partition or file	swapon(8)	115
tune2fs	Adjusts ext2 and ext3 file system parameters	tunefs(8)	108, 119
vol_id	Determines file system types and reads labels and UUIDs	vol_id(8)	118
xfs_mdrestore	Restores an XFS metadata dump to a filesystem image	xfs_mdrestore(8)	112
xfs_metadump	Produces metadata dumps from XFS file systems	xfs_metadump(8)	112

Summary

- After partitioning, a file system must be created on a new partition before it can be used. To do so, Linux provides the mkfs command (with a number of file-system-specific auxiliary tools that do the actual work).
- Improperly unmounted file systems may exhibit inconsistencies. If Linux notes such file systems when it boots, these will be checked automatically and, if possible, repaired. These checks can also be triggered manually using programs such as fsck and e2fsck.
- The mount command serves to integrate file systems into the directory tree.
- With dd, partitions can be backed up at block level.
- To enable disk quotas, you must mount the file systems accordingly, initialise the database, and activate the quota system.
- Quotas are specified using edquota and supervised using quota or repquota.

Bibliography

Quota-Mini-HOWTO03 Ralf van Dooren. “Quota mini-HOWTO”, August 2003.
`http://tldp.org/HOWTO/Quota.html`

8

Booting Linux

Contents

Goals

- Knowing the GRUB Legacy and GRUB 2 boot loaders and how to configure them
- Being able to diagnose and fix system start problems

Prerequisites

- Basic knowledge of the PC startup procedure
- Handling of configuration files

8.1 Fundamentals

When you switch on a Linux computer, an interesting and intricate process takes place during which the computer initialises and tests itself before launching the actual operating system (Linux). In this chapter, we consider this process in some detail and explain how to adapt it to your requirements and to find and repair problems if necessary.

The word "to boot" is short for "to pull oneself up by one's bootstraps". While, as Newton tells us, this is a physical impossibility, it is a good image for what goes on, namely that the computer gets itself started from the most basic beginnings.

Immediately after the computer is switched on, its firmware—depending on the computer's age, either the "basic input/output system" (BIOS) or "unified extensible firmware interface" (UEFI) takes control. What happens next depends on the firmware.

boot sector
master boot record

BIOS startup On BIOS-based systems, the BIOS searches for an operating system on media like CD-ROM or hard disk, depending on the boot order specified in the BIOS setup. On disks (hard or floppy), the first 512 bytes of the boot medium will be read. These contain special information concerning the system start. Generally, this area is called the **boot sector**; a hard disk's boot sector is also called the **master boot record** (MBR).

We already came across the MBR when discussing the eponymous disk partitioning scheme in chapter 6. We're now looking at the part of the MBR that does *not* contain partitioning information.

boot loader

The first 446 bytes of the MBR contain a minimal startup program which in turn is responsible for starting the operating system—the **boot loader**. The rest is occupied by the partition table. 446 bytes are not enough for the complete boot loader, but they suffice for a small program which can fetch the rest of the boot loader from disk using the BIOS. In the space between the MBR and the start of the first partition—at least sector 63, today more likely sector 2048 there is enough room for the rest of the boot loader. (We shall come back to that topic presently.)

GRUB

Modern boot loaders for Linux (in particular, the "Grand Unified Boot loader" or **GRUB**) can read common Linux file systems and are therefore able to find the operating system kernel on a Linux partition, load it into RAM and start it there.

boot manager

GRUB serves not just as a boot loader, but also as a **boot manager**. As such, it can, according to the user's preferences, launch various Linux kernels or even other operating systems.

Bootable CD-ROMs or DVDs play an important role for the installation or update of Linux systems, or as the basis of "live systems" that run directly from read-only media without having to be installed on disk. To boot a Linux computer from CD, you must in the simplest case ensure that the CD-ROM drive is ahead of the firmware's boot order than the hard disk, and start the computer while the desired CD is in the drive.

In the BIOS tradition, booting off CD-ROMs follows different rules than booting off hard disk (or floppy disk). The "El Torito" standard (which specifies these rules) basically defines two approaches: One method is to include an image of a bootable floppy disk on the CD-ROM (it may be as big as 2.88 MiB), which the BIOS finds and boots; the other method is to boot directly off the CD-ROM, which requires a specialised boot loader (such as `ISOLINUX` for Linux).

With suitable hardware and software (usually part of the firmware today), a PC can boot via the network. The kernel, root file system, and everything else can reside on a remote server, and the computer itself can be diskless and hence ear-friendly. The details would be a bit too involved and are irrelevant for LPIC-1 in any case; if necessary, look for keywords such as "PXE" or "Linux Terminal Server Project".

UEFI boot procedure UEFI-based systems do not use boot sectors. Instead, the UEFI firmware itself contains a boot manager which exploits information about the desired operating system which is held in non-volatile RAM (NVRAM). Boot loaders for the different operating systems on the computer are stored as regular files on an "EFI system partition" (ESP), where the firmware can read and start them. The system either finds the name of the desired boot loader in NVRAM, or else falls back to the default name, `/EFI/BOOT/BOOTX64.EFI`. (The `X64` here stands for "64-bit Intel-style PC". Theoretically, UEFI also works for 32-bit systems, but that doesn't mean it is a great idea.) The operating-system specific boot loader then takes care of the rest, as in the BIOS startup procedure.

The ESP must officially contain a FAT32 file system (there are Linux distributions that use FAT16, but that leads to problems with Windows 7, which requires FAT32). A size of 100 MiB is generally sufficient, but some UEFI implementations have trouble with FAT32 ESPs which are smaller than 512 MiB, and the Linux `mkfs` command will default to FAT16 for partitions of up to 520 MiB. With today's prices for hard disks, there is little reason not to play it safe and create an ESP of around 550 MiB.

In principle it is possible to simply write a complete Linux kernel as `BOOTX64.EFI` on the ESP and thus manage without any boot loader at all. PC-based Linux distributions don't usually do this, but this approach is interesting for embedded systems.

Many UEFI-based systems also allow BIOS-style booting from MBR-partitioned disks, i. e., with a boot sector. This is called "compatibility support module" or CSM. Sometimes this method is used automatically if a traditional MBR is found on the first recognised hard disk. This precludes an UEFI boot from an ESP on an MBR-partitioned disk and is not 100% ideologically pure.

UEFI-based systems boot from CD-ROM by looking for a file called `/EFI/BOOT/BOOTX64.EFI`—like they would for disks. (It is feasible to produce CD-ROMs that boot via UEFI on UEFI-based systems and via El Torito on BIOS-based systems.)

UEFI Secure Boot

"UEFI Secure Boot" is supposed to prevent computers being infected with "root kits" that usurp the startup procedure and take over the system before the actual operating system is being started. Here the firmware refuses to start boot loaders that have not been cryptographically signed using an appropriate key. Approved boot loaders, in turn, are responsible for only launching operating system kernels that have been cryptographically signed using an appropriate key, and approved operating system kernels are expected to insist on correct digital signatures for dynamically loadable drivers. The goal is for the system to run only "trusted" software, at least as far as the operating system is concerned.

A side effect is that this way one gets to handicap or exclude potentially undesirable operating systems. In principle, a company like Microsoft could exert pressure on the PC industry to only allow boot loaders and operating systems signed by Microsoft; since various anti-trust agencies would take a dim view to this, it is unlikely that such a step would become part of official company policy. It is more likely that the manufacturers of PC motherboards and UEFI implementations concentrate their testing and debugging efforts on the "boot Windows" application, and that Linux boot loaders will be difficult or impossible to get to run simply due to inadvertent firmware bugs.

Shim

Linux supports UEFI Secure Boot in various ways. There is a boot loader called "Shim" (developed by Matthew Garrett) which a distributor can have signed by Microsoft. UEFI starts Shim and Shim then starts another boot loader or operating system kernel. These can be signed or unsigned; the security envisioned by UEFI Secure Boot is, of course, only obtainable with the signatures. You can install your own keys and then sign your own (self-compiled) kernels.

The details for this would be carrying things too far. Consult the Linup Front training manual *Linux System Customisation*

PreLoader

An alternative to Shim is "PreLoader" (by James Bottomley, distributed by the Linux Foundation). PreLoader is simpler than Shim and makes it possible to accredit a (possibly unsigned) subsequent boot loader with the system, and boot it later without further enquiries.

Hard disks: MBR vs. GPT The question of which partitioning scheme a hard disk is using and the question of whether the computer boots via the BIOS (or CSM) or UEFI really don't have a lot to do with each other. At least with Linux it is perfectly possible to boot a BIOS-based system from a GPT-partitioned disk or a UEFI-based system from an MBR-partitioned disk (the latter possibly via CSM).

To start a BIOS-based system from a GPT-partitioned disk it makes sense to create a "BIOS boot partition" to hold that part of the boot loader that does not fit into the MBR. The alternative—using the empty space between the MBR and the start of the first partition—is not reliable for GPT-partitioned disks, since the GPT partition table takes up at least part of this space and/ or the first partition might start immediately after the GPT partition table. The BIOS boot partition does not need to be huge at all; 1 MiB is probably amply enough.

After the boot loader The boot loader loads the Linux operating system kernel and passes the control to it. With that, it is itself extraneous and can be removed from the system; the firmware, too, will be ignored from now on—the kernel is left to its own devices. In particular, it must be able to access all drivers required to initialise the storage medium containing the root file system, as well as that file system itself (the boot loader used the firmware to access the disk), typically at least a driver for an IDE, SATA, or SCSI controller and the file system in question. These drivers must be compiled into the kernel or—the preferred method today—will be taken from "early userspace", which can be configured without having to recompile the kernel. (As soon as the root file system is available, everything is peachy because all drivers can be read from there.) The boot loader's tasks also include reading the early-userspace data.

The "early userspace" used to be called an "initial RAM disk", because the data was read into memory *en bloc* as a (usually read-only) medium, and treated by the kernel like a block-oriented disk. There used to be special compressed file systems for this application. The method most commonly used today stipulates that the early-userspace data is available as a `cpio` archive which the kernel extracts directly into the disk block cache, as if you had read each file in the archive directly from a (hypothetical) storage medium. This makes it easier to get rid of the early userspace once it is no longer required.

The kernel uses `cpio` instead of `tar` because `cpio` archives in the format used by the kernel are better-standardised and easier to unpack than `tar` archives.

As soon as the "early userspace" is available, a program called `/init` is invoked. This is in charge of the remaining system initialisation, which includes tasks such as the identification of the storage medium that should be made available as the root file system, the loading of any required drivers to access that medium and the file system (these drivers, of course, also come from early userspace), possibly the (rudimentary) configuration of the network in case the root file system resides on a remote file server, and so on. Subsequently, the early userspace puts the desired root file system into place at "/" and transfers control to the actual init program—today most often either System-V init (chapter 9) or systemd (chapter 10), in each case under the name of `/sbin/init`. (You can juse the kernel command line option `init=` to pick a different program.)

If no early userspace exists, the operating system kernel makes the storage medium named on its command line using the `root=` option available as the root file system, and starts the program given by the `init=` option, by default `/sbin/init`.

Exercises

8.1 [2] Whereabouts on an MBR-partitioned hard disk may a boot loader reside? Why?

8.2 GRUB Legacy

8.2.1 GRUB Basics

Many distributions nowadays use GRUB as their standard boot loader. It has various advantages compared to LILO, most notably the fact that it can handle the common Linux file systems. This means that it can read the kernel directly from a file such as `/boot/vmlinuz`, and is thus immune against problems that can develop if you install a new kernel or make other changes to your system. Furthermore, on the whole GRUB is more convenient—for example offering an interactive GRUB shell featuring various commands and thus allowing changes to the boot setup for special purposes or in case of problems.

GRUB shell

The GRUB shell allows access to the file system without using the usual access control mechanism. It should therefore never be made available to unauthorised people, but be protected by a password (on important computers, at least). See also Section 8.2.5.

Right now there are two widespread versions of GRUB: The older version ("GRUB Legacy") is found in older Linux distributions—especially those with an "enterprise" flavour'—, while the newer distributions tend to rely on the more modern version GRUB 2 (section 8.2.4).

The basic approach taken by GRUB Legacy follows the procedure outlined in section 8.1. During a BIOS-based startup, the BIOS finds the first part ("stage 1") of the boot loader in the MBR of the boot disk (all 446 bytes of it). Stage 1 is able to find the next stage based on sector lists stored inside the program (as part of the 446 bytes) and the BIOS disk access functions[1].

The "next stage" is usually stage 1.5, which is stored in the otherwise unused space immediately after the MBR and before the start of the first partition. Stage 1.5 has rudimentary support for Linux file systems and can find GRUB's "stage 2" within the file system (normally below `/boot/grub`). Stage 2 may be anywhere on the disk. It can read file systems, too, and it fetches its configuration file, displays the menu, and finally loads and starts the desired operating system (in the case of Linux, possibly including the "early userspace").

[1] At least as long as the next stage can be found within the first 1024 "cylinders" of the disk. There are historical reasons for this and it can, if necessary, be enforced through appropriate partitioning.

Stage 1 could read stage 2 directly, but this would be subject to the same restrictions as reading stage 1.5 (no file system access and only within the first 1024 cylinders). This is why things aren't usually arranged that way.

GRUB can directly load and start most Unix-like operating systems for x86 computers, including Linux, Minix, NetBSD, GNU Hurd, Solaris, ReactOS, Xen, and VMware ESXi[2]. The relevant standard is called "multiboot". GRUB starts multiboot-incompatible systems (notably Windows) by invoking the boot loader of the operating system in question—a procedure called "chain loading".

To make GRUB Legacy work with GPT-partitioned disks, you need a BIOS boot partition to store its stage 1.5. There is a version of GRUB Legacy that can deal with UEFI systems, but for UEFI boot you are generally better off using a different boot loader.

8.2.2 GRUB Legacy Configuration

`/boot/grub/menu.lst`

The main configuration file for GRUB Legacy is usually stored as `/boot/grub/menu.lst`. It contains basic configuration as well as the settings for the operating systems to be booted. This file might look as follows:

```
default 1
timeout 10

title linux
   kernel (hd0,1)/boot/vmlinuz root=/dev/sda2
   initrd (hd0,1)/boot/initrd
title failsafe
   kernel (hd0,1)/boot/vmlinuz.bak root=/dev/sda2 apm=off acpi=off
   initrd (hd0,1)/initrd.bak
title someothersystem
   root (hd0,2)
   makeactive
   chainloader +1
title floppy
   root (fd0)
   chainloader +1
```

The individual parameters have the following meaning:

`default` Denotes the default system to be booted. Caution: GRUB counts from 0! Thus, by default, the configuration above launches the `failsafe` entry.

`timeout` This is how many seconds the GRUB menu will be displayed before the `default` entry will be booted.

`title` Opens an operating system entry and specifies its name, which will be displayed within the GRUB menu.

`kernel` Specifies the Linux kernel to be booted. `(hd0,1)/boot/vmlinuz`, for example, means that the kernel is to be found in `/boot/vmlinuz` on the first partition of the zeroth hard disk, thus in our example, for `linux`, on `/dev/hda2`. Caution: The zeroth hard disk is the first hard disk in the BIOS boot order! There is no distinction between IDE and SCSI! And: GRUB starts counting at 0 … Incidentally, GRUB takes the exact mapping of the individual drives from the `device.map` file.

After the kernel location, arbitrary kernel parameters can be passed. This includes the `boot=` entry.

[2] The "U" in GRUB must stand for something, after all.

initrd Denotes the location of the cpio archive used for the "early userspace".

root Determines the system partition for foreign operating systems. You can also specify media that only occasionally contain something bootable, such as the floppy disk drive—this will let you boot from floppy even though the floppy disk is disabled in the BIOS boot order.

chainloader +1 Denotes the boot loader to be loaded from the foreign system's system partition. Generally this is the content of that partition's boot loader.

makeactive Marks the specified partition temporarily as "bootable". Some operating systems (not Linux) require this in order to be able to boot off the partition in question. By the way: GRUB supports a few more such directives, for example map, which makes it possible to fool a system into believing it is installed on a different hard disk (than, e. g., the often disdained second disk) than it actually is.

8.2.3 GRUB Legacy Installation

Here "installation" does not refer to the installation of an RPM package but the installation of the GRUB boot sector, or stage 1 (and very likely the stage 1.5). This is very seldom required, for example during the original installation of the system (where the installation procedure of your distribution will do it for you).

The installation is done using the grub command, which invokes the GRUB shell. It is most convenient to use a "batch" file, since otherwise you would have to start from the very beginning after an erroneous input. Some distributions (e. g., those by SUSE/Novell) already come with a suitable file. In this case, the installation procedure might look like

```
# grub --batch --device-map=/boot/grub/device.map < /etc/grub.inst
```

The --device-map option creates a device.map file under the specified name, if none exists already.

/etc/grub.inst

The /etc/grub.inst file could have the following content:

```
root (hd0,1)
setup (hd0)
quit
```

Here, root denotes the partition containing GRUB's "home directory" (usually /boot/grub—the other parts of GRUB necessary for the installation will be looked for in this directory).

⚠ The partition you specify using root here has nothing to do with the partition containing your Linux distribution's root directory, which you specify using root= in your Linux kernels' menu entries. At least not necessarily. See also Section 8.3.

setup installs GRUB on the specified device, here in hd0's MBR. GRUB's setup command is a simplified version of a more general command called install, which should work in most cases.

grub-install

💡 Alternatively, you may use the grub-install script to install the GRUB components. This comes with some distributions.

disk specification

Inside the GRUB shell it is straightforward to figure out how to specify a hard disk in the root or kernel directives. The GRUB shell command find is useful here:

```
# grub
⊲⊲⊲⊲⊲
grub> find /boot/vmlinuz
 (hd0,1)
```

8.2.4 GRUB 2

new implementation

GRUB 2 is a completely new implementation of the boot loader that did not make particular concessions to GRUB-Legacy compatibility. GRUB 2 was officially released in June 2012, even though various distributions used earlier versions by default.

The LPIC-1 certificate requires knowledge of GRUB 2 from version 3.5 of the exam (starting on 2 July 2012).

As before, GRUB 2 consists of several stages that build on each other:

- Stage 1 (`boot.img`) is placed inside the MBR (or a partition's boot sector) on BIOS-based systems. It can read the first sector of stage 1.5 by means of the BIOS, and that in turn will read the remainder of stage 1.5.
- Stage 1.5 (`core.img`) goes either between the MBR and the first partition (on MBR-partitioned disks) or else into the BIOS boot partition (on GPT-partitioned disks). Stage 1.5 consists of a first sector which is tailored to the boot medium (disk, CD-ROM, network, …) as well as a "kernel" that provides rudimentary functionality like device and file access, processing a command line, etc., and an arbitrary list of modules.

  This modular structure makes it easy to adapt stage 1.5 to size restrictions.
- GRUB 2 no longer includes an explicit stage 2; advanced functionality will be provided by modules and loaded on demand by stage 1.5. The modules can be found in `/boot/grub`, and the configuration file in `/boot/grub/grub.cfg`.

On UEFI-based systems, the boot loader sits on the ESP in a file called `EFI/`⟨*operating system*⟩`/grubx64.efi`, where ⟨*operating system*⟩ is something like `debian` or `fedora`. Have a look at the `/boot/efi/EFI` directory on your UEFI"=based Linux system.

Again, the "`x64`" in "`grubx64.efi`" stands for "64-bit PC".

configuration file

grub-mkconfig

update-grub

The configuration file for GRUB 2 looks markedly different from that for GRUB Legacy, and is also rather more complicated (it resembles a bash script more than a GRUB Legacy configuration file). The GRUB 2 authors assume that system managers will not create and maintain this file manually. Instead there is a command called `grub-mkconfig` which can generate a `grub.cfg` file. To do so, it makes use of a set of auxiliary tools (shell scripts) in `/etc/grub.d`, which, e. g., search `/boot` for Linux kernels to add to the GRUB boot menu. (`grub-mkconfig` writes the new configuration file to its standard output; the `update-grub` command calls `grub-mkconfig` and redirects its output to `/boot/grub/grub.cfg`.)

You should therefore not modify `/boot/grub/grub.cfg` directly, since your distribution is likely to invoke `update-grub` after, e. g., installing a kernel update, which would overwrite your changes to `grub.cfg`.

Usually you can, for instance, add more items to the GRUB 2 boot menu by editing the `/etc/grub.d/40_custom` file. `grub-mkconfig` will copy the content of this file verbatim into the `grub.cfg` file. As an alternative, you could add your configuration settings to the `/boot/grub/custom.cfg` file, which will be read by `grub.cfg` if it exists.

For completeness' sake, here is an excerpt from a typical `grub.cfg` file. By analogy to the example in Section 8.2.2, a menu entry to start Linux might look like this for GRUB 2:

```
menuentry 'Linux' --class gnu-linux --class os {
  insmod gzio
  insmod part_msdos
  insmod ext2
  set root='(hd0,msdos2)'
  linux  /boot/vmlinuz root=/dev/hda2
  initrd /boot/initrd.img
}
```

(grub-mkconfig usually produces more complicated stuff.) Do note that the GRUB modules for decompression (gzio), for MS-DOS-like partitioning support (part_msdos) and the ext2 file system must be loaded explicitly. With GRUB 2, partition numbering starts at 1 (it used to be 0 for GRUB Legacy), so (hd0,msdos2) refers to the second MS-DOS partition on the first hard disk. Instead of kernel, linux is used to start a Linux kernel.

8.2.5 Security Advice

boot parameters

password

The GRUB shell offers many features, in particular access to the file system without the root password! Even entering boot parameters may prove dangerous since it is easy to boot Linux directly into a root shell. GRUB makes it possible to close these loopholes by requiring a password.

For GRUB Legacy, the password is set in the menu.lst file. Here, the entry "password --md5 ⟨*encrypted password*⟩" must be added to the global section. You can obtain the encrypted password via the grub-md5-crypt command (or "md5crypt" within the GRUB shell) and then use, e. g., the GUI to "copy and paste" it to the file. Afterwards, the password will need to be input whenever something is changed interactively in the GRUB menu.

You can also prevent particular systems from being booted by adding the lock option to the appropriate specific section within menu.lst. GRUB will query for the password when that system is to be booted. All other systems can still be started without a password.

Exercises

8.2 [2] Which file contains your boot loader's configuration? Create a new entry that will launch another operating system. Make a backup copy of the file first.

8.3 [!3] Prevent a normal user from circumventing init and booting directly into a shell. How do you generate a password request when a particular operating system is to be booted?

8.3 Kernel Parameters

Linux kernel runtime configuration

Linux can accept a command line from the boot loader and evaluate it during the kernel start procedure. The parameters on this command line can configure device drivers and change various kernel options. This mechanism for Linux kernel runtime configuration is particularly helpful with the generic kernels on Linux distribution boot disks, when a system with problematic hardware needs to be booted. To do this, LILO supports the append=…option, while GRUB lets you append parameters to the kernel specification.

Alternatively, you can enter parameters interactively as the system is being booted. You may have to grab GRUB's attention quickly enough (e. g., by pressing a cursor or shift key while the boot menu or splash screen is displayed). Afterwards you can navigate to the desired menu entry and type e. GRUB then presents you with the desired entry, which you can edit to your heart's content before continuing the boot operation.

configuring device drivers

There are various types of parameters. The first group overwrites hardcoded defaults, such as root or rw. Another group of parameters serves to configure device drivers. If one of these parameters occurs on the command line, the initialisation function for the device driver in question is called with the arguments specified there rather than the built-in default values.

Nowadays most Linux distributions use modular kernels that have only very few device drivers built in. Modular device drivers cannot be configured from the kernel command line.

During booting, if there are problems with a device driver that is built into the kernel, you can usually disable this driver by specifying the number 0 as the parameter for the corresponding boot parameter.

general settings

Finally, there are parameters governing general settings. These include, e. g., `init` or `reserve`. We shall be discussing some typical parameters from the multitude of possible settings. Further parameters can be found within the kernel sources' documentation area. Specific details for particular hardware must be researched in the manual or on the Internet.

`ro` This causes the kernel to mount the root partition read-only

`rw` This causes the kernel to mount the root partition with writing enabled, even if the kernel executable or the boot loader configuration file specify otherwise

`init=`⟨*program*⟩ Runs ⟨*program*⟩ (e. g., `/bin/bash`) instead of the customary `/sbin/init`

⟨*runlevel*⟩ Boots into runlevel ⟨*runlevel*⟩, where ⟨*runlevel*⟩ is generally a number between 1 and 5. Otherwise the initial runlevel is taken from `/etc/inittab`. (Irrelevant for computers running systemd.)

`single` Boots to single-user mode.

`maxcpus=`⟨*number*⟩ On a multi-processor (or, nowadays, multi-core) system, use only as many CPUs as specified. This is useful for troubleshooting or performance measurements.

`mem=`⟨*size*⟩ Specifies the amount of memory to be used. On the one hand, this is useful if the kernel cannot recognise the correct size by itself (fairly unlikely these days) or you want to check how the system behaves with little memory. The ⟨*size*⟩ is a number, optionally followed by a unit ("TokenG" for gibibytes, "`M`" for mebibytes, or "`K`" for kibibytes).

> A typical mistake is something like `mem=512`. Linux is thrifty about system resources, but even it can't quite squeeze itself into 512 bytes (!) of RAM.

`panic=`⟨*seconds*⟩ Causes an automatic reboot after ⟨*seconds*⟩ in case of a catastrophic system crash (called a "kernel panic" in the patois, Edsger Dijkstra's dictum, "The use of anthropomorphic terminology when dealing with computing systems is a symptom of professional immaturity", notwithstanding).

`hd`*x*`=noprobe` Causes the kernel to ignore the disk-like device `/dev/hd`*x* (IDE disk, CD-ROM, …) completely. It is not sufficient to disable the device in the BIOS, as Linux will find and access it even so.

`noapic` and similar parameters like `nousb`, `apm=off`, and `acpi=off` tell Linux not to use certain kernel features. These options can help getting Linux to run at all on unusual computers, in order to analyse problems in these areas more thoroughly and sort them out.

A complete list of all parameters available on the kernel command line is given in the file `Documentation/kernel-parameters.txt`, which is part of the Linux source code. (However, before you install kernel sources just to get at this file, you should probably look for it on the Internet.)

Incidentally, if the kernel notices command-line options that do not correspond to kernel parameters, it passes them to the init process as environment variables.

init environment variables

8.4 System Startup Problems

8.4.1 Troubleshooting

Usually things are simple: You switch on the computer, stroll over to the coffee machine (or not—see Section 9.1), and when you come back you are greeted by the graphical login screen. But what to do if things don't work out that way?

The diagnosis of system startup problems sometimes isn't all that easy—all sorts of messages zoom by on the screen or (with some distributions) are not displayed at all, but hidden behind a nice little picture. The system logging service (syslogd) is also started only after a while. Fortunately, though, Linux does not leave you out in the cold if you want to look at the kernel boot messages at leisure.

For logging purposes, the system startup process can be divided into two phases. The "early" phase begins with the first signs of life of the kernel and continues up to the instant where the system logging service is activated. The "late" phase begins just then and finishes in principle when the computer is shut down.

The kernel writes early-phase messages into an internal buffer that can be displayed using the dmesg command. Various distributions arrange for these messages to be passed on to the system logging service as soon as possible so they will show up in the "official" log.

The system logging service, which we are not going to discuss in detail here, runs during the "late" phase. It will be covered in the Linup Front training manual, *Linux Administration II* (and the LPI-102 exam). For now it will be sufficient to know that most distribution place most messages sent to the system logging service into the /var/log/messages file. This is also where messages from the boot process end up if they have been sent after the logging service was started.

On Debian GNU/Linux, /var/log/messages contains only part of the system messages, namely anything that isn't a grave error message. If you would like to see *everything* you must look at /var/log/syslog—this contains all messages except (for privacy reasons) those dealing with authentication. The "early phase" kernel messages, too, incidentally.

Theoretically, messages sent after init was started but before the system logging service was launched might get lost. This is why the system logging service is usually among the first services started after init.

8.4.2 Typical Problems

Here are some of the typical snags you might encounter on booting:

The computer does not budge at all If your computer does nothing at all, it probably suffers from a hardware problem. (If you're diagnosing such a case by telephone, then do ask the obvious questions such as "Is the power cable plugged into the wall socket?"—perhaps the cleaning squad was desperate to find a place to plug in their vacuum cleaner—, and "Is the power switch at the back of the case switched to *On*?". Sometimes the simple things will do.) The same is likely when it just beeps or flashes its LEDs rhythmically but does not appear to actually start booting.

The beeps or flashes can allow the initiated to come up with a rough diagnosis of the problem. Details of hardware troubleshooting, though, are beyond the scope of this manual.

Things go wrong before the boot loader starts The firmware performs various self-tests and outputs error messages to the screen if things are wrong (such as malfunctioning RAM chips). We shall not discuss how to fix these problems. If everything works fine, your computer ought to identify the boot disk and launch the boot loader.

The boot loader does not finish This could be because the operating system cannot find it (e. g., because the drive it resides on does not show up in the firmware boot order) or it is damaged. In the former case you should ensure that your firmware does the Right Thing (not our topic). In the latter case you should receive at least a rudimentary error message, which together with the boot loader's documentation should allow you to come up with an explanation.

GRUB as a civilised piece of software produces clear-text error messages which are explained in more detail in the GRUB info documentation.

The cure for most of the fundamental (as opposed to configuration-related) boot loader problems, if they cannot be obviously reduced to disk or BIOS errors, consist of booting the system from CD-ROM—the distribution's "rescue system" or a "live distribution" such as Knoppix recommend themselves—and to re-install the boot loader.

The same applies to problems like a ruined partition table in the MBR. Should you ever accidentally overwrite your MBR, you can restore a backup (you do have one, don't you?) using `dd` or re-instate the partitioning using `sfdisk` (you do have a printout of your partition table stashed away somewhere, don't you?) and rewrite the boot loader.

In case of the ultimate worst-case partition table scenario, there are programs which will search the whole disk looking for places that look like file system superblocks, and (help) recover the partition scheme that way. We're keeping our fingers crossed on your behalf that you will never need to run such a program.

The kernel doesn't start Once the boot loader has done its thing the kernel should at least start (which generally leads to some activity on the screen). Distribution kernels are generic enough to run on most PCs, but there may still be problems, e. g., if you have a computer with extremely modern hardware which the kernel doesn't yet support (which is fatal if, for example, a driver for the disk controller is missing) or you have messed with the initial RAM disk (Shame, if you didn't know what you were doing!). It may be possible to reconfigure the BIOS (e. g., by switching a SATA disk controller into a "traditional" IDE-compatible mode) or to deactivate certain parts of the kernel (see Section 8.3) in order to get the computer to boot. It makes sense to have another computer around so you can search the Internet for help and explanations.

If you are fooling around with the kernel or want to install a new version of your distribution kernel, do take care to have a known-working kernel around. If you always have a working kernel in your boot loader menu, you can save yourself from the tedious job of slinging CDs about.

Other problems Once the kernel has finished its initialisations, it hands control off to the "init" process. You will find out more about this in Chapter 9. However, you should be out of the woods by then.

8.4.3 Rescue systems and Live Distributions

As a system administrator, you should always keep a "rescue system" for your distribution handy, since usually you need it exactly when you are least in a position to obtain it quickly. (This applies in particular if your Linux machine is your only computer.) A rescue system is a pared-down version of your distribution which you can launch from a CD or DVD (formerly a floppy disk or disks) and which runs in a RAM disk.

Should your Linux distribution not come with a separate rescue system on floppy disk or CD, then get a "live distribution" such as Knoppix. Live distributions are started from CD (or DVD) and work on your computer without needing to be installed first. You can find Knoppix as an ISO image on `http://www.knoppix.de/` or, every so often, as a freebie with computer magazines.

The advantage of rescue systems and live distributions consists in the fact that they work without involving your hard disk. Thus you can do things like `fsck` your root file system, which are forbidden while your system is running from hard disk. Here are a few problems and their solutions:

Hosed the kernel? Boot the rescue system and re-install the corresponding package. In the simplest case, you can enter your installed system's root file from the rescue system like so:

```
# mount -o rw /dev/sda1 /mnt                       Device name may differ
# chroot /mnt
# _                               We are now seeing the installed distribution
```

After this you can activate the network interface or copy a kernel package from a USB key or CD-ROM and install it using the package management tool of your distribution.

Forgot the `root` password? Boot the rescue system and change to the installed distribution as above. Then do

```
# passwd
```

(You could of course fix this problem without a rescue system by restarting your system with "`init=/bin/bash rw`" as a kernel parameter.)

Live distributions such as Knoppix are also useful to check in the computer store whether Linux supports the hardware of the computer you have been drooling over for a while already. If Knoppix recognises a piece of hardware, you can as a rule get it to run with other Linux distributions too. If Knoppix does not recognise a piece of hardware, this may not be a grave problem (there might be a driver for it somewhere on the Internet that Knoppix does not know about) but you will at least be warned.

If there is a matching live version of your distribution—with Ubuntu, for example, the live CD and the installation CD are identical—, things are especially convenient, since the live distribution will typically recognise the same hardware that the installable distribution does.

Commands in this Chapter

`dmesg`	Outputs the content of the kernel message buffer	`dmesg`(8)	137
`grub-md5-crypt`	Determines MD5-encrypted passwords for GRUB Legacy	`grub-md5-crypt`(8)	135

Summary

- A boot loader is a program that can load and start an operating system.
- A boot manager is a boot lader that lets the user pick one of several operating systems or operating system installations.
- GRUB is a powerful boot manager with special properties—such as the possibility of accessing arbitrary files and a built-in command shell.
- The GRUB shell helps to install GRUB as well as to configure individual boot procedures.

9

System-V Init and the Init Process

Contents

Goals

- Understanding the System-V Init infrastructure
- Knowing `/etc/inittab` structure and syntax
- Understanding runlevels and init scripts
- Being able to shut down or restart the system orderly

Prerequisites

- Basic Linux system administration knowledge
- Knowledge of system start procedures (Chapter 8)

9.1 The Init Process

`/sbin/init`

After the firmware, the boot loader, the operating system kernel and (possibly) the early userspace have done their thing, the "init process" takes over the reins. Its job is to finish system initialisation and supervise the ongoing operation of the system. For this, Linux locates and starts a program called `/sbin/init`.

The init process always has process ID 1. If there is an early userspace, it inherits this from the process that was created to run `/init`, and subsequently goes on to replace its program text by that of the init process.

Incidentally, the init process enjoys a special privilege: it is the only process that cannot be aborted using "`kill -9`". (It can decide to shuffle off this mortal coil of its own accord, though.)

If the init process really quits, the kernel keeps running. There are purists who start a program as the init process that will set up packet filtering rules and then exit. Such a computer makes a practically impregnable firewall, but is somewhat inconvenient to reconfigure without a rescue system …

You can tell the Linux kernel to execute a different program as the init process by specifying an option like "`init=/sbin/myinit`" on boot. There are no special properties that this program must have, but you should remember that, if it ever finishes, you do not get another one without a reboot.

9.2 System-V Init

System-V init

Basics The traditional infrastructure that most Linux distributions used to use is called "System-V init" (pronounced "sys-five init"). The "V" is a roman numeral 5, and it takes its name from the fact that it mostly follows the example of Unix System V, where something very similar showed up for the first time. That was during the 1980s.

For some time there was the suspicion that an infrastructure designed approximately 30 years ago was no longer up to today's demands on a Linux computer's init system. (Just as a reminder: When System-V init was new, the typical Unix system was a VAX with 30 serial terminals.) Modern computers must, for example, be able to deal with frequent changes of hardware (cue USB), and that is something that System-V init finds relatively difficult to handle. Hence there were several suggestions for alternatives to System-V init. One of these—systemd by Lennart Poettering and Kay Sievers—seems to have won out and is the current or upcoming standard of practically all Linux distributions of importance (we discuss it in more detail in chapter 10). Another is Upstart by Scott James Remnant (see section 9.3).

runlevels

One of the characteristic features of System-V init are **runlevels**, which describe the system's state and the services that it offers. Furthermore, the init process ensures that users can log in on virtual consoles, directly-connected serial terminals, etc., and manages system access via modems if applicable. All of this is configured by means of the `/etc/inittab` file

`/etc/inittab`

The syntax of `/etc/inittab` (Figure 9.1), like that of many other Linux configuration files, is somewhat idiosyncratic (even if it is really AT&T's fault). All lines that are not either empty or comments— starting with "`#`" as usual—consist of four fields separated by colons:

Label The first field's purpose is to identify the line uniquely. You may pick an arbitrary combination of up to four characters. (Do yourself a favour and stick with letters and digits.) The label is not used for anything else.

```
# Standard runlevel
id:5:initdefault

# First script to be executed
si::bootwait:/etc/init.d/boot

# runlevels
l0:0:wait:/etc/init.d/rc 0
l1:1:wait:/etc/init.d/rc 1
l2:2:wait:/etc/init.d/rc 2
l3:3:wait:/etc/init.d/rc 3
#l4:4:wait:/etc/init.d/rc 4
l5:5:wait:/etc/init.d/rc 5
l6:6:wait:/etc/init.d/rc 6

ls:S:wait:/etc/init.d/rc S
~~:S:respawn:/sbin/sulogin

# Ctrl-Alt-Del
ca::ctrlaltdel:/sbin/shutdown -r -t 4 now

# Terminals
1:2345:respawn:/sbin/mingetty --noclear tty1
2:2345:respawn:/sbin/mingetty tty2
3:2345:respawn:/sbin/mingetty tty3
4:2345:respawn:/sbin/mingetty tty4
5:2345:respawn:/sbin/mingetty tty5
6:2345:respawn:/sbin/mingetty tty6

# Serial terminal
# S0:12345:respawn:/sbin/agetty -L 9600 ttyS0 vt102

# Modem
# mo:235:respawn:/usr/sbin/mgetty -s 38400 modem
```

Figure 9.1: A typical `/etc/inittab` file (excerpt)

This is not 100% true for lines describing terminals, where according to convention the label corresponds to the name of the device file in question, but without the "`tty`" at the beginning, hence `1` for `tty1` or `S0` for `ttyS0`. Nobody knows exactly why.

Runlevels The runlevels this line applies to. We haven't yet explained in detail how runlevels work, so excuse us for the moment for limiting ourselves to telling you that they are usually named with digits and the line in question will be considered in all runlevels whose digit appears in this field.

In addition to the runlevels with digits as names there is one called "`S`". More details follow below.

Action The third field specifies how to handle the line. The most important possibilities include

`respawn` The process described by this line will immediately be started again once it has finished. Typically this is used for terminals which, after the current user is done with their session, should be presented brand-new to the next user.

`wait` The process described by this line is executed once when the system changes to the runlevel in question, and `init` waits for it to finish.

`bootwait` The process described by this line will be executed once during system startup. `init` waits for it to finish. The runlevel field on this line will be ignored.

`initdefault` The runlevel field of this line specifies which runlevel the system shoud try to reach after booting.

With LSB-compliant distributions, this field usually says "`5`" if the system should accept logins on the graphical screen, otherwise "`3`". See below for details.

If this entry (or the whole file `/etc/inittab`) is missing, you will need to state a run level on the console.

`ctrlaltdel` Specifies what the system should do if the init process is being sent a `SIGINT`—which usually happens if anyone presses the Ctrl+Alt+Del combination. Normally this turns out to be some kind of `shutdown` (see Section 9.4).

There are a few other actions. `powerwait`, `powerfail`, `powerokwait`, and `powerfailnow`, for example, are used to interface System-V init with UPSs. The details are in the documentation (`init`(8) and `inittab`(5)).

Command The fourth field describes the command to be executed. It extends to the end of the line and you can put whatever you like.

If you have made changes to `/etc/inittab`, these do not immediately take effect. You must execute the "`telinit q`" command first in order to get `init` to reread the configuration file.

The Boot Script With System-V init, the init process starts a shell script, the boot script, typically `/etc/init.d/boot` (Novell/SUSE), `/etc/rc.d/init.d/boot` (Red Hat), or `/etc/init.d/rcS` (Debian). (The exact name occurs in `/etc/inittab`; look for an entry whose action is `bootwait`.)

The boot script performs tasks such as checking and possibly correcting the file systems listed in `/etc/fstab`, initialising the system name and Linux clock, and other important prerequisites for stable system operation. Next, kernel modules will be loaded if required, file systems mounted and so on. The specific actions and their exact order depend on the Linux distribution in use.

Today, `boot` usually confines itself to executing the files in a directory such as `/etc/init.d/boot.d` (SUSE) in turn. The files are executed in the order of their names. You can put additional files in this directory in order to execute custom code during system initialisation.

Exercises

9.1 [2] Can you find out where your distribution keeps the scripts that the boot script executes?

9.2 [!2] Name a few typical tasks performed by the boot script. In which order should these be executed?

Runlevels After executing the boot script, the init process attempts to place the system in one of the various runlevels. Exactly which one is given by `/etc/inittab` or determined at the system's boot prompt and passed through to `init` by the kernel.

runlevels

The various runlevels and their meaning have by now been standardised across most distributions roughly as follows:

standardised runlevels

1 Single-user mode with no networking

2 Multi-user mode with no network servers

3 Multi-user mode with network servers

4 Unused; may be configured individually if required

5 As runlevel 3, but with GUI login

6 Reboot

0 System halt

The system runs through the `S` (or `s`) runlevel during startup, before it changes over to one out of runlevels 2 to 5. If you put the system into runlevel 1 you will finally end up in runlevel `S`.

When the system is started, the preferred runlevels are 3 or 5—runlevel 5 is typical for workstations running a GUI, while runlevel 3 makes sense for server systems that may not even contain a video interface. In runlevel 3 you can always start a GUI afterwards or redirect graphics output to another computer by logging into your server from that machine over the network.

These predefined runlevels derive from the LSB standard. Not all distributions actually enforce them; Debian GNU/Linux, for example, mostly leaves runlevel assignment to the local administrator.

You may use runlevels 7 to 9 but you will have to configure them yourself.

During system operation, the runlevel can be changed using the `telinit` command. This command can only be executed as `root`: "`telinit 5`" changes immediately to runlevel ⟨*runlevel*⟩. All currently running services that are no longer required in the new runlevel will be stopped, while non-running services that are required in the new runlevel will be started.

`telinit` command

You may use `init` in place of `telinit` (the latter is just a symbolic link to the former, anyway). The program checks its PID when it starts, and if it is not 1, it behaves like `telinit`, else `init`.

The `runlevel` command displays the previous and current runlevel:

`runlevel`

```
# runlevel
N 5
```

Here the system is currently in runlevel 5, which, as the value "N" for the "previous runlevel" suggests, was entered right after the system start. Output such as "5 3" would mean that the last runlevel change consisted of bringing the system from runlevel 5 to runlevel 3.

We have concealed some more runlevels from you, namely the "on-demand runlevels" A, B, and C. You may make entries in /etc/inittab which are meant for any of these three runlevels and use the ondemand action, such as

```
xy:AB:ondemand:…
```

If you say something like

```
# telinit A
```

these entries are executed, but the actual runlevel does not change: If you were in runlevel 3 before the telinit command, you will still be there when it finishes. a, b, and c are synonyms for A, B, and C.

Exercises

9.3 [!2] Display the current runlevel. What exactly is being output? Change to runlevel 2. Check the current runlevel again.

9.4 [2] Try the on-demand runlevels: Add a line to /etc/inittab which applies to, e. g., runlevel A. Make init reread the inittab file. Then enter the »telinit A« command.

Init Scripts The services available in the various runlevels are started and stopped using the scripts in the /etc/init.d (Debian, Ubuntu, SUSE) or /etc/rc.d/init.d (Red Hat) directories. These scripts are executed when changing from one runlevel to another, but may also be invoked manually. You may also add your own scripts. All these scripts are collectively called **init scripts**.

init scripts

The init scripts>parametersinit scripts usually support parameters such asstart, stop, status, restart, or reload, which you can use to start, stop, …, the corresponding services. The "/etc/init.d/network restart" command might conceivably deactivate a system's network cards and restart them with an updated configuration.

runlevel directories

Of course you do not need to start all services manually when the system is started or you want to switch runlevels. For each runlevel *r* there is a rc*r*.d directory in /etc (Debian and Ubuntu), /etc/rc.d (Red Hat), or /etc/init.d (SUSE). The services for each runlevel and the runlevel transitions are defined in terms of these directories, which contain symbolic links to the scripts in the init.d directory. These links are used by a script, typically called /etc/init.d/rc, to start and stop services when a runlevel is entered or exited.

Activating services

This is done according to the names of the links, in order to determine the starting and stopping order of the services. There are dependencies between various services—there would not be much point in starting network services such as the Samba or web servers before the system's basic networking support has been activated. The services for a runlevel are activated by calling all symbolic links in its directory that start with the letter "S", in lexicographical order with the start parameter. Since the link names contain a two-digit number after the "S", you can predetermine the order of invocation by carefully choosing these numbers. Accordingly, to deactivate the services within a runlevel, all the symbolic links starting with the letter "K" are called in lexicographical order with the stop parameter.

If a running service is also supposed to run in the new run level, an extraneous restart can be avoided. Therefore, before invoking a K link, the rc script checks whether there is an S link for the same service in the new runlevel's directory. If so, the stopping and immediate restart are skipped.

Debian GNU/Linux takes a different approach: Whenever a new runlevel *r* is entered, *all* symbolic links in the *new* directory (/etc/rc*r*.d) are executed. Links beginning with a "K" are passed stop and links beginning with a "S" are passed start as the parameter.

Configuring services

To configure services in a runlevel or to create a new runlevel, you can in principle manipulate the symbolic links directly. However, most distributions deprecate this.

The Red Hat distributions use a program called chkconfig to configure runlevels. "chkconfig quota 35", for example, inserts the quota service not in runlevel 35, but runlevels 3 and 5. "chkconfig -l" gives a convenient overview of the configured runlevels.

The SUSE distributions use a program called insserv to order the services in each runlevel. It uses information contained in the init scripts to calculate a sequence for starting and stopping the services in each runlevel that takes the dependencies into account. In addition, YaST2 offers a graphical "runlevel editor", and there is a chkconfig program which however is just a front-end for insserv.

Nor do you have to create links by hand on Debian GNU/Linux—you may use the update-rc.d program. However, manual intervention is still allowed—update-rc.d's purpose is really to allow Debian packages to integrate their init scripts into the boot sequence. With the

```
# update-rc.d mypackage defaults
```

command, the /etc/init.d/mypackage script will be started in runlevels 2, 3, 4, and 5 and stopped in runlevels 0, 1 and 6. You can change this behaviour by means of options. If you do not specify otherwise, update-rc.d uses the sequence number 20 to calculate the position of the service—contrary to SUSE and Red Hat, this is not automated.—The insserv command is available on Debian GNU/Linux as an optional package; if it is installed, it can manage at least those init scripts that do contain the necessary metadata like it would on the SUSE distributions. However, this has not been implemented throughout.

Exercises

9.5 [!2] What do you have to do to make the syslog service reread its configuration?

9.6 [1] How can you conveniently review the current runlevel configuration?

9.7 [!2] Remove the cron service from runlevel 2.

single-user mode

Single-User Mode In **single-user mode** (runlevel S), only the system administrator may work on the system console. There is no way of changing to other virtual consoles. The single-user mode is normally used for administrative work, especially if the file system needs to be repaired or the quota system set up.

You can mount the root file system read-only on booting, by passing the `S` option on the kernel command line. If you boot the system to single-user mode, you can also disable writing to the root file system "on the fly", using the `remount` and `ro` mount options: "`mount -o remount,ro /`" remounts the root partition read-only; "`mount -o remount,rw /`" undoes it again.

To remount a file system "read-only" while the system is running, no process may have opened a file on the file system for writing. This means that all such programs must be terminated using `kill`. These are likely to be daemons such as `syslogd` or `cron`.

It depends on your distribution whether or not you get to leave single-user mode, and how.

To leave single-user mode, Debian GNU/Linux recommends a reboot rather than something like »`telinit 2`«. This is because entering single-user mode kills all processes that are not required in signle-user mode. This removes some essential background processes that were started when the system passed through runlevel `S` during boot, which is why it is unwise to change from runlevel `S` to a multi-user runlevel.

Exercises

9.8 [!1] Put the system into single-user mode (*Hint:* `telinit`). What do you need to do to actually enter single-user mode?

9.9 [1] Convince yourself that you really are the single user on the system while single-user mode is active, and that no background processes are running.

9.3 Upstart

While System-V init traditionally stipulates a "synchronous" approach—the init system changes its state only through explicit user action, and the steps taken during a state change, like init scripts, are performed in sequence—, Upstart uses an "event-based" philosophy. This means that the system is supposed to react to external events (like plugging in an USB device). This happens "asynchronously". Starting and stopping services creates new events, so that—and that is one of the most important differences between System-V init and Upstart—a service can be restarted automatically if it crashes unexpectedly. (System-V init, on the other hand, wouldn't be bothered at all.)

Upstart has been deliberately designed to be compatible with System-V init, at least to a point where init scripts for services can be reused without changes.

Upstart was developed by Scott James Remnant, at the time an employee of Canonical (the company behind Ubuntu) and accordingly debuted in that distributon. Since Ubuntu 6.10 ("Edgy Eft") it is the standard init system on Ubuntu, although it used to be run in a System-V compatible mode at first; since Ubuntu 9.10 ("Karmic Koala") it is running in "native" mode.

It turns out that Ubuntu is currently in the process of switching over to systemd (see chapter 10).

Since version 3.5 of the LPIC-1 certificate exams (as of 2 July 2012) you are expected to know that Upstart exists and what its major properties are. Configuration and operational details are not required.

```
# rsyslog - system logging daemon
#
# rsyslog is an enhanced multi-threaded replacement for the traditional
# syslog daemon, logging messages from applications

description     "system logging daemon"

start on filesystem
stop on runlevel [06]

expect fork
respawn

exec rsyslogd -c4
```

Figure 9.2: Upstart configuration file for job `rsyslog`

Upstart is also purported to accelerate the boot process by being able to initialise servides in parallel. In actual practice this isn't the case, as the limiting factor during booting is, for the most part, the speed with which blocks of data can be moved from disk to RAM. At the Linux Plumbers Conference 2008, Arjan van de Ven and Auke Kok demonstrated that it is possible to boot an Asus EeePC all the way to a usable desktop (i. e., not a Windows-like desktop with a churning hard disk in the background) within 5 seconds. This work was based on System-V init rather than Upstart.

Jobs

Upstart configuration is based on the idea of "Jobs" that take on the role of init scripts (although init scripts, as we mentioned, are also supported). Upstart distinguishes "tasks"—jobs that run for a limited time and then shut themselves down—and "services"—jobs that run permanently "in the background".

Tasks can be long-running, too. The main criterion is that services—think of a mail, database, or web server—do not terminate of their own accord while tasks do.

Jobs are configured using files within the `/etc/init` directory. The names of these files derive from the job name and the "`.conf`" suffix. See figure 9.2 for an example.

One of the main objectives of Upstart is to avoid the large amounts of template-like code typical for most System-V init scripts. Accordingly, the Upstart configuration file confines itself to stating how the service is to be started ("`exec rsyslogd -c4`"). In addition, it specifies that the service is to be restarted in case it crashes ("`respawn`") and how Upstart can find out which process to track ("`expect fork`" says that the `rsyslog` process puts itself into the background by creating a child process and then exiting—Upstart must then watch out for that child process).—Compare this to `/etc/init.d/syslogd` (or similar) on a typical Linux based on System-V init.

While with "classic" System-V init the system administrator assigns a "global" order in which the init scripts for a particular runlevel are to be executed, with Upstart the jobs decide "locally" where they want to place themselves within a network of dependencies. The "`start on` …" and "`stop on` …" lines stipulate events that lead to the job being started or stopped. In our example, `rsyslog` is started as soon as the file system is available, and stopped when the system transitions to the "runlevels" 0 (halt) or 6 (reboot). System-V init's runlevel directories with symbolic links are no longer required.

Upstart supports runlevels mostly for compatibility with Unix tradition and to ease the migration of System-V init based systems to Upstart. They are not required in principle, but at the moment are still necessary to shut down the system (!).

Newer implementations of System-V init also try to provide dependencies between services in the sense that init script X is always executed after init script Y and so on. (This amounts to a scheme for automatic assignment of the priority numbers within the runlevel directories.) This is done using metadata contained in standardised comments at the beginning of the init scripts. The facilities that this approach provides do fall short of those of Upstart, though.

On system boot, Upstart creates the `startup` event as soon as its own initialisation is complete. This makes it possible to execute other jobs. The complete boot sequence derives from the `startup` event and from events being created through the execution of further jobs and expected by others.

For example, on Ubuntu 10.04 the `startup` event invokes the `mountall` task which makes the file systems available. Once that is finished, the `filesystem` event is created (among others), which in turn triggers the start of the `rsyslog` service from Figure 9.2.

With Upstart, the `initctl` command is used to interact with the `init` process:

```
# initctl list                              Which jobs are running now?
alsa-mixer-save stop/waiting
avahi-daemon start/running, process 578
mountall-net stop/waiting
rc stop/waiting
rsyslog start/running, process 549
◁◁◁◁◁
# initctl stop rsyslog                                       Stop a job
rsyslog stop/waiting
# initctl status rsyslog                             What is its status?
rsyslog stop/waiting
# initctl start rsyslog                                   Restart a job
rsyslog start/running, process 2418
# initctl restart rsyslog                                Stop and start
rsyslog start/running, process 2432
```

The “`initctl stop`”, “`initctl start`”, “`initctl status`”, and “`initctl stop`” can be abbreviated to “`stop`”, “`start`”, ….

9.4 Shutting Down the System

A Linux computer should not simply be powered off, as that could lead to data loss—possibly there are data in RAM that ought to be written to disk but are still waiting for the proper moment in time. Besides, there might be users logged in on the machine via the network, and it would be bad form to surprise them with an unscheduled system halt or restart. The same applies to users taking advantage of services that the computer offers on the Net.

It is seldom necessary to shut down a Linux machine that should really run continuously. You can install or remove software with impunity and also reconfigure the system fairly radically without having to restart the operating system. The only cases where this is really necessary include kernel changes (such as security updates) or adding new or replacing defective hardware inside the computer case.

The first case (kernel changes) is being worked on. The `kexec` infrastructure makes it possible to load a second kernel into memory and jump into it directly (without the detour via a system reboot). Thus it is quite possible that in the future you will always be able to run the newest kernel without actually having to reboot your machine.

With the correct kind of (expensive) hardware you can also mostly sort out the second case: Appropriate server systems allow you to swap CPUs, RAM modules, and disks in and out while the computer is running.

There are numerous ways of shutting down or rebooting the system:

on/off switch

- By valiantly pushing the system's on/off switch. If you keep it pressed until the computer is audibly shutting down the system will be switched off. You should only do this in cases of acute panic (fire in the machine hall or a sudden water influx).

shutdown

- Using the shutdown command. This is the cleanest method of shutting down or rebooting.
- For System-V init: The "telinit 0" command can be used to switch to runlevel 0. This is equivalent to a shutdown.
- Using the halt command. This is really a direct order to the kernel to halt the system, but many distributions arrange for halt to call shutdown if the system is not in runlevels 0 or 6 already.

  reboot

  There is a reboot command for reboots, which like halt usually relies on shutdown. (In fact, halt and reboot are really the same program.)

The commands are all restricted to the system administrator.

The key combination Ctrl+Alt+Del may also work if it is configured appropriately in /etc/inittab (see Section 9.1).

Graphical display managers often offer an option to shut down or reboot the system. You may have to configure whether the root password must be entered or not.

Finally, modern PCs may interpret a (short) press on the on/off switch as "Please shut down cleanly" rather than "Please crash right now".

Normally you will be using the second option, the shutdown command. It ensures that all logged-in users are made aware of the impending system halt, prevents new logins, and, according to its option, performs any requisite actions to shut down the system:

```
# shutdown -h +10
```

for example will shut down the system in ten minutes' time. With the -r option, the system will be restarted. With no option, the system will go to single-user mode after the delay has elapsed.

You may also give the time of shutdown/reboot as an absolute time:

```
# shutdown -h 12:00                                          High Noon
```

For shutdown, the now keyword is a synonym of "+0"—immediate action. Do it only if you are sure that nobody else is using the system.

Here is exactly what happens when the shutdown command is given:

broadcast message

1. All users receive a broadcast message saying that the system will be shut down, when, and why.
2. The shutdown command automatically creates the /etc/nologin file, which is checked by login (or, more precisely, the PAM infrastructure); its existence prevents new user logins (except for root).

For consolation, users that the system turns away are being shown the content of the `/etc/nologin` file.

The file is usually removed automatically when the system starts up again.

3. The system changes to runlevel 0 or 6. All services will be terminated by means of their init scripts (more exactly, all services that do not occur in runlevels 0 or 6, which is usually all of them).
4. All still-running processes are first sent `SIGTERM`. They may intercept this signal and clean up after themselves before terminating.
5. Shortly afterwards, all processes that still exist are forcibly terminated by `SIGKILL`.
6. The file systems are unmounted and the swap spaces are deactivated.
7. Finally, all system activities are finished. Then either a warm start is initiated or the computer shut off using APM or ACPI. If that doesn't work, the message "`System halted`" is displayed on the console. At that point you can hit the switch yourself.

You may pass some text to `shutdown` after the shut-down delay time, which is displayed to any logged-in users:

```
# shutdown -h 12:00 '
System halt for hardware upgrade.
Sorry for the inconvenience!
'
```

If you have executed `shutdown` and then change your mind after all, you can cancel a pending shutdown or reboot using

```
# shutdown -c "No shutdown after all"
```

(of course you may compose your own explanatory message).

wall

By the way: The mechanism that `shutdown` uses to notify users of an impending system halt (or similar) is available for your use. The command is called `wall` (short for "write to all"):

```
$ wall "Cake in the break room at 3pm!"
```

will produce a message of the form

```
Broadcast message from hugo@red (pts/1) (Sat Jul 18 00:35:03 2015):

Cake in the break room at 3pm!
```

on the terminals of all logged-in users.

If you send the message as a normal user, it will be received by all users who haven't blocked their terminal for such messages using "`mesg n`". If you want to reach those users, too, you must send the message as `root`.

Even if you're not logged in on a text terminal but are instead using a graphical environment: Today's desktop environments will pick up such messages and show them in an extra window (or something; that will depend on the desktop environment).

If you're `root` and the parameter of `wall` looks like the name of an existing file, that file will be read and its content sent as the message:

```
# echo "Cake in the break room at 3pm!" >cake.txt
# wall cake.txt
```

You don't get to do this as an ordinary user, but you can still pass the message on `wall`'s standard input. (You can do that as `root`, too, of course.) Don't use this for *War and Peace*.

If you're `root`, you can suppress the header line "`Broadcast message` …" using the `-n` option (short for `--nobanner`).

Exercises

9.10 [!2] Shut down your system 15 minutes from now and tell your users that this is simply a test. How do you prevent the actual shutdown (so that it *really* is simply a test)?

What happens if you (as `root`) pass `wall` the name of a non-existent file as its parameter?

9.11 [2] `wall` is really a special case of the `write` command, which you can use to "chat" with other users of the same computer in an unspeakably primitive fashion. Try `write`, in the easiest case between two different users in different windows or consoles. (`write` was a lot more interesting back when one had a VAX with 30 terminals.)

Commands in this Chapter

`chkconfig`	Starts or shuts down system services (SUSE, Red Hat)	`chkconfig`(8)	147
`halt`	Halts the system	`halt`(8)	151
`initctl`	Supervisory tool for Upstart	`initctl`(8)	150
`insserv`	Activates or deactivates init scripts (SUSE)	`insserv`(8)	147
`reboot`	Restarts the computer	`reboot`(8)	151
`runlevel`	Displays the previous and current run level	`runlevel`(8)	145
`shutdown`	Shuts the system down or reboots it, with a delay and warnings for logged-in users	`shutdown`(8)	151
`update-rc.d`	Installs and removes System-V style init script links (Debian)	`update-rc.d`(8)	147

Summary

- After starting, the kernel initialises the system and then hands off control to the `/sbin/init` program as the first userspace process.
- The `init` process controls the system and takes care, in particular, of activating background services and managing terminals, virtual consoles, and modems.
- The system distinguishes various "runlevels" (operating states) that are defined through different sets of running services.
- A single-user mode is available for large or intrusive administrative tasks.
- The `shutdown` command is a convenient way of shutting down or rebooting the system (and it's friendly towards other users, too).
- You can use the `wall` command to send a message to all logged-in users.
- Linux systems seldom need to be rebooted—actually only when a new operating system kernel or new hardware has been installed.

10

Systemd

Contents

Goals

- Understanding the systemd infrastructure
- Knowing the structure of unit files
- Understanding and being able to configure targets

Prerequisites

- Knowledge of Linux system administration
- Knowledge of system start procedures (Chapter 8)
- Knowledge about System-V init (Chapter 9)

10.1 Overview

Systemd, by Lennart Poettering and Kay Sievers, is another alternative to the old-fashioned System-V init system. Like Upstart, systemd transcends the rigid limitations of System-V init, but implements various concepts for the activation and control of services with much greater rigour than Upstart.

Systemd is considered the future standard init system by all mainstream distributions. On many of them—such as Debian, Fedora, RHEL, CentOS, openSUSE, and SLES—it is now provided by default. Even Ubuntu, originally the main instigator of Upstart, has by now declared for systemd.

dependencies

While System-V init and Upstart use explicit dependencies among services—for instance, services using the system log service can only be started once that service is running—, systemd turns the dependencies around. A service requiring the system log service doesn't do this because the log service needs to be running, but because it itself wants to send messages to the system log. This means it must access the communication channel that the system log service provides. Hence it is sufficient if *systemd itself* creates that communication channel and passes it to the system log service once that becomes available—the service wanting to send messages to the log will wait until its messages can actually be accepted. Hence, systemd can in principle create all communication channels first and then start all services simultaneously without regard to any dependencies whatsoever. The dependencies will sort themselves out without any explicit configuration.

This approach also works when the system is running: If a service is accessed that isn't currently running, systemd can start it on demand.

The same approach can in principle also be used for file systems: If a service wants to open a file on a file system that is currently unavailable, the access is suspended until the file system can actually be accessed.

units
targets

Systemd uses "units" as an abstraction for parts of the system to be managed such as services, communication channels, or devices. "Targets" replace SysV init's runlevels and are used to collect related units. For example, there is a target `multiuser.target` that corresponds to the traditional runlevel 3. Targets can depend on the availability of devices—for instance, a `bluetooth.target` could be requested when a USB Bluetooth adapter is plugged in, and it could launch the requisite software. (System-V init starts the Bluetooth software as soon as it is configured, irrespective of whether Bluetooth hardware is actually available.)

In addition, systemd offers more interesting properties that System-V init and Upstart cannot match, including:

- Systemd supports service activation "on demand", not just depending on hardware that is recognised (as in the Bluetooth example above), but also via network connections, D-Bus requests or the availability of certain paths within the file system.
- Systemd allows very fine-grained control of the services it launches, concerning, e. g., the process environment, resource limits, etc. This includes security improvements, e. g., providing only a limited view on the file system for certain services, or providing services with a private `/tmp` directory or networking environment.

  With SysV init this can be handled on a case-by-case basis within the init scripts, but by comparison this is very primitive and tedious.
- Systemd uses the Linux kernel's `cgroups` mechanism to ensure, e. g., that stopping a service actually stops all related processes.
- If desired, systemd handles services' logging output; the services only need to write messages to standard output.

- Systemd makes configuration maintenance easier, by cleanly separating distribution default and local customisations.
- Systemd contains a number of tools in C that handle system initialisation and do approximately what distribution-specific "runlevel S" scripts would otherwise do. Using them can speed up the boot process considerably and also improves cross-distribution standardisation.

Systemd is designed to offer maximum compatibility with System-V init and other "traditions". For instance, it supports the init scripts of System-V init if no native configuration file is available for a service, or it takes the file systems to be mounted on startup from the `/etc/fstab` file.

You can use the `systemctl` command to interact with a running systemd, e. g., to start or stop services explicitly:

```
# systemctl status rsyslog.service
● rsyslog.service - System Logging Service
   Loaded: loaded (/lib/systemd/system/rsyslog.service; enabled)
   Active: active (running) since Do 2015-07-16 15:20:38 CEST;▷
    ◁ 3h 12min ago
     Docs: man:rsyslogd(8)
           http://www.rsyslog.com/doc/
 Main PID: 497 (rsyslogd)
   CGroup: /system.slice/rsyslog.service
           └─497 /usr/sbin/rsyslogd -n
# systemctl stop rsyslog.service
Warning: Stopping rsyslog.service, but it can still be activated by:
  syslog.socket
# systemctl start rsyslog.service
```

Systemd calls such change requests for the system state "jobs", and puts them into a queue.

transactions

Systemd considers status change requests "transactions". If a unit is being started or stopped, it (and any units that depend on it) are put into a temporary transaction. Then systemd checks that the transaction is consistent—in particular, that no circular dependencies exist. If that isn't the case, systemd tries to repair the transaction by removing jobs that are not essential in order to break the cycle(s). Non-essential jobs that would lead to running services being stopped are also removed. Finally, systemd checks whether the jobs within the transaction would conflict with other jobs that are already in the queue, and refuses the transaction if that is the case. Only if the transaction is consistent and the minimisation of its impact on the system is complete, will its jobs be entered into the queue.

Exercises

10.1 [!1] Use "`systemctl status`" to get a picture of the units that are active on your computer. Check the detailed status of some interesting-looking units.

10.2 Unit Files

One of the more important advantages of systemd is that it uses a unified file format for all configuration files—no matter whether they are about services to be started, devices, communication channels, file systems, or other artefacts that systemd manages.

This is in stark contrast to the traditional infrastructure based on System-V init, where almost every functionality is configured in a different way: permanently running background services in `/etc/inittab`, runlevels and their services via init scripts, file systems in `/etc/fstab`, services that are run on demand in `/etc/inetd.conf`, … Every single such file is syntactically different from all others, while with systemd, only the details of the possible (and sensible) configuration settings differ—the basic file format is always the same.

A very important observation is: Unit files are “declarative”. This means that they simply describe what the desired configuration *looks like*—unlike System V init’s init scripts, which contain executable code that tries to *achieve* the desired configuration.

Init scripts usually consider of huge amounts of boilerplate code which depends on the distribution in question, but which you still need to read and understand line-per-line if there is a problem or you want to do something unusual. For somewhat more complex background services, init scripts of a hundred lines or more are not unusual. Unit files for systemd, though, usually get by with a dozen lines or two, and these lines are generally pretty straightforward to understand.

Of course unit files occasionally contain shell commands, for example to explain how a specific service should be started or stopped. These, however, are generally fairly obvious one-liners.

Syntax The basic syntax of unit files is explained in `systemd.unit`(5). You can find an example for a unit file in figure 10.1. A typical characteristic is the subdivision into sections that start with a title in square brackets[1]. All unit files (no matter what they are supposed to do) can include `[Unit]` and `[Install]` sections (see below). Besides, there are sections that are specific to the purpose of the unit.

As usual, blank lines and comment lines are ignored. Comment lines can start with a `#` or `;`. Over-long lines can be wrapped with a `\` at the end of the line, which will be replaced by a space character when the file is read. Uppercase and lowercase letters are important!

options

Lines which are not section headers, empty lines, nor comment lines contain “options” according to a “⟨*name*⟩ `=` ⟨*value*⟩” pattern. Various options may occur several times, and systemd’s handling of that depends on the option: Multiple options often form a list; if you specify an empty value, all earlier settings will be ignored. (If that is the case, the documentation will say so.)

Options that are not listed in the documentation will be flagged with a warning by systemd and otherwise ignored. If a section or option name starts with “`X-`”, it is ignored completely (options in an “`X-`” section do not need their own “`X-`” prefix).

Yes/no settings in unit files can be given in a variety of ways. `1`, `true`, `yes`, and `on` stand for “yes”, `0`, `false`, `no`, and `off` for “no”.

Times can also be specified in various ways. Simple integers will be interpreted as seconds[2]. If you append a unit, then that unit applies (allowed units include `us`, `ms`, `s`, `min`, `h`, `d`, `w` in increasing sequence from microseconds to weeks—see `systemd.time`(7)). You can concatenate several time specifications with units (as in “`10min 30s`”), and these times will be added (here, 630 seconds).

[1] The syntax is inspired by the `.desktop` files of the “XDG Desktop Entry Specification” [XDG-DS14], which in turn have been inspired by the INI files of Microsoft Windows.

[2] Most of the time, anyway—there are (documented) exceptions.

```
#  This file is part of systemd.
#
#  systemd is free software; you can redistribute it and/or modify
#  it under the terms of the GNU Lesser General Public License as
#  published by the Free Software Foundation; either version 2.1
#  of the License, or (at your option) any later version.

[Unit]
Description=Console Getty
Documentation=man:agetty(8)
After=systemd-user-sessions.service plymouth-quit-wait.service
After=rc-local.service
Before=getty.target

[Service]
ExecStart=-/sbin/agetty --noclear --keep-baud console ▷
 ◁ 115200,38400,9600 $TERM
Type=idle
Restart=always
RestartSec=0
UtmpIdentifier=cons
TTYPath=/dev/console
TTYReset=yes
TTYVHangup=yes
KillMode=process
IgnoreSIGPIPE=no
SendSIGHUP=yes

[Install]
WantedBy=getty.target
```

Figure 10.1: A systemd unit file: `console-getty.service`

Searching and finding settings Systemd tries to locate unit files along a list of directories that is hard-coded in the program. Directories nearer the front of the list have precedence over directories nearer the end.

The details are system-dependent, at least to a certain degree. The usual list is normally something like

`/etc/systemd/system`	*Local configuration*
`/run/systemd/system`	*Dynamically generated unit files*
`/lib/systemd/system`	*Unit files for distribution packages*

Local customisation

Systemd offers various clever methods for customising settings without having to change the unit files generally provided by your distribution—which would be inconvenient if the distribution updates the unit files. Imagine you want to change a few settings in the `example.service` file:

- You can copy the distribution's `example.service` file from `/lib/systemd/system` to `/etc/systemd/system` and make any desired customisations. The unit file furnished by the distribution will then not be considered at all.

- You can create a directory `/etc/systemd/system/example.service.d` containing a file—for example, `local.conf`. The settings in that file override settings with the same name in `/lib/systemd/system/example.service`, but any settings not mentioned in `local.conf` stay intact.

  Take care to include any required section titles in `local.conf`, such that the options can be identified correctly.

  Nobody keeps you from putting several files into `/etc/systemd/system/example.service.d`. The only prerequisite is that file names must end in `.conf`. Systemd makes no stipulation about the order in which these files are read—it is best to ensure that every option occurs in just one single file.

Template unit files Sometimes several services can use the same or a very similar unit file. In this case it is convenient not to have to maintain several copies of the same unit file. Consider, for example, the terminal definition lines in `/etc/inittab`—it would be good not to have to have one unit file per terminal.

Instantiation

Systemd supports this by means of unit files with names like `example@.service`. You could, for example, have a file called `getty@.service` and then configure a virtual console on `/dev/tty2` simply by creating a symbolic link from `getty@tty2.service` to `getty@.service`. When this console is to be activated, systemd reads the `getty@.service` file and replaces the `%I` key, wherever it finds it, by whatever comes between `@` and `.` in the name of the unit file, i. e., `tty2`. The result of that replacement is then put into force as the configuration.

In fact, systemd replaces not just `%I` but also some other sequences (and that not just in template unit files). The details may be found in `systemd.unit`(5), in the "Specifiers" section.

Basic settings All unit files may contain the `[Unit]` and `[Install]` sections. The former contains general information about the unit, the latter provides details for its installation (for example, to introduce explicit dependencies—which we shall discuss later).

Here are some of the more important options from the `[Unit]` section (the complete list is in `systemd.unit`(5)):

`Description` A description of the unit (as free text). Will be used to make user interfaces more friendly.

`Documentation` A space-separated list of URLs containing documentation for the unit. The allowed protocol schemes include `http:`, `https:`, `file:`, `info:`, and `man:` (the latter three refer to locally-installed documentation). An empty value clears the list.

`OnFailure` A space-separated list of other units which will be activated if this unit transitions into the `failed` state.

`SourcePath` The path name of a configuration file from which this unit file has been generated. This is useful for tools that create unit files for systemd from external configuration files.

`ConditionPathExists` Checks whether there is a file (or directory) under the given absolute path name. If not, the unit will be classed as `failed`. If there is a `!` in front of the path name, then a file (or directory) with that name must *not* exist. (There are loads of other "`Condition`…" tests—for example, you can have the execution of units depend on whether the system has a particular computer architecture, is running in a virtual environment, is running on AC or battery power or on a computer with a particular name, and so on. Read up in `systemd.unit`(5).)

Exercises

10.2 [!2] Browse the unit files of your system under `/lib/systemd/system` (or `/usr/lib/systemd/system`, depending on the distribution). How many different `Condition`… options can you find?

10.3 Unit Types

Systemd supports a wide variety of "units", or system components that it can manage. These are easy to tell apart by the extensions of the names of the corresponding unit files. As mentioned in Section 10.2, all units share the same basic file format. Here is a list of the most important unit types:

`.service` A process on the computer that is executed and managed by systemd. This includes both background services that stay active for a long time (possibly until the system is shut down), and processes that are only executed once (for example when the system is booting).

When a service is invoked by name (such as `example`) but no corresponding unit file (here, `example.service`) can be found, systemd looks for a System-V init script with the same name and generates a service unit for that on the fly. (The compatibility is fairly wide-ranging but not 100% complete.)

`.socket` A TCP/IP or local socket, i. e., a communication end point that client programs can use to contact a server. Systemd uses socket units to activate background services on demand.

Socket units always come with a corresponding service unit which will be started when systemd notes activity on the socket in question.

`.mount` A "mount point" on the system, i. e., a directory where a file system should be mounted.

The names of these units are derived from the path name by means of replacing all slashes ("`/`") with hyphens ("`-`") and all other non-alphanumeric (as per ASCII) characters with a hexadecimal replacement such as `\x2d` ("`.`" is only converted if it is the first character of a path name). The name of the root directory ("`/`") becomes

“-”, but slashes at the start or end of all other names are removed. The directory name /home/lost+found, for instance, becomes home-lost\ textbackslash x2bfound.

You can try this replacement using the “systemd-escape -p” command:

```
$ systemd-escape -p /home/lost+found
-home-lost\x2bfound
$ systemd-escape -pu home-lost\\x2bfound
/home/lost+found
```

The “-p” option marks the parameter as a path name. The “-u” option undoes the replacement.

.automount Declares that a mount point should be mounted on demand (instead of prophylactically when the system is booted). The names of these units result from the same path name transformation. The details of mounting must be described by a corresponding mount unit.

.swap Describes swap space on the system. The names of these units result from the path name transformation applied to the device or file name in question.

.target A “target”, or synchronisation point for other units during system boot or when transitioning into other system states. Vaguely similar to System-V init’s runlevels. See section 10.5.

.path Observes a file or a directory and starts another unit (by default, a service unit of the same name) when, e. g., changes to the file have been noticed or a file has been added to an otherwise empty directory.

.timer Starts another unit (by default, a service unit of the same name) at a certain point in time or repeatedly at certain intervals. This makes systemd a replacement for cron and at.

(There are a few other unit types, but to explain all of them here would be carrying things too far.)

Exercises

10.3 [!2] Look for examples for all of these units on your system. Examine the unit files. If necessary, consult the manpages for the various types.

10.4 Dependencies

As we have mentioned before, systemd can mostly get by without explicit dependencies because it is able to exploit implicit dependencies (e. g., on communication channels). Even so, it is sometimes necessary to specify explicit dependencies. Various options in the [Unit] section of a service file (e.g., example.service) allow you to do just that. For example:

Requires Specifies a list of other units. If the current unit is activated, the listed units are also activated. If one of the listed units is deactivated or its activation fails, then the current unit will also be deactivated. (In other words, the current unit “depends on the listed units”.)

The Requires dependencies have nothing to do with the *order* in which the units are started or stopped—you will have to configure that separately with After or Before. If no explicit order has been specified, systemd will start all units at the same time.

You can specify these dependencies without changing the unit file, by creating a directory called `/etc/systemd/system/example.service.requires` and adding symbolic links to the desired unit files to it. A directory like

```
# ls -l /etc/systemd/system/example.service.requires
lrwxrwxrwx 1 root root 34 Jul 17 15:56 network.target -> ▷
 ◁ /lib/systemd/system/network.target
lrwxrwxrwx 1 root root 34 Jul 17 15:57 syslog.service -> ▷
 ◁ /lib/systemd/system/syslog.service
```

corresponds to the setting

```
[Unit]
Requires = network.target syslog.service
```

in `example.service`.

`Wants` A weaker form of `Requires`. This means that the listed units will be started together with the current unit, but if their activation fails this has no influence on the process as a whole. This is the recommended method of making the start of one unit depend on the start of another one.

Here, too, you can specify the dependencies “externally” by creating a directory called `example.service.wants`.

`Conflicts` The reverse of `Requires`—the units listed here will be stopped when the current unit is started, and vice versa.

Like `Requires`, `Conflicts` makes no stipulation to the order in which units are started or stopped.

If a unit *U* conflicts with another unit *V* and both are to be started at the same time, this operation fails if both units are an essential part of the operation. If one (or both) units are not essential parts of the operation, the operation is modified: If only one unit is not mandatory, that one will not be started, if both are not mandatory, the one mentioned in `Conflicts` will be started and the one whose unit file contains the `Conflicts` option will be stopped.

`Before` (and `After`) These lists of units determine the starting order. If `example.service` contains the “`Before=example2.service`” option and both units are being started, the start of `example2.service` will be delayed until `example.service` has been started. `After` is the converse of `Before`, i. e., if `example2.service` contains the option “`After=example.service`” and both units are being started, the same effect results—`example2.service` will be delayed.

Notably, this has nothing to do with the dependencies in `Requires` and `Conflicts`. It is common, for example, to list units in both `Requires` and `After`. This means that the listed unit will be started before the one whose unit file contains these settings.

When deactivating units, the reverse order is observed. If a unit with a `Before` or `After` dependency on another unit is deactivated, while the other is being started, then the deactivation takes place before the activation no matter in which direction the dependency is pointing. If there is no `Before` or `After` dependency between two units, they will be started or stopped simultaneously.

Table 10.1: Common targets for systemd (selection)

Target	Description
basic.target	Basic system startup is finished (file systems, swap space, sockets, timers etc.)
ctrl-alt-del.target	Is executed when Ctrl+Alt+Del was pressed. Often the same as reboot.target.
default.target	Target which systemd attempts to reach on system startup. Usually either multi-user.target or graphical.target.
emergency.target	Starts a shell on the system console. For emergencies. Is usually activated by means of the "systemd.unit=emergency.target" on the kernel command line.
getty.target	Activates the statically-defined getty instances (for terminals). Corresponds to the getty lines in /etc/inittab on System-V init.
graphical.target	Establishes a graphical login prompt. Depends on multi-user.target.
halt.target	Stops the system (without powering it down).
multi-user.target	Establishes a multi-user system without a graphical login prompt. Used by graphical.target.
network-online.target	Serves as a dependency for units that require network services (*not* ones that provide network services), such as mount units for remote file systems. How exactly the system determines whether the network is available depends on the method for network configuration.
poweroff.target	Stops the system and powers it down.
reboot.target	Restarts the system.
rescue.target	Performs basic system initialisation and then starts a shell.

Exercises

10.4 [!1] What advantage do we expect from being able to configure dependencies via symbolic links in directories like example.service.requires instead of the example.service unit file?

10.5 [2] Check your system configuration for examples of Requires, Wants and Conflicts dependencies, with or without corresponding Before and After dependencies.

10.5 Targets

Targets in systemd are roughly similar to runlevels in System-V init: a possibility of conveniently describing a set of services. While System-V init allows only a relatively small number of runlevels and their configuration is fairly involved, systemd makes it possible to define various targets very easily.

Unit files for targets have no special options (the standard set of options for [Unit] and [Install] should be enough). Targets are only used to aggregate other units via dependencies or create standardised names for synchronisation points in dependencies (local-fs.target, for example, can be used to start units depending on local file systems only once these are actually available). An overview of the most important targets is in Table 10.1.

In the interest of backwards compatibility to System-V init, systemd defines a number of targets that correspond to the classical runlevels. Consider table 10.2.

Table 10.2: Compatibility targets for System-V init

Ziele	Äquivalent
`runlevel0.target`	`poweroff.target`
`runlevel1.target`	`rescue.target`
`runlevel2.target`	`multi-user.target` (recommended)
`runlevel3.target`	`graphical.target` (recommended)
`runlevel4.target`	`graphical.target` (recommended)
`runlevel5.target`	`graphical.target` (recommended)
`runlevel6.target`	`reboot.target`

default target

You can set the default target which systemd will attempt to reach on system boot by creating a symbolic link from `/etc/systemd/system/default.target` to the desired target's unit file:

```
# cd /etc/systemd/system
# ln -sf /lib/systemd/system/multi-user.target default.target
```

(This is the moral equivalent to the `initdefault` line in the `/etc/inittab` file of System-V init.) A more convenient method is the "`systemctl set-default`" command:

```
# systemctl get-default
multi-user.target
# systemctl set-default graphical
Removed symlink /etc/systemd/system/default.target.
Created symlink from /etc/systemd/system/default.target to ▷
 ◁ /lib/systemd/system/graphical.target.
# systemctl get-default
graphical.target
```

(As you can see, that doesn't do anything other than tweak the symbolic link, either.)

Activate specific target

To activate a specific target (like changing to a specific runlevel on System-V init), use the "`systemctl isolate`" command:

```
# systemctl isolate multi-user
```

("File*.target" will be appended to the parameter if necessary). This command starts all units that the target depends upon and stops all other units.

"`systemctl isolate`" works only for units in whose `[Unit]` sections the "`AllowIsolate`" option is switched on.

To stop the system or to change to the rescue mode (System-V init aficionados would call this "single-user mode") there are the shortcuts

```
# systemctl rescue
# systemctl halt
# systemctl poweroff                          Like halt, but with power-down
# systemctl reboot
```

These commands correspond roughly to their equivalents using "`systemctl isolate`", but also output a warning to logged-in users. You can (and should!) of course keep using the `shutdown` command.

You can return to the default operating state using

```
# systemctl default
```

Exercises

10.6 [!2] Which other services does the `multi-user.target` depend on? Do these units depend on other units in turn?

10.7 [2] Use “`systemctl isolate`” to change your system to the rescue (single-user) mode, and “`systemctl default`” to come back to the standard mode. (*Hint:* Do this from a text console.)

10.8 [2] Restart your system using “`systemctl reboot`” and then once again with `shutdown`. Consider the difference.

10.6 The `systemctl` Command

The `systemctl` command is used to control systemd. We have already seen a few applications, and here is a more systematic list. This is, however, still only a small excerpt of the complete description.

The general structure of `systemctl` invocations is

```
# systemctl ⟨subcommand⟩ ⟨parameters⟩ …
```

`systemctl` supports a fairly large zoo of subcommands. The allowable parameters (and options) depend on the subcommand in question.

unit names as parameters

Often unit names are expected as parameters. These can be specified either with a file name extension (like, e. g., `example.service`) or without (`example`). In the latter case, systemd appends an extension that it considers appropriate—with the `start` command, for example, “`.service`”, with the `isolate` command on the other hand, “`.target`”.

Commands for units The following commands deal with units and their management:

`list-units` Displays the units systemd knows about. You may specify a unit type (`service`, `socket`, …) or a comma-separated list of unit types using the `-t` option, in order to confine the output to units of the type(s) in question. You can also pass a shell search pattern in order to look for specific units:

```
# systemctl list-units "ssh*"
UNIT        LOAD   ACTIVE SUB     DESCRIPTION
ssh.service loaded active running OpenBSD Secure Shell server

LOAD   = Reflects whether the unit definition was properly loaded.
ACTIVE = The high-level unit activation state, i.e. generalization
         of SUB.
SUB    = The low-level unit activation state, values depend on
         unit type.

1 loaded units listed. Pass --all to see loaded but inactive units,
too. To show all installed unit files use 'systemctl
list-unit-files'.
```

As usual, quotes are a great idea here, so the shell will not vandalise the search patterns that are meant for systemd.

`start` Starts one or more units mentioned as parameters.

> You can use shell search patterns here, too. The search patterns only work for units that systemd knows about; inactive units that are not in a failed state will not be searched, nor will units instantiated from templates whose exact names are not known before the instantiation. You should not overtax the search patterns.

stop Stops one or more units mentioned as parameters (again with search patterns).

reload Reloads the configuration for the units mentioned as parameters (if the programs underlying these units go along). Search patterns are allowed.

> This concerns the configuration of the background services themselves, not the configuration of the services from systemd's point of view. If you want systemd to reload its own configuration with respect to the background services, you must use the "`systemctl daemon-reload`" command.

> What exactly happens on a "`systemctl reload`" depends on the background service in question (it usually involves a `SIGHUP`). You can configure this in the unit file for the service.

restart Restarts the units mentioned as parameters (search patterns are allowed). If a unit doesn't yet run, it is simply started.

try-restart Like `restart`, but units that don't run are not started.

reload-or-restart (and `reload-or-try-restart`) Reloads the configuration of the named units (as per `reload`), if the units allow this, or restarts them (as per `restart` or `try-restart`) if they don't.

> Instead of `reload-or-try-restart` you can say `force-reload` for convenience (this is at least somewhat shorter).

isolate The unit in question is started (including its dependencies) and all other units are stopped. Corresponds to a runlevel change on System-V init.

kill Sends a signal to one or several processes of the unit. You can use the `--kill-who` option to specify which process is targeted. (The options include `main`, `control`, and `all`—the latter is the default—, and `main` and `control` are explained in more detail in `systemctl`(1).) Using the `--signal` option (`-s` for short) you can determine which signal is to be sent.

status Displays the current status of the named unit(s), followed by its most recent log entries. If you do not name any units, you will get an overview of all units (possibly restricted to particular types using the `-t` option).

> The log excerpt is usually abridged to 10 lines, and long lines will be shortened. You can change this using the `--lines` (`-n`) and `--full` (`-l`) options.

> "`status`" is used for human consumption. If you want output that is easy to process by other programs, use "`systemctl show`".

cat Displays the configuration file(s) for one or more units (including fragments in configuration directories specifically for that unit). Comments with the file names in question are inserted for clarity.

help Displays documentation (such as man pages) for the unit(s) in question: For example,

```
$ systemctl help syslog
```

invokes the manual page for the system log service, regardless of which program actually provides the log service.

With most distributions, commands like

```
# service example start
# service example stop
# service example restart
# service example reload
```

work independently of whether the system uses systemd or System-V init.

In the next section, there are a few commands that deal with installing and deinstalling units.

Other commands Here are a few commands that do not specifically deal with particular units (or groups of units).

`daemon-reload` This command causes systemd to reload its configuration. This includes regenerating unit files that have been created at runtime from other configuration files on the system, and reconstructing the dependency tree.

Communication channels that systemd manages on behalf of background services will persist across the reload.

`daemon-reexec` Restarts the `systemd` program. This saves systemd's internal state and restores it later.

This command is mostly useful if a new version of `systemd` has been installed (or for debugging `systemd`. Here, too, communication channels that systemd manages on behalf of background services will persist across the reload.

`is-system-running` Outputs the current state of the system. Possible answers include:

`initializing` The system is in the early boot stage (the `basic.target`, `rescue.target`, or `emergency.target` targets have not yet been reached).

`starting` The system is in the late boot stage (there are still jobs in the queue).

`running` The system is running normally.

`degraded` The system is running normally, but one or more units are in a failed state.

`maintenance` One of the `rescue.target` or `emergency.target` targets are active.

`stopping` The system is being shut down.

Exercises

10.9 [!2] Use `systemctl` to stop, start, and restart a suitably innocuous service (such as `cups.service`) and to reload its configuration.

10.10 [2] The `runlevel` command of System-V init outputs the system's current runlevel. What would be an approximate equivalent for systemd?

10.11 [1] What is the advantage of

```
# systemctl kill example.service
```

versus

```
# killall example
```

(or "`pkill example`")?

10.7 Installing Units

To make a new background service available using systemd, you need a unit file, for example `example.service`. (Thanks to backwards compatibility, a System-V init script would also do, but we won't go there just now.) You need to place this in a suitable file (we recommend `/etc/systemd/system`. Next, it should appear when you invoke "`systemctl list-unit-files`":

```
# systemctl list-unit-files
UNIT FILE                             STATE
proc-sys-fs-binfmt_misc.automount     static
org.freedesktop.hostname1.busname     static
org.freedesktop.locale1.busname       static
◁◁◁◁◁
example.service                       disabled
◁◁◁◁◁
```

The `disabled` state means that the unit is available in principle, but is not being started automatically.

Activating units

You can "activate" the unit, or mark it to be started when needed (e. g., during system startup or if a certain communication channel is being accessed), by issuing the "`systemctl enable`" command:

```
# systemctl enable example
Created symlink from /etc/systemd/system/multi-user.target.wants/▷
 ◁example.service to /etc/systemd/system/example.service.
```

The command output tells you what happens here: A symbolic link to the service's unit file from the `/etc/systemd/system/multi-user.target.wants` directory ensures that the unit will be started as a dependency of the `multi-user.target`.

You may ask yourself how systemd knows that the `example` unit should be integrated in the `multi-user.target` (and not some other target). The answer to that is: The `example.service` file has a section saying

```
[Install]
WantedBy=multi-user.target
```

After an `enable`, systemd does the equivalent of a "`systemctl daemon-reload`". However, no units will be started (or stopped).

You could just as well create the symbolic links by hand. You would, however, have to take care of the "`systemctl daemon-reload`" yourself, too.

If you want the unit to be started immediately, you can either give the

```
# systemctl start example
```

command immediately afterwards, or you invoke "`systemctl enable`" with the `--now` option.

You can start a unit directly (using "`systemctl start`") without first activating it with "`systemctl enable`". The former actually starts the service, while the latter only arranges for it to be started at an appropriate moment (e. g., when the system is booted, or a specific piece of hardware is connected).

You can deactivate a unit again with "`systemctl disable`". As with `enable`, systemd does an implicit `daemon-reload`.

Here, too, the unit will not be stopped if it is currently running. (You are just preventing it from being activated later on.) Use the `--now` option or an explicit “`systemctl stop`”.

The “`systemctl reenable`” command is equivalent to a “`systemctl disable`” immediately followed by a “`systemctl enable`” for the units in question. This lets you do a “factory reset” of units.

Masking a unit

The “`systemctl mask`” command lets you “mask” a unit. This means to block it completely. This will not only prevent it from starting automatically, but will also keep it from being started by hand. “`systemctl unmask`” reverts that operation.

Systemd implements this by linking the name of the unit file in `/etc/systemd/system` symbolically to `/dev/null`. Thus, eponymous files in directories that systemd considers later (like `/lib/systemd/system`) will be completely ignored.

Exercises

10.12 [!2] What happens if you execute “`systemctl disable cups`”? (Watch the commands being output.) Reactivate the service again.

10.13 [2] How can you “mask” units whose unit files are in `/etc/systemd/system`?

Commands in this Chapter

`systemctl`	Main control utility for systemd	`systemctl`(1)	157, 166

Summary

- Systemd is a modern alternative to System-V init.
- “Units” are system components managed by systemd. They are configured using unit files.
- Unit files bear a vague resemblance to Microsoft Windows INI files.
- Systemd supports flexible mechanisms for local configuration and the automatic creation of similar unit files from “templates”.
- Systemd lets you manage a multitude of different units—services, mount points, timers, …
- Dependencies between units can be expressed in various ways.
- “Targets” are units that vaguely correspond to System-V init’s runlevels. They are used to group related services and for synchronisation.
- You can use the `systemctl` command to control systemd.
- Systemd contains powerful tools to install and deinstall units.

Bibliography

systemd “systemd System and Service Manager”.
`http://www.freedesktop.org/wiki/Software/systemd/`

XDG-DS14 Preston Brown, Jonathan Blandford, Owen Taylor, et al. “Desktop Entry Specification”, April 2014.
`http://standards.freedesktop.org/desktop-entry-spec/latest/`

11

Dynamic (AKA Shared) Libraries

Contents

Goals

- Being able to identify shared libraries
- Knowing where shared libraries are usually stored
- Being able to manage shared libraries

Prerequisites

- Solid knowledge of the Linux command line
- Programming experience is useful but not essential

11.1 Compiling and Installing Software

An important principle of programming is not to reinvent the wheel continually. Most programs share a lot of code—from basic functions like handling files and strings to more specialised stuff like dealing with date or time data. Neither does it make sense to write this code from scratch over and over again, nor is it reasonable to integrate it bodily into every program. The former because it is a lot of tedious work and the same mistakes would have to be corrected again and again; the latter because it takes a lot of space on disk and is extremely inconvenient *if* a mistake does need to be corrected.

software libraries

The answer to the first objection—the code should not have to be rewritten over and over again—is software libraries. They serve to collect often-required functionality, to endow it with a standardised and well-documented interface, and to make it available to other programs. For example, an essential ingredient for a functioning Linux system is `libc`, which bundles all sorts of vital functionality for programs written in the C programming language.

Practically all programs on a Linux system are written in C, if only indirectly: Many are written in native C, some are written in languages derived from C such as C++ which usually also use `libc`, and even shell scripts and programs in languages such as `awk`, Python, and Perl use the shell or the `awk`, `python`, and `perl` programs—all of which are written in C …

In addition to `libc`, a typical Linux system contains a few hundred or thousand other libraries that cater for requirements ranging from support for mathematical functions or various data formats to providing complete graphical interfaces.

linker

In the first instance, libraries are important when programming. In the simplest case, a “linker” is used to combine a program’s “object code”—that is, the output of the compiler, which translates a program written, say, in C to instructions executable on the CPU—with the object code of the libraries used by this program to yield an “executable program” that you can then start.

Consider, by way of illustration, the following tiny C program:

```
/* sqrttest.c */
#include <math.h>
#include <stdio.h>
int main(void)
{
  printf("%g\n", sin(3.141592654/2.0));
  return 0;
}
```

(It serves to answer the nagging question about the value of sin(π/2).) The C compiler turns this into an assembly language program that starts approximately like

```
; sqrttest.s
        .file   "sqrttest.c"
        .section        .rodata
.LC1:
        .string "%g\n"
        .align 8
.LC0:
        .long   1414677840
        .long   1073291771
        .text
.globl main
        .type   main, @function
```

```
main:
        leal    4(%esp), %ecx
        andl    $-16, %esp
        pushl   -4(%ecx)
        pushl   %ebp
◁◁◁◁◁
```

and the assembler, in turn, produces an object file which, displayed "readably", looks like

```
7f 45 4c 46 01 01 01 00  00 00 00 00 00 00 00 00
01 00 03 00 01 00 00 00  00 00 00 00 00 00 00 00
14 01 00 00 00 00 00 00  34 00 00 00 00 00 28 00
0b 00 08 00 8d 4c 24 04  83 e4 f0 ff 71 fc 55 89
e5 51 83 ec 14 dd 05 08  00 00 00 dd 1c 24 e8 fc
ff ff ff dd 5c 24 04 c7  04 24 00 00 00 00 e8 fc
◁◁◁◁◁
```

(952 bytes in total, if you must know). This file then needs to be put together with the libraries `libc` (for the `printf()` function) and `libm` (for the `sin()` function) to create an executable program, which is the linker's job. Fortunately all of this can be done using a single command such as

```
$ gcc -static sqrttest.c -lm -o sqrttest-static
```

On the author's system, the resulting program weighs in at a mere 560,608 bytes.

Advantages of libraries

In this case, the program and libraries form a permanent combinaton. The libraries spare us the tedious repetition of code that is useful in several programs, but if a problem is found in a library, we need to re-link all programs using that library. The fact that a trivial program is half a megabyte long is also vaguely less than satisfying.

dynamic libraries

Here is where dynamic libraries come in. The basic idea behind them is that the actual program and libraries are not combined permanently during linking, but are only put together when the program is actually executed. Thus, program and libraries remain independent of each other. This results in the following advantages:

- As long as a library's interface stays the same (i. e., the "signatures" of the functions concerned—the number and types of their arguments and results—do not change), the library can easily be replaced without having to re-link all programs using it. When a program is started, it automatically fetches the new library version.
- If *n* programs are using the library, it does not require *n* times the space on disk (as part of each of those programs). One file is enough.
- In fact it usually suffices to have the library in *memory* just once. All processes can share one copy. Another saving! This is why we speak of "shared libraries".

Incidentally, you're paying for this with time: On the one hand it takes a bit longer to launch a program using dynamic libraries, since the connection between the actual program and its libraries needs to be made at runtime instead of when the program is compiled to an executable file. On the other hand—depending on the processor and programming model—it may be the case that dynamic libraries carry a runtime penalty as opposed to "static", i. e., connected-at-compile-time, libraries. This penalty usually amounts to a low single-digit percentage and is more than outweighed by the advantages mentioned above.

Our example from above could be linked against dynamic libraries using

```
$ gcc sqrttest.c -lm -o sqrttest-dynamic
```

The result is all of 6510 bytes long—a marked improvement from more than 500,000!

Perhaps you would think that disk space is so cheap nowadays that half a megabyte here or there is not worth losing sleep over. Dynamic libraries were introduced to Unix in the 1980s, when disk space was still measured in megabytes rather than terabytes; libraries weren't quite as big as they are today, but one used to notice the difference rather more in these days. Anyway, the other advantages are really even more important.

Rehabilitation of statically linked programs

To rehabilitate statically linked programs we should mention that they also have their place. On the one hand, they work fairly reliably even is the system is all but hosed otherwise—should you ever manage to somehow junk your dynamic `libc`, you will be glad to have a statically-linked version of `busybox` or `sash` on your system, since `bash` and its friends such as `cp` or `tar` will be no good for repairs. (The alternative would be to boot the rescue system, but that is of course a cop-out for wimps.) On the other hand, statically linked programs are huge and inconvenient, but you know what you have[1] – so if you want to distribute a complex program in executable form, linking statically makes you independent of the libraries available on the systems the program is supposed to run on later. (This can of course imply different problems, but an exhausting discussion would be beyond the scope of this manual.)

LSB

A large part of the exertions of the LSB (Linux Standard Base), an agreement between various distributors as to which environment should be available for software distributed in executable form by third-party vendors (think SAP, Oracle and games), concerns itself with prescribing the version and content of important dynamic libraries.

11.2 Dynamic Libraries In Practice

As we said earlier, most executable programs on a typical Linux system are linked dynamically. You can make sure using the `file` command:

```
$ file sqrttest-dynamic
sqrttest-dynamic: ELF 32-bit LSB executable, Intel 80386, ▷
 ◁   version 1 (SYSV), dynamically linked (uses shared libs),▷
 ◁   for GNU/Linux 2.6.8, not stripped
```

By way of comparison:

```
$ file sqrttest-static
sqrttest-static: ELF 32-bit LSB executable, Intel 80386, ▷
 ◁   version 1 (SYSV), statically linked, for GNU/Linux 2.6.8,▷
 ◁   not stripped
```

The "`not stripped`" refers to the fact that our small (?) C program still contains information supposed to make debugging easier. You can slim it down to just below 500 kB by executing "`strip sqrttest-static`".

Required libraries

You can find out in detail which libraries `sqrttest-dynamic` requires by using the `ldd` command:

[1] Apologies to the Persil man.

```
$ ldd sqrttest-dynamic
        linux-gate.so.1 =>  (0xb7f51000)
        libm.so.6 => /lib/i686/cmov/libm.so.6 (0xb7f0b000)
        libc.so.6 => /lib/i686/cmov/libc.so.6 (0xb7db0000)
        /lib/ld-linux.so.2 (0xb7f52000)
```

Its output tells you the names of the libraries as well as the names of the files corresponding to them—if possible (otherwise the file name is empty).

We have already discussed `libc` and `libm`. `/lib/ld-linux.so.2` is the "dynamic linker", the program that causes the actual program to be connected to its dynamic libraries when it is started (we shall come back to this presently).

So what is `linux-gate.so.1`, and what is it good for? This question is great for finding out how much your buddies *really* know about Linux. The most obvious observation is that `ldd` didn't find a file corresponding to this library. Thus by rights the program should not even run. If you look you will even find that there is *no file on the system by this name at all*. However, this doesn't matter, as the kernel takes care of making this "virtual dynamic library" available to all programs. `linux-gate.so.1` is used to speed up system calls on modern x86 processors (which, in 2009, basically means "all of them"). The details are too unsavoury to explain in detail; read [Pet08] but not immediately before or after a meal.

On Intel- and AMD-based 64-bit systems, the `linux-vdso.so.1` "library" serves the same purpose as `linux-gate.so.1` on 32-bit systems. Don't let yourself become confused.

Libraries: where?

Typical places where Linux distributors put libraries (static or dynamic ones) include the `/lib` and `/usr/lib` directories. As usual, the former is mostly meant for libraries that need to be available immediately after the system is booted, and the latter caters for those that are only needed after all file systems have been mounted.

Recognising dynamic libraries

How do you recognise a dynamic library, anyway? By its name—the names of dynamic libraries usually end in `.so`, usually followed by a version number. (The names of static libraries end in `.a`, and a version number does not apply there.) If you want to be anal-retentive, you can can also trot out `file`:

```
$ file /lib/libc-2.7.so
/lib/libc-2.7.so:  ELF 32-bit LSB shared object, Intel 80386,▷
 ◁ version 1 (SYSV), dynamically linked (uses shared libs),▷
 ◁ for GNU/Linux 2.6.8, stripped
```

Look closely—we have

```
ELF 32-bit LSB shared object
```

instead of

```
ELF 32-bit LSB executable
```

—a subtle difference!

With static libraries, the following happens:

```
$ file /usr/lib/libc.a
/usr/lib/libc.a: current ar archive
```

It does take a little experience to conclude "library!" from this—take a look at ar(1).

File names for dynamic libraries

In "real life", the file name of a dynamic library could look approximately like this:

```
$ ls -l /usr/lib/libcurl.*
lrwxrwxrwx root root     16 Jan 22 01:23 /usr/lib/libcurl.so.4▷
 ◁  -> libcurl.so.4.1.0
-rw-r--r-- root root 271364 Dec 27 14:33 /usr/lib/libcurl.so.4.1.0
```

The symbolic link is used to relate actual programs and the library. A program actually asks for a specific version of a library—in our example, `libcurl.so.4`. It depends on the system whether this amounts to `/usr/lib/libcurl.so.4.1.0` or `/usr/lib/libcurl.so.4.2.1`, whichever is installed. (The basic assumption is that all files purporting to be `libcurl.so.4` implement the same interface.) More information about versioning dynamic libraries is in Section 11.4.

Incidentally, dynamic libraries may depend on other dynamic libraries, too. You may find out about this using `ldd` (of course):

```
$ ldd /usr/lib/libcurl.so.4.1.0
        linux-gate.so.1 =>  (0xb7f6f000)
        libidn.so.11 => /usr/lib/libidn.so.11 (0xb7edb000)
        libssh2.so.1 => /usr/lib/libssh2.so.1 (0xb7eba000)
        liblber-2.4.so.2 => /usr/lib/liblber-2.4.so.2 (0xb7eac000)
◁◁◁◁◁
```

(What you get to see if you apply `ldd` to a program is the transitive closure over all dynamic libraries that the program together with all of its dynamic libraries depends on.)

Exercises

11.1 [!1] Find a statically-linked executable on your system (apart from `sqrttest-static`).

11.2 [1] What does `ldd` say if you apply it to a statically-linked program?

11.3 [2] (For shell programmers.) Which program in `/usr/bin` on your system is linked against the greatest number of dynamic libraries?

11.3 Installing and Locating Dynamic Libraries

dynamic linker

File/etc/ld.so.cache

When you start a program, the dynamic linker must find and load the required libraries. However, the program itself contains only a library's name, not the name of the file it is to be found in. To avoid having the dynamic linker search all of the file system (or even only the `/lib` and `/usr/lib` directories), the File/etc/ld.so.cache file contains an "index" of all known dynamic libraries. The dynamic linker finds only those libraries that occur in this index.

`ldconfig`

`/etc/ld.so.conf`

The index is created using the `ldconfig` program, which searches all directories listed in `/etc/ld.so.conf` as well as the two standard directories `/lib` and `/usr/lib` for libraries. All found libraries are entered into the index. In addition, `ldconfig` takes care of creating the "main version number" symbolic links for library files.

`/lib` and `/usr/lib` are always searched for libraries, no matter what `/etc/ld.so.conf` says.

index content

You can inspect the current state of the index using

```
# ldconfig -p
909 libs found in cache `/etc/ld.so.cache'
        libzvt.so.2 (libc6) => /opt/gnome/lib/libzvt.so.2
        libz.so.1 (libc6) => /lib/libz.so.1
        libz.so (libc6) => /usr/lib/libz.so
◁◁◁◁◁
```

This does not generate a new index, but only outputs the content of `ld.so.cache` in a readable format. If a library does not show up here, it cannot be found by the linker, either.

New dynamic libraries

To add a new dynamic library from the standard directories—besides `/lib` and `/usr/lib`, this typically includes `/usr/local/lib`—, then a simple call to

```
# ldconfig
```

should suffice to update `/etc/ld.so.cache`. If your library is not located in one of the directories mentioned in `/etc/ld.so.conf`, you have to add the directory to that file first.

Usually the package installation tool of your distribution should take care of invoking `ldconfig` when packages including dynamic libraries are installed. Therefore, the previous paragraph applies mostly if you want to install libraries that you wrote yourself, or ones in software packages that you compiled from source code.

LD_LIBRARY_PATH

If you lack the administrator rights necessary to call `ldconfig` you can still use your own dynamic libraries. On top of the cache in `/etc/ld.so.cache`, the dynamic linker also searches the directories enumerated in the `LD_LIBRARY_PATH` environment variable. Its syntax corresponds to that of the `PATH` variable—directory names are separated by colons:

```
$ export LD_LIBRARY_PATH=$HOME/lib:/opt/foo/lib
```

The dynamic linker searches the content of the directories listed in `LD_LIBRARY_PATH` (if set) first, then the content of `/etc/ld.so.cache`, and then the content of the `/lib` and `/usr/lib` directories (in that order—this is just a safety net in case `/etc/ld.so.cache` does not exist). Libraries in a directory mentioned in `LD_LIBRARY_PATH` whose names are the same as "official" libraries are thus found first.

The documentation of the dynamic linker (`ld-linux.so`(8)) contains many interesting and thrilling options that allow you to do strange and wonderful things.

Exercises

11.4 [!2] The directories in `LD_LIBRARY_PATH` are not considered when the program to be started uses the Set-UID or Set-GID mechanism (see Section 3.5). Why is that?

11.4 Dynamic Library Versioning

Users of Microsoft Windows know the problem called "DLL hell"[2]: Many software packages come with their own versions of important system libraries and do

[2] DLLs, or "dynamically loadable libraries", are the Windows equivalent to the dynamic libraries of Linux.

not shy away from recklessly supplanting pre-existing versions of those libraries with their own. If, of course, the supplanted version happens to be exactly the one that another software package brought in, chaos is not far away.

third-party software

Under Linux this problem is much less grave, since much less third-party software is delivered in executable form to begin with. You receive binary software mostly from your distributor, and other things are compiled from source code (making use of pre-installed libraries as much as possible). Problems are more likely to result from the fact that the Linux software landscape evolves fairly quickly, which of course also applies to libraries.

version numbers

For this reason, dynamic libraries are assigned version numbers that typically contain up to three parts. For example, earlier on we talked about `libcurl.4.1.0`. These version numbers work as follows:

major version number

- The first number after the library name is the "major version number". It should be incremented whenever the programming interface offered by the library to other programs changes in an incompatible fashion. For example, a function that used to accept two arguments might suddenly require three, or an argument that used to be an integer might turn into a pointer to a string.

  It goes without saying that a responsible developer should try to avoid such changes whenever possible, as they imply that the programs using the library need to be adapted.

- The second number (after the second dot) is the "minor version number". It is incremented when the programming interface is extended without changes to existing calls. For example, a completely new function might be added. Older programs can still use the library because everything they require is still available in an identical fashion; of course, newer programs that do use the newly added calls will not work with older versions of the library.

patch level

- The third number (at the end) is the "patch level". It is incremented when there are changes to the library *implementation* that do not affect the interface. For example, a bug in the implementation of a function might be fixed. The fact that the library is dynamically linked makes it possible to simply replace the defective library by the corrected one—all subsequently-started programs automatically pick up the new version.

daemons

  Replacing a defective library by a new one has no impact on programs *already running* with the defective library—such as daemons. When in doubt, such programs must be restarted manually to benefit from the improvements.

  The sufferers in this improvement process are, of course, programs that rely on the *presence* of the implementation error. It sometimes happens that knowledge of a bug gets passed around the developer community, and programmers either exploit it (if it has some beneficial side effect—which happens) or work around it in a way that does not take into account the fact that at some point the error might no longer be present[3]. In such a situation, a correction amounts to an incompatible change and should, in the worst case, cause even the major version number to be incremented.

[3] A long time ago, your author had to contend with a C compiler that would calculate the reciprocal of the actual result when performing floating-point divisions. (Don't ask.) In a software package written by a colleague, there occured exactly one such division in a strategic place, where (uncommented and long forgotten) the dividend and divisor had been swapped to accommodate this error. Ferreting this out as the reason for a grisly sequence of other faults, after updating to a fixed C compiler, took us a few days.

Programs always require a certain major number of a library, such as `libcurl.so.4`, which is mapped to a concrete file, such as `/usr/lib/libcurl.so.4.1.`, by means of the symbolic link.

multiple versions simultaneously

Hence it is quite feasible for a system to simultaneously contain programs requiring multiple (major) versions of the same library. You can simply leave the various library files installed and put the burden of finding the correct one for each program on the dynamic linker.

The LSB standard prescribes certain major versions of common libraries and, in each case, standardises a programming interface for the library in question. LSB-conforming distributions must offer these libraries for the benefit of LSB executables, but these libraries do not need to be the ones that all of the rest of the system is using. Hence it is quite possible to make a distribution conform to LSB from the point of view of library support by placing a set of the required libraries in a different directory, and to foist this on LSB executables instead of the usual system directories for libraries through dynamic linker trickery (such as `LD_LIBRARY_PATH`).

The same strategy is also useful for "normal" third-party software that does not rely on LSB. As long as the kernel-`libc` interface does not change incompatibly (which, fortunately, the kernel developers diligently try to avoid), a software package can provide its own `libc` (plus, presumably, all sorts of other libraries) and so try to become independent of the library support offered by the underlying Linux system. Whether this is always a good idea in real life remains an open question—after all, as a software vendor one may have to support independent bug fixes to the libraries and distribute them to all customers—, but it is a fact that on Linux there is no direct equivalent to "DLL hell". Score one for the penguin!

Commands in this Chapter

busybox	A shell that already contains variants of many Unix tools	`busybox`(1)	174
file	Guesses the type of a file's content, according to rules	`file`(1)	174
ldconfig	Builds the dynamic library cache	`ldconfig`(8)	176
ldd	Displays the dynamic libraries used by a program	`ldd`(1)	174
sash	"Stand-Alone Shell" with built-in commands, for troubleshooting	`sash`(8)	174
strip	Removes symbol tables from object files	`strip`(1)	174

Summary

- Libraries offer frequently-needed functionality in standardised form to several programs.
- Dynamic libraries save memory and disk space and facilitate program maintenance.
- `file` lets you find out whether a program is dynamically linked, and identify shared libraries as such.
- `ldd` outputs the names of dynamic libraries that a program (or a dynamic library) uses.
- Dynamic libraries are searched in `/lib`, `/usr/lib`, and the directories mentioned in `/etc/ld.so.conf`. The `ldconfig` command constructs an index.
- Several versions of the same library may be installed at the same time.

Bibliography

Pet08 Johan Petersson. “What is linux-gate.so.1?”, August 2008. `http://www.trilithium.com/johan/2005/08/linux-gate/`

12

Software Package Management Using Debian Tools

Contents

Goals

- Knowing the basics of Debian packaging tools
- Being able to use `dpkg` for package management
- Being able to use `apt-get`, `apt-cache`, and `aptitude`
- Being aware of the principles of Debian package integrity
- Knowing how to convert RPM packages to Debian packages using `alien`

Prerequisites

- Knowledge of Linux system administration
- Experience with Debian GNU/Linux or a Debian GNU/Linux derivative is helpful

12.1 Overview

Software packages in Debian GNU/Linux and derived distributions such as Ubuntu, Knoppix, Xandros, or Sidux are maintained using the `dpkg` tools. It serves to install software packages, to manage dependencies, to catalog installed software, to control updates to software packages, and to de-install packages that are no longer required. Program such as `aptitude` serve as front-ends to `dpkg`, allowing the convenient selection of software packages. The `debconf` infrastructure is used to configure packages upon installation.

The Debian and Red Hat package management systems were developed at about the same time and have different strengths and weaknesses. As usual in the free software community, the religious wars around `dpkg` and `rpm` have not led to one of the competitors carrying the day. With the increasing popularity of Debian derivatives—most notably Ubuntu—this remains unlikely for the foreseeable future, too.

The LSB standard for a basic Linux infrastructure that third-party vendors can port their software to does prescribe a restricted version of RPM as its package format. However, this does not imply that a LSB-compliant Linux distribution must be RPM-based from the ground up, but only that it must be able to install software packages from third-party vendors that conform to the LSB flavour of RPM.

For Debian GNU/Linux, this is of course a piece of cake. The reason why Debian GNU/Linux is not officially touted as “LSB-compliant” is because LSB is run by an industry consortium of which Debian, being a non-commercial project, is not a member. The description for the `lsb` package on Debian GNU/Linux states:

> The intent of this package is to provide a best current practice way of installing and running LSB packages on Debian GNU/Linux. Its presence does not imply that Debian fully complies with the Linux Standard Base, and should not be construed as a statement that Debian is LSB-compliant.

While its title talks about “Debian tools”, everything in this chapter also applies to Ubuntu, since Ubuntu takes substantial parts of its infrastructure from Debian GNU/Linux. We shall be pointing out significant differences that do exist.

12.2 The Basis: `dpkg`

12.2.1 Debian Packages

packages
package names

Within the Debian infrastructure, the software on the system is divided into packages. Packages have names that indicate the software contained within and their version. The

```
hello_2.8-2_amd64.deb
```

file, for example, contains the `hello` program’s 2.8 version; in particular this is the second release of this package within the distribution (for a future 2.9 package the count would start over at 1). Packages like `apt` which have been specifically developed for Debian do not include a “Debian release number”. The `amd64` indicates that the package contains architecture-specific parts for Intel and AMD x86 processors (and compatibles) in 64-bit mode—32-bit packages use `i386`, and packages that contain only documentation or architecture-independent scripts use `all` instead.

package structure

A Debian package is an archive created using the ar program and generally contains three components:

```
$ ar t hello_2.8-2_amd64.deb
debian-binary
control.tar.gz
data.tar.gz
```

installation

The debian-binary file contains the version number of the package format (currently 2.0). In control.tar.gz there are Debian-specific scripts and control files, and data.tar.gz contains the actual package files. During installation, control.tar.gz is unpacked first, so that a possible preinst script can be executed prior to unpacking the actual package files. After this, data.tar.gz will be unpacked, and the package will be configured if necessary by executing the postinst script from control.tar.gz.

Exercises

12.1 [2] Obtain an arbitrary Debian package (such as hello) and take it apart using ar and tar. Can you find the installation scripts? Which information is contained in control.tar.gz, and which is in data.tar.gz?

12.2.2 Package Installation

You can easily install a locally-available Debian package using the

```
# dpkg --install hello_2.8-2_amd64.deb
```

command, where --install can be abbreviated to -i. With --unpack and --configure (-a), the unpacking and configuration steps can also be executed separately.

In real life, the short option names such as -i are convenient. However, if you intend to pass the LPI-101 exam, you should be sure to learn the long option names as well, since these, vexatingly, occur in the exam questions. In the case of -i and --install, it is probably straightforward to come up with the correspondence; with -a and --configure, this is already somewhat less obvious.

dpkg.cfg

Options for dpkg can be given on the command line or else placed in the /etc/dpkg/dpkg.cfg file. In this file, the dashes at the start of the option names must be omitted.

Upgrade

If a package is installed using "dpkg --install", even though an earlier version already exists on the system, the older version is deinstalled before configuring the new one. If an error occurs during installation, the old version can be restored in many cases.

installation problems

There are various reasons that might prevent a successful package installation, including:

- The package requires one or more other packages that either have not yet been installed, or that are not included in the same installation operation. The corresponding check can be disabled using the --force-depends option—but this can severely mess up the system.
- An earlier version of the package is installed and set to hold (e. g., using aptitude). This prevents newer versions of the package from being installed.
- The package tries to unpack a file that already exists on the system and belongs to a different package, unless the current package is explicitly labeled as "replacing" that package, or the --force-overwrite option was specified.

conflicts

Some packages conflict with each other (see the possibilities for package dependencies on page 187). For example, only one mail transport program may be installed at one time; if you want to install, e. g., Postfix, Exim (the Debian default MTA) must be removed at the same time. `dpkg` manages this if certain conditions are fulfilled.

Virtual packages

Sometimes packages do not depend on a particular other package but on a "virtual" package describing a feature that can, in principle, be provided by any of several other packages, such as `mail-transport-agent`, which is provided by packages like `postfix`, `exim`, or `sendmail`. In this case it is possible to replace, say, Exim by Postfix in spite of dependencies, as a program providing the "virtual" functionality will be available at all times.

Exercises

12.2 [1] Download a Debian package—say, `hello`—from `ftp.debian.org` (or any other Debian mirror) and install it using `dpkg`. (If you use anything else, such as `apt-get`—see next section—, you're cheating!) You can find Debian packages on the server reasonably conveniently given their names, by looking in `pool/main` for a subdirectory corresponding to the first character of the name, and in there looking for a subdirectory whose name is that of the package[1], in our case `pool/main/h/hello`. Exception: Since very many package names start with `lib`, a package like `libfoobar` ends up in `pool/main/libf`.

12.3 [2] Locate the current list of virtual packages in Debian GNU/Linux. Where did you find it? When did the last update take place?

12.2.3 Deleting Packages

A package is removed using the

```
# dpkg --remove hello
```

command ("`dpkg -r`", for short). Its configuration files (all the files listed in the `conffiles` file within `control.tar.gz`), though, are kept around in order to facilitate a subsequent reinstallation of the package. The

```
# dpkg --purge hello
```

(or "`dpkg -P`") command removes the package including its configuration files.

The "configuration files" of a Debian package are all files in the package whose names occur in the `conffiles` file in `control.tar.gz`. (Look at `/var/lib/dpkg/info/`⟨*package name*⟩`.conffiles`.)

Removal problems

Package removal does not necessarily work, either. Possible obstacles include:

- The package is required by one or more other packages that are not about to be removed as well.
- The package is marked "essential" (to system functionality). The shell, for example, cannot simply be removed since some important scripts could no longer be executed.

Here, too, the relevant checks can be disabled using suitable `--force-`… options (at your own risk).

Exercises

12.4 [1] Remove the package you installed during Exercise 12.2. Make sure that its configuration files are removed as well.

[1] The source code package, really, which may differ. So don't get too fancy.

12.2.4 Debian Packages and Source Code

source control file

When dealing with source code, a basic principle of the Debian project is to distinguish clearly between the original source code and any Debian-specific changes. Accordingly, all changes are placed in a separate archive. In addition to the Debian-specific control files, these include more or less extensive fixes and customisations to the software itself. For each package version, there is also a "source control file" (using the .dsc suffix) containing checksums for the original archive and the changes file, which will be digitally signed by the Debian maintainer in charge of the package:

```
$ ls hello*
-rw-r--r-- 1 anselm anselm   6540 Jun  7 13:18 hello_2.8-2.debian.tar.gz
-rw-r--r-- 1 anselm anselm   1287 Jun  7 13:18 hello_2.8-2.dsc
-rw-r--r-- 1 anselm anselm 697483 Mai 27 23:47 hello_2.8.orig.tar.gz
```

You can also see that the original source code archive does not change for all of the 2.8 version of the program (it does not contain a Debian release number). Every new version of the Debian package of hello's 2.8 version comes with new .dsc and .debian.tar.gz files. The latter contains all the changes relative to the original archive (rather than the hello_2.8-2 package).

In former times, the Debian project used a less complicated structure where there was one single file (created with diff) containing all Debian-specific changes—in our example, hypothetically hello_2.8-2.diff.gz. This approach is still supported, and you may find this structure with older packages that have not been changed to use the new method instead. The new structure does have the advantage that different changes—like the introduction of the Debian-specific control files and any changes to the actual original code—can be more cleanly separated, which greatly simplifies maintaining the package within the Debian project.

dpkg-source

The dpkg-source command is used to reconstruct a package's source code from the original archive and the Debian changes such that you can recompile your own version of the Debian package. To do so, it must be invoked with the name of the source control file as an argument:

```
$ dpkg-source -x hello_2.8-2.dsc
```

The original archive and the .debian.tar.gz or .diff.gz file must reside in the same directory as the source control file. dpkg-source also places the unpacked source code there.

Preparing Debian packages

dpkg-source is also used when generating source archives and Debian change files during Debian package preparation. However, this topic is beyond the scope of the LPIC-1 certification.

Exercises

12.5 [1] Obtain the source code for the package you installed during Exercise 12.2, and unpack it. Take a look at the debian subdirectory of the resulting directory.

12.2.5 Package Information

package list

You can obtain a list of installed packages using "dpkg --list" (-l, for short):

```
$ dpkg --list
Desired=Unknown/Install/Remove/Purge/Hold
| Status=Not/Installed/Config-files/Unpacked/Failed-config/H
```

```
|/ Err?=(none)/Hold/Reinst-required/X=both-problems (Status,
||/ Name           Version        Description
+++-==============-==============-==========================
ii  a2ps           4.13b+cvs.2003 GNU a2ps - 'Anything to Po
ii  aalib1         1.4p5-19       ascii art library
ii  abcm2ps        4.0.7-1        Translates ABC music descr
ii  abcmidi        20030521-1     A converter from ABC to MI
◁◁◁◁◁
```

shell search patterns

(truncated on the right for space reasons) This list can be narrowed down using shell search patterns:

```
$ dpkg -l lib*-tcl
Desired=Unknown/Install/Remove/Purge/Hold
| Status=Not/Installed/Config-files/Unpacked/Failed-config/H
|/ Err?=(none)/Hold/Reinst-required/X=both-problems (Status,
||/ Name           Version        Description
+++-==============-==============-==========================
pn  libdb3-tcl     <none>         (no description available)
un  libdb4.0-tcl   <none>         (no description available)
un  libdb4.1-tcl   <none>         (no description available)
un  libmetakit-tcl <none>         (no description available)
ii  libsqlite-tcl  2.8.9-1        SQLite TCL bindings
rc  libsqlite0-tcl 2.6.1-2        SQLite TCL bindings
```

The packages with version "<none>" are part of the distribution but are either not installed on the current system (status un) or have been removed (status pn).

package status

You can find out about an individual package's status with the --status (-s) option:

```
$ dpkg --status hello
Package: hello
Status: install ok installed
Priority: optional
Section: devel
Installed-Size: 553
Maintainer: Santiago Vila <sanvila@debian.org>
Architecture: amd64
Version: 2.8-2
Depends: libc6 (>= 2.4), dpkg (>= 1.15.4) | install-info
Description: The classic greeting, and a good example
 The GNU hello program produces a familiar, friendly greeting. It
  allows non-programmers to use a classic computer science tool which
  would otherwise be unavailable to them.
  .
  Seriously, though: this is an example of how to do a Debian package.
  It is the Debian version of the GNU Project's `hello world' program
  (which is itself an example for the GNU Project).
Homepage: http://www.gnu.org/software/hello/
```

Besides the package name (Package:), its output includes information about the package's status and priority (from required via important, standard and optional down to extra) and its approximate area of interest (Section:). The Maintainer: is the person who is in charge of the package on behalf of the Debian project.

Priorities

Packages of priority required are necessary for proper operation of the system (usually because dpkg depends on them). The important priority encompasses packages one would expect to be available on a Unix-like system[2].

[2] The definition is something like "A package is important if, should it be missing, experienced Unix users would shake their heads and go "WTF?".

standard adds those packages that make sense for a net but not overly restrictive system running in text mode—this priority describes what you get if you install Debian GNU/Linux without selecting any additional packages. The optional priority applies to everything you might want to install if you do not look too closely and have no particular requirements. This includes things like the X11 GUI and a whole load of applications (such as TeX). There should be no conflicts within optional. Finally, extra is used for all packages that conflict with packages of other priorities, or that are only useful for specialised applications.

Packages may not depend on packages of lower priority. For this to hold in all cases, the priorities of some packages have deliberately been tweaked.

package dependencies

An important area of information are the package dependencies, of which there are several types:

Depends The named packages must be configured for the package to be able to be configured. As in the preceding example, specific versions of the packages may be called for.

Pre-Depends The named packages must be completely installed before installation of the package can even begin. This type of dependency is used if, for example, the package's installation scripts absolutely require software from the other package.

Recommends A non-absolute but very obvious dependency. You would nearly always install the named packages alongside this package, and only refrain from doing so in very unusual circumstances.

Suggests The named packages are useful in connection with the package but not required.

Enhances Like Suggests, but in reverse—this package is useful for the named package (or packages).

Conflicts This package cannot be installed at the same time as the named packages.

If a package isn't installed locally at all, "dpkg --status" only outputs an error message:

```
# dpkg -s xyzzy
Package `xyzzy' is not installed and no info is available.
Use dpkg --info (= dpkg-deb --info) to examine archive files,
and dpkg --contents (= dpkg-deb --contents) to list their contents.
```

list of files

The --listfiles (-L) option provides a list of files within the package:

```
$ dpkg --listfiles hello
/.
/usr
/usr/share
/usr/share/doc
/usr/share/doc/hello
/usr/share/doc/hello/changelog.Debian.gz
/usr/share/doc/hello/copyright
/usr/share/doc/hello/NEWS
/usr/share/doc/hello/changelog.gz
/usr/share/info
/usr/share/info/hello.info.gz
/usr/share/man
```

```
/usr/share/man/man1
/usr/share/man/man1/hello.1.gz
/usr/share/locale
◁◁◁◁◁
```

package search

Finally, you can use the `--search` (or `-S`) option to find out which package (if any) claims a given file. Search patterns are allowed:

```
$ dpkg -S bin/m*fs
dosfstools: /sbin/mkdosfs
cramfsprogs: /usr/sbin/mkcramfs
util-linux: /sbin/mkfs.cramfs
smbfs: /sbin/mount.smbfs
◁◁◁◁◁
```

The search may take some time, though.

If you're looking for the package for a file that is *not* on your system—for example, if you plan to install that package afterwards—, you can use the search form on `http://www.debian.org/distrib/packages#search_contents`. This allows you to search any or all Debian distributions and architectures as well as to search for exact file name matches and file names containing certain search terms.

Exercises

12.6 [3] How many packages whose names start with `lib` are installed on your system? How many of those packages have priority `required`?

12.2.6 Package Verification

integrity of an installed package

The integrity of an installed package can be checked using the `debsums` program (from the eponymous package):

```
$ debsums hello
/usr/share/doc/hello/changelog.Debian.gz    OK
/usr/share/doc/hello/copyright              OK
/usr/share/doc/hello/NEWS                   OK
/usr/share/doc/hello/changelog.gz           OK
/usr/share/info/hello.info.gz               OK
◁◁◁◁◁
```

This compares the MD5 checksums of the individual files with the content of the corresponding file in `/var/lib/dpkg/info` (here, `hello.md5sums`). If an actual file's checksum does not match the set value, `FAILED` is displayed in place of `OK`.

protection from intruders

`debsums` can uncover "inadvertent" changes to a package's files, but does not provide protection from intruders who maliciously modify files. After all, a cracker could place the checksum of a modified file in its package's `md5sums` list. Neither does this method help agains "Trojan" packages that hide malicious code behind an innocuous facade. We shall be coming back to the "integrity of packages" topic in Section 12.4.

Exercises

12.7 [!2] Change a file in an installed Debian package. (Look for a not-so-important one, like that from Exercise 12.2.) You could, for example, append a few lines of text to the `README.Debian` file (be `root`). Check the integrity of the package's files using `debsums`.

12.3 Debian Package Management: The Next Generation

12.3.1 APT

`dpkg` is a powerful tool, but still somewhat restricted in its potential. For example, it is a bit aggravating that it will notice unfilled dependencies between packages, but then just throw in the towel instead of contributing constructively to a solution of the problem. Furthermore, while it is nice be able to install locally-available packages, one would wish for convenient access to FTP or web servers offering packages.

The `dselect` program, which in the early days of Debian served as an interactive front-end to package management, is officially deprecated today—its inconvenience was proverbial, even though reading the manual did help as a rule.

Quite early in the history of the Debian project (by today's standards), the Debian community started developing APT, the "Advanced Packaging Tool". This project, in its significance as in its ultimate pointlessness, is comparable to the quest of the Knights of the Round Table for the Holy Grail, but, like the Grail quest, APT development led to many noble deeds along the way. Although few dragons were slain and damsels freed from distress, the APT developers produced very important and powerful "partial solutions" whose convenience and feature set remains unequalled (which is why some RPM-based distributions have begun to ad-"apt" them for their purposes).

12.3.2 Package Installation Using `apt-get`

The first of these tools is `apt-get`, which represents a sort of intelligent superstructure for `dpkg`. It does not offer an interactive interface for package selection, but could initially be used as a back-end for `dselect`, to install packages selected in `dselect`. Today it is mostly useful on the command line. The most important properties of `apt-get` include the following:

Several installation sources

- `apt-get` can manage a set of installation sources simultaneously. For example, it is possible to use a "stable" Debian distribution on CD-ROM in parallel to a HTTP-based server containing security updates. Packages are normally installed from CD-ROM; only if the HTTP server offers a more current version of a package will it be fetched from the network. Certain packages can be requested from certain sources; you can, for example, use a stable Debian distribution for the most part but take some packages from a newer "unstable" distribution.

Upgrades

- It is possible to update all of the distribution at once (using "`apt-get dist-upgrade`"), with dependencies being resolved even in the face of package renamings and removals.

auxiliary tools

- A multitude of auxiliary tools allows, e. g., setting up caching proxy servers for Debian packages (`apt-proxy`), installing packages on systems that are not connected to the Internet (`apt-zip`), or retrieving a list of bugs for a package before actually installing it (`apt-listbugs`). With `apt-build`, you can compile packages with specific optimisations for your system and create a local package repository containing such packages.

package sources

Package sources for `apt-get` are declared in `/etc/apt/sources.list`:

```
deb http://ftp.de.debian.org/debian/ stable main
deb http://security.debian.org/ stable/updates main
deb-src http://ftp.de.debian.org/debian/stable main
```

Binary packages will be fetched from `http://ftp.de.debian.org/`, as will the corresponding source code. In addition, the `security.debian.org` server is accessed, on which the Debian project places updated package version that fix security bugs.

operating procedure

The standard operating procedure using `apt-get` is as follows: First you update the local package availability database:

```
# apt-get update
```

This consults all package sources and integrates the results into a common package list. You can install packages using “`apt-get install`”:

```
# apt-get install hello
Reading Package Lists... Done
Building Dependency Tree... Done
The following NEW packages will be installed:
  hello
0 upgraded, 1 newly installed, 0 to remove and 0 not upgraded.
Need to get 68.7kB of archives.
After unpacking 566kB of additional disk space will be used.
```

This will also install or upgrade all packages mentioned in `Depends:` dependencies, as well as any packages that these packages depend upon, and so on.

You may also install several packages at the same time:

```
# apt-get install hello python
```

or install some packages and install others simultaneously: The

```
# apt-get install hello- python python-django+
```

command would remove the `hello` package and install the `python` and `python-django` packages (including their dependencies). The “+” is not mandatory but allowed. With “`apt-get remove`” you can remove packages directly.

simple update

The “`apt-get upgrade`” installs the newest available versions of all packages installed on the system. This will not remove installed packages nor install new packages; packages that cannot be updated without such actions (because dependencies have changed) remain at their present state.

“intelligent” update

The “`apt-get dist-upgrade`” command enables an “intelligent” conflict resolution scheme which tries to resolve changed dependencies by judiciously removing and installing packages. This prefers more important packages (according to their priority) over less important ones.

source code

You can fetch a package’s source code using the “`apt-get source`” command:

```
# apt-get source hello
```

This also works if the binary package is one of several that have been created from a (differently named) source package.

The `apt` programs are usually configured by means of the `/etc/apt/apt.conf` file. This includes options for `apt-get`, `apt-cache`, and other commands from the `apt` bunch.

Exercises

12.8 [!1] Use `apt-get` to install the `hello` package and then remove it again.

12.9 [1] Download the source code for the `hello` package using `apt-get`.

12.3.3 Information About Packages

Another useful program is `apt-cache`, which searches `apt-get`'s package sources:

apt-cache

```
$ apt-cache search hello                             hello in name or description
◁◁◁◁◁
grhino-data - othello/reversi boardgame - data-files
gtkboard - many board games in one program
hello - The classic greeting, and a good example
hello-debhelper - The classic greeting, and a good example
jester - board game similar to Othello
◁◁◁◁◁
$ apt-cache show hello
Package: hello
Version: 2.8-2
Installed-Size: 553
Maintainer: Santiago Vila <sanvila@debian.org>
Architecture: amd64
◁◁◁◁◁
```

The output of "`apt-cache show`" mostly corresponds to that of "`dpkg --status`", except that it works for all packages in a package source, no matter whether they are installed locally, while `dpkg` only deals with packages that are actually installed.

There are also a few other interesting `apt-cache` subcommands: `depends` displays all the dependencies of a package as well as the names of packages fulfilling that dependency:

```
$ apt-cache depends hello
hello
  Depends: libc6
 |Depends: dpkg
  Depends: install-info
```

The vertical bar in the second dependency line indicates that the dependency in this line or the one in the following line must be fulfilled. In this example, the `dpkg` package or the `install-info` package must be installed (or both).

Conversely, `rdepends` lists the names of all packages depending on the named package:

```
$ apt-cache rdepends python
python
Reverse Depends:
  libboost-python1.4.1
  mercurial-nested
  mercurial-nested
  python-apt
  python-apt
◁◁◁◁◁
```

If a package occurs several times in the list, this is probably because the original package specified it several times, typically with version numbers. The `python-apt` package, for example, contains among other things

```
… python (>= 2.6.6-7~), python (<< 2.8), …
```

to signal that it will only work with particular versions of the Debian Python package.

`stats` provides an overview of the content of the package cache:

```
$ apt-cache stats
Total package names: 33365 (1335k)                        All packages in the cache
  Normal packages: 25672                                Packages that really exist
  Pure virtual packages: 757                       Placeholders for functionality
  Single virtual packages: 1885                        Just one implementation
  Mixed virtual packages: 267                          Several implementations
  Missing: 4784                    Packages in dependencies that no (longer?) exist
Total distinct versions: 28955 (1506k)                Package versions in the cache
Total distinct descriptions: 28955 (695k)
Total dependencies: 182689 (5115k)                 Number of pairwise relationships
Total ver/file relations: 31273 (500k)
Total Desc/File relations: 28955 (463k)
Total Provides mappings: 5747 (115k)
Total globbed strings: 100 (1148)
Total dependency version space: 756k
Total slack space: 73.5k
Total space accounted for: 8646k
```

Exercises

12.10 [2] How can you find *all* packages that must be installed for a particular package to work? (Compare the output of "`apt-cache depends x11-apps`" to that of "`apt-cache depends libxt6`".)

12.3.4 `aptitude`

The program `aptitude` does package selection and management and has taken over the old `dselect`'s rôle in Debian GNU/Linux. On the console or inside a terminal emulator, it features an interactive user interface with menus and dialogs, but also provides command-line options that are roughly compatible to those of `apt-get`. Since Debian 4.0 (popularly called "etch"), `aptitude` is the recommended program for package installation and updates.

Newer versions of `aptitude` include a GTK+-based user interface that can be installed optionally.

improvements

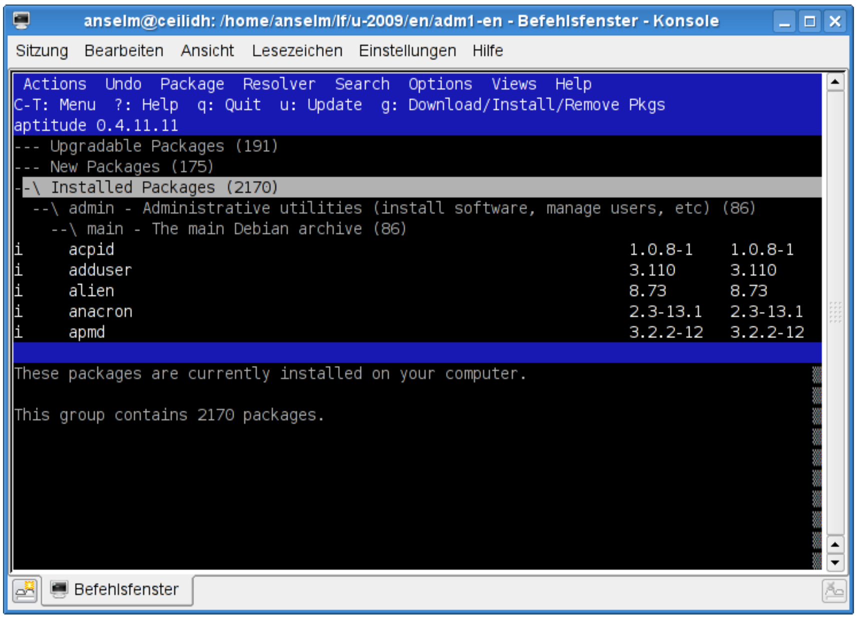
Compared to `apt-get` and `dselect`, `aptitude` offers various improvements, including:

- It does not necessarily need to be invoked as `root`, but asks for the `root` password before actions requiring administrator privileges are performed.
- It can remember which packages have been installed to fulfil dependencies, and remove these automatically if all packages depending on them have been removed. (In the meantime `apt-get` has learned to do that, too; see `apt-get`(8), the `autoremove` command.)
- With `aptitude`, you have interactive access to all versions of a package available from various package sources, not just the most up-to-date one.

interactive UI

The `aptitude` command invokes the interactive UI (Figure 12.1). Near the top of the console (or terminal, as the case may be) you see a "menu bar", below that there is a line with a short help message and a line with the program's version number. The remainder of the screen is split in two parts: The upper half displays an overview of the various types of package (updated, new, installed, and so on), the lower half is used for explanatory messages.

With the ↑ and ↓ keys you can navigate in the package list. Lines starting wiht `---` represent the headings of "subtrees" of the package list, and ↩ can be


anselm@ceilidh: /home/anselm/lf/u-2009/en/adm1-en - Befehlsfenster - Konsole
Sitzung Bearbeiten Ansicht Lesezeichen Einstellungen Hilfe
Befehlsfenster

```
 Actions  Undo  Package  Resolver  Search  Options  Views  Help
C-T: Menu  ?: Help  q: Quit  u: Update  g: Download/Install/Remove Pkgs
aptitude 0.4.11.11
--- Upgradable Packages (191)
--- New Packages (175)
--\ Installed Packages (2170)
  --\ admin - Administrative utilities (install software, manage users, etc) (86)
    --\ main - The main Debian archive (86)
i     acpid                                                        1.0.8-1    1.0.8-1
i     adduser                                                      3.110      3.110
i     alien                                                        8.73       8.73
i     anacron                                                      2.3-13.1   2.3-13.1
i     apmd                                                         3.2.2-12   3.2.2-12

These packages are currently installed on your computer.

This group contains 2170 packages.
```

Figure 12.1: The `aptitude` program

used to "open" the next level of such a subtree. (You can open all of the subtree by typing].) / gives you a window that lets you enter a search term (or regular expression) for a package name. When scrolling through the package lists, explanations for the packages encountered are displayed in the lower part of the screen, and you can scroll these up or down using the a and z keys. The i key lets you change from the explanatory text to a representation of the dependencies.

If the cursor bar sits on a package's line, you can select it for installation or updating using the + key, or mark it for deletion using -. If you want to remove it completely (as in "`dpkg --purge`"), use _. = sets a package's status to "hold", which means that it will no longer be automatically upgraded.

With u, you can update the package lists (like "`apt-get update`") and then check the "Updated Packages" subtree to find which packages `aptitude` would update. Using U, you can mark all of these packages for actual updating. The "New Packages" subtree displays those packages added since the last update; f empties this list and places these packages among the "normal" lists. The Ctrl+t key combination opens the menu bar, in which you can move using the arrow keys and select a function using ↩.

The g command starts the actual installation, update, or package removal. At first it shows an overview of the planned actions, which you may revise using the usual commands. Another g starts the actual work: First all required new packages are fetched, then `aptitude` calls `dpkg` to atually install or remove the desired packages.

Contrary to popular perception, `aptitude` is not really a front-end to `apt-get`, but does by itself whatever `apt-get` would otherwise do.

solution strategies

If conflicts occur, `aptitude` offers solution strategies by way of suitable proposals for installations, updates, or package removals, from which you can pick the one that is most appropriate.

In its default configuration, `aptitude` automatically installs even those packages that a package marks `Recommended:`. This is not always what is wanted and can be switched off from the "Options" menu.

You can install and use `aptitude` on Ubuntu, but it is not the recommended program. For this reason, it does not agree 100% with the graphical tools proposed for package management by Ubuntu—so you should either do everything like Ubuntu recommends, or else do everything using `aptitude`.

12.4 Debian Package Integrity

The `debsums` program is used to check the integrity of the files in a single package (Section 12.2.6). This is nice but does not ensure that an attacker has not manipulated both the files in the package and the `.md5sums` file containing the original checksums. The question remains: How does the Debian project ensure the integrity of complete packages (and, based on this, the integrity of the whole distribution)? This works as follows:

integrity of complete packages

- Every Debian package is cryptographically signed by a Debian developer. This means that the receipient of the package can use the developer's public key to verify that they received the package in the state it was in when the developer released it.

  The Debian developer who signed the package must not necessarily have been the person who assembled it. In principle, every Debian developer may sign and release any package in Debian (a "non-maintainer upload"), and this is being done in practice for timely updates fixing critical security holes and to adopt "orphaned" packages. Furthermore, there are numerous people who help with Debian GNU/Linux and, even though they are not formally Debian developers (or whose applications for developer status are pending), maintain packages. These people cannot by themselves release packages, but must do this via a "sponsor" who must be a Debian developer. The sponsor assumes the responsibility that the package is reasonable.

  ⚠ You should not overestimate the security gained through digital signatures: A developer's signature does not guarantee that there is no malicious code in a package, but only that the developer signed the package. Theoretically it is possible for a cracker to pass the Debian developer accreditation procedure and be able to release official packages into the distribution—whose control scripts most users will execute uncritically as `root`. Most other Linux distributions share the same weaknesses.

- The Debian infrastructure only accepts packages for publication that have been signed by a Debian developer.

- On the Debian server (and all servers mirroring Debian GNU/Linux) there is a file (or several) called `Packages.gz` for each current Debian distribution. This file contains the MD5 checksums of all packages in the distribution, as they are on the server; since the server only accepts packages from accredited developers, these are authentic.

- For each current Debian distribution on the server there is a file called `Release`, which contains the MD5 checksums of the `Packages.gz` file(s) involved. This file is cryptographically signed (the signature is in a file called `Release.gpg`).

With this chain of checksums and signatures, the integrity of packages in the distribution can be checked:

- A new package is downloaded and its MD5 checksum is determined.

- We check whether the signature of the `Release` file is correct, and, if so, read the MD5 checksum of `Packages.gz` from that file.

- With that checksum, we verify the integrity of the actual `Packages.gz` file.
- The MD5 checksum of the package given in `Packages.gz` must match that of the downloaded file.

If the MD5 checksum of the downloaded file does not match the "nominal value" from `Packages.gz`, the administrator is made aware of this fact and the package is not installed (just yet, anyway).

It is possible to configure a Debian system such that it *only* installs packages that can be verified in this way. (Usually all you get is warnings which can be overridden manually.) With this, you can construct an infrastructure where only packages from a considered-safe-and-sensible "subdistribution" can be installed. These packages may be from Debian GNU/Linux or else have been made available locally.

The APT infrastructure only trusts package sources for which a public GnuPG key has been placed in the `/etc/apt/trusted.gpg`. The `apt-key` program is used to maintain this file.

The current public keys for Debian package sources are contained in the `debian-archive-keyring` package and can be renewed by updating this file. (Debian rotates the keys on a yearly basis).

You can find out more about managing signed packages in Debian in [F^{+}07, chapter 7]. We explain GnuPG in the Linup Front training manual *Linux Administration II*.

12.5 The debconf Infrastructure

Sometimes questions come up during the installation of software packages. For example, if you are installing a mail server package, it is important to know, in order to generate an appropriate configuration file, whether the computer in question is connected directly to the Internet, whether it is part of a LAN with its own dedicated mail server, or whether it uses a dial-up connection to access the net. It is also necessary to know the domain the computer is to use for its messages and so on.

The debconf mechanism is designed to collect this information and to store it for future use. It is basically a database for system-wide configuration settings, which can, for example, be accessed by the installation scripts of a package. To manipulate the database, debconf supports modular user interfaces covering all tastes from very simple textual prompts to text-oriented dialogs and various graphical desktop applications such as KDE and GNOME. There are also interfaces to popular programming languages like Python.

You can redo the initial debconf-based configuration of a software package at any time by giving a command like

```
# dpkg-reconfigure my-package
```

`dpkg-reconfigure` repeats the questions asked during the original installation process of the package, using the pre-set user interface.

You can select another user interface on an *ad-hoc* basis by means of the `--frontend` (or `-f`) option. The possible names are given in `debconf`(7) if you have installed the `debconf-doc` package. The default is `dialog`.

To change the user interface temporarily if you are not calling `dpkg-reconfigure` directly, use the `DEBCONF_FRONTEND` environment variable:

```
# DEBCONF_FRONTEND=noninteractive aptitude upgrade
```

With `dpkg-reconfigure`, you can also control the level of detail of the questions you will be asked. Use the `--priority` (or `-p`) option followed by a priority. The possible priorities are (in descending order):

`critical` Questions you absolutely must answer lest terrible things happen.

`high` Questions without a sensible default—your opinion counts.

`medium` Questions with a sensible default.

`low` Trivial questions with a default that works most of the time.

If you say something like

```
# dpkg-reconfigure --priority=medium my-package
```

you will be asked all priority `critical`, `high`, and `medium` questions; any priority `low` questions will be skipped.

For *ad-hoc* changes if debconf is called indirectly, there is also the `DEBCONF_PRIORITY` environment variable.

The debconf infrastructure is fairly complex but useful. For example, it is possible to put the answers into an LDAP database that is accessible to all computers on a network. You can thus install a large number of machines without manual intervention. To explain this in detail would, however, be beyond the scope of this manual.

Exercises

12.11 [1] How can you change the pre-set user interface for debconf on a permanent basis?

12.6 `alien`: Software From Different Worlds

Many software packages are only available in the popular RPM format. Commercial packages in particular are more likely to be offered for the Red Hat or SUSE distributions, even though nothing would prevent anyone from trying the software on Debian GNU/Linux (serious use may be precluded by the loss of manufacturer support if a non-approved platform is used). You cannot install RPM packages on a Debian system directly, but the `alien` program makes it possible to convert the package formats of various Linux distributions—besides RPM also the Stampede and Slackware formats (not that these are desperately required)—to the Debian package format (and vice-versa).

Important: While `alien` will let you convert packages from one format to another, there is no guarantee whatsoever that the resulting package will be useful in any way. On the one hand, the programs in the package may depend on libraries that are not available (or not available in the appropriate version) in the target distribution—since `alien` does not deal with dependencies, you must sort out any problems of this type manually. On the other hand, it is quite possible that the package integrates itself into the source distribution in a way that is impossible or difficult to replicate on the target distribution.

As a matter of principle, the farther “down” a package sits in the system the smaller the probability that `alien` will do what you want. With packages that consist of a few executable programs without bizarre library dependencies, the corresponding manual pages, and a few example files, chances are good for `alien` to do the Right Thing. With system services that must integrate into the system boot sequence, things may well look different. And you should not even *think* of replacing `libc` …

`alien` is used, in particular, to convert LSB-compliant software packages for installation on a Debian GNU/Linux system—LSB specifies RPM as the software distribution package format.

After these introductory remarks, we'll show you quickly how to use `alien` to convert a RPM package to a Debian package:

```
# alien --to-deb paket.rpm
```

(Where `--to-deb` represents the default case and may be left out.) The reverse is possible using

```
# alien --to-rpm paket.deb
```

To assemble and disassemble RPM files, the `rpm` program must be installed (which is available as a package for Debian GNU/Linux); to assemble `deb` packages you need a few pertinent Debian packages which are listed in `alien`(1p). (As mentioned in Section 12.2, you can take `deb` packages to bits on almost all Linux systems using "on-board tools" such as `tar`, `gzip`, and `ar`.)

Commands in this Chapter

alien	Converts various software packaging formats	`alien`(1)	196
apt-get	Powerful command-line tool for Debian GNU/Linux package management	`apt-get`(8)	189
aptitude	Convenient package installation and maintenance tool (Debian)	`aptitude`(8)	192
dpkg	Debian GNU/Linux package management tool	`dpkg`(8)	182
dpkg-reconfigure	Reconfigures an already-installed Debian package	`dpkg-reconfigure`(8)	195

Bibliography

F^{+}07 Javier Fernández-Sanguino Peña, et al. "Securing Debian Manual", 2007.
http://www.debian.org/doc/manuals/securing-debian-howto/

13

Package Management with RPM and YUM

Contents

Goals

- Knowing the basics of RPM and related tools
- Being able to use `rpm` for package management
- Being able to use YUM

Prerequisites

- Knowledge of Linux system administration
- Experience with an RPM-based Linux distribution is helpful

13.1 Introduction

The “Red Hat Package Manager” (RPM, for short) is a tool for managing software packages. It supports the straightforward installation and deinstallation of packages while ensuring that different packages do not conflict and that dependencies between the packages are taken into account. In addition, RPM allows you to specify package queries and to ensure the integrity of packages.

RPM’s core is a database. Software packages add themselves when they are installed and remove themselves again when they are deinstalled. To allow this, software packages must be provided in a specific format, i. e., as RPM packages.

The RPM package format is used used by many distributions (including those by Red Hat, Novell/SUSE, TurboLinux, and Mandriva). An arbitrary RPM package, though, cannot generally be installed on any RPM-based distribution without forethought: Since the RPM package contains compiled software, it must fit the processor architecture in use; since file system structure, the type of service control (init scripts, etc.), and the internal description of dependencies differ between distributions or even between different versions of the same distribution, careless installation across distribution may cause problems.

RPM was originally developed by Red Hat and was accordingly called “Red Hat Package Manager” at first. Since various other distributions have taken to using the program, it has been renamed to “RPM Package Manager”.

At the moment there is a certain controversy as to who is in charge of further development of this critical piece of infrastructure. After a long hiatus, during which nobody really bothered to put out a canonical version, some Fedora developers tried in 2006/7 to restart RPM development as an officially distribution-neutral project (this project is now led by Panu Matilainen of Red Hat, with developers affiliated with some other RPM-using distributions in support). Independently, Jeff Johnson, the last official RPM developer at Red Hat (who is no longer with the company), is putting work into RPM and claims that his code represents “the official code base”—although no Linux distribution seems to pay attention.

file name

An RPM package has a compound file name, for example

```
openssh-3.5p1-107.i586.rpm
```

which usually consists of the package name (`openssh-3.5p1-107`), the architecture (`i586`) and the `.rpm` suffix. The package name is used to identify the package internally once it has been installed. It contains the name of the software (`openssh`) and the software version as assigned by its original developers (`3.5p1`) followed by a release number (`107`) assigned by the package builder (the distributor).

basic mode

The “RPM Package Manager” is invoked using the `rpm` command, followed by a basic mode. The most important modes will be discussed presently, excepting the modes for initialising the RPM database and constructing and signing RPM packages, which are outside the scope of this course.

options

There is a number of global options as well as supplementary, mode-specific options. Since some modes and supplementary options are identical, the mode must (unlike with `tar`) be specified first.

Global options include `-v` and `-vv`, which increase the “verbosity” of RPM’s output.

RPM’s configuration is stored within the `/usr/lib/rpm` directory; local or individual customisations are made within the `/etc/rpmrc` or `~/.rpmrc` files, but should not be necessary for normal operations.

13.2 Package Management Using `rpm`

13.2.1 Installation and Update

An RPM package is installed in `-i` mode, followed by the package file's path name, such as

```
# rpm -i /tmp/openssh-3.5p1-107.i586.rpm
```

You may also specify the path as an HTTP or FTP URL in order to install package files that reside on remote serers. It is permissible to specify several packages at once, as in "`rpm -i /tmp/*.rpm`".

Additionally, there are the two related modes `-U` ("upgrade") and `-F` ("freshen"). The former removes any older versions of a package as well as installing the new one, while the latter installs the package only if an earlier version is already installed (which is subsequently removed).

All three modes support a number of options, which must absolutely be mentioned *after* the mode. Besides `-h` ("hash mark", for a progress bar) there is `--test`, which prevents the actual installation and only checks for possible conflicts.

When a conflict occurs, the package in question is not installed. Conflicts arise if

- an already-installed package is to be installed again,
- a package is to be installed even though it is already installed in a different version (mode `-i`) or a more current version (mode `-U`),
- the installation would overwrite a file belonging to a different package,
- a package requires a different package which is not already installed or about to be installed.

If the installation fails for any of these reasons, you can force it to be performed through options. For example, the `--nodeps` option disables the dependency check.

Further options can influence the installation itself (rather than just the security checks). For example, you can move packages to different directories on installation. This is the only way to install, e. g., Apache 1.3 and Apache 2.0 at the same time, since usually both of them would claim `/sbin/http` for themselves: one of the two must move to `/usr/local`.

13.2.2 Deinstalling Packages

Packages can be deinstalled using `-e` ("erase") mode, e. g.,

```
# rpm -e openssh-3.5p1-107
```

Note that you need to specify the *internal* package name rather than the package file path, since RPM does not remember the latter. (The next section will tell you how to find out the package name.) You can also abbreviate the package name as long as it stays unique. If there is no other `openssh` package, you might also remove it by

```
# rpm -e openssh
```

Again, RPM takes care not to remove packages that other packages depend upon.

The `--test` and `--nodeps` options have the same meaning as upon installation; they must also appear after the mode.

configuration files

When deinstalling, all of the package's installed files will be removed unless they are configuration files that you have changed. These will not be removed but merely renamed by appending the `.rpmsave` suffix. (The RPM package determines which of its files will be considered configuration files.)

13.2.3 Database and Package Queries

The "RPM Package Manager" becomes even more useful if you do not just consider it a package installation and removal tool, but also an information source. The mode for this is `-q` ("query"), and you can specify in more detail what kind of information you would like to obtain and from which package.

Specifying the Package Without further options, `rpm` expects an internal package name, which may be abbreviated, and it outputs the full package name:

```
$ rpm -q openssh
openssh-3.5p1-107
```

This makes it easy to determine how current your system is. You can also find the package claiming a file, using the `-f` option:

```
$ rpm -qf /usr/bin/ssh
openssh-3.5p1-107
```

This lets you relate unknown files to a package. As a third possibility, you can obtain a list of *all* installed packages with the `-a` option:

```
$ rpm -qa
```

This list may of course be processed further, as in the following example:[1]

```
$ rpm -qa | grep cups
cups-client-1.1.18-82
cups-libs-1.1.18-82
kdelibs3-cups-3.1.1-13
cups-drivers-1.1.18-34
cups-drivers-stp-1.1.18-34
cups-1.1.18-82
```

Finally, RPM allows you to query a non-installed package. Use `-p` followed by the package file's name:

```
$ rpm -qp /tmp/openssh-3.5p1-107.i586.rpm
openssh-3.5p1-107
```

This does not look too spectacular, since the internal package name was already part of the package file name. But the file name might have been changed until it no longer had anything to do with the actual package name, and secondly, there are other questions that you might want to ask.

Specifying the Query If you are not just interested in the package name, you can extend your query. Every extension may be combined with every way of specifying a packet. Via

```
$ rpm -qi openssh
```

you can obtain detailed information (`-i`) about the package; while `-l` provides a list of all files belonging to the package, together with `-v` it forms an equivalent to the `ls -l` command:

[1] The naming and arrangement of packages is the package preparer's concern; differences may occur depending on the distribution and version.

```
$ rpm -qlf /bin/bash
/bin/bash
/bin/sh
◁◁◁◁◁
$ rpm -qlvf /bin/bash
-rwxr-xr-x  root  root   491992 Mar 14  2003 /bin/bash
lrwxrwxrwx  root  root        4 Mar 14  2003 /bin/sh -> bash
◁◁◁◁◁
```

It is important to note that the files listed for a package are only those that show up in the RPM database, namely those that a package brought along when it was installed. This does not include files that were generated during installation (by the package's installation scripts) or during system operation (log files, etc.).

We have already seen that RPM treats configuration files specially (when deinstalling). The second class of special files are documentation files; these can be omitted from installation. The `-c` and `-d` options of query mode behave like `-l`, but they confine themselves to configuration and documentation files, respectively.

Advanced Queries The following details are not relevant for LPIC-1, but they will improve your understanding of RPM's concepts and database structure.

dependencies

Dependencies between packages can belong to various types. For example, a package may simply require a shell, i. e., `/bin/sh`. By means of

```
$ rpm -qf /bin/sh
bash-2.05b-105
```

it is straightforward to find out that this file is provided by the `bash` package (the same can be done for non-installed packages).

Things are different, e. g., for the SAINT package, a security analysis tool which requires a web browser. Every specific dependency on a particular web browser would be unduly limiting. For this reason, RPM lets packages provide or depend upon "capabilities". In our example, SAINT requires the abstract capability "web browser". The files and capabilities that a package requires can be queried using the `--requires` option:[2]

```
$ rpm -q --requires saint
web_browser
/bin/rm
/bin/sh
/usr/bin/perl
◁◁◁◁◁
```

The packages providing these capabilities can be found using the `--whatprovides` option:

```
$ rpm -q --whatprovides web_browser
w3m-0.3.2.2-53
mozilla-1.2.1-65
lynx-2.8.4-391
```

For SAINT, you need just one of these packages.

In the same manner, the `--provides` and `--whatrequires` options allow you to query the services (or files, with the `-l` option) that a package offers, and a service's consumers.

[2] Here, again, the assignment and naming of capabilities is up to the package preparer; it may thus differ between distributions and versions.

13.2.4 Package Verification

Pre-Installation Checks Two things may happen to a package which might preclude its installation: It may have been damaged during the download, i. e., the package is erroneous. Or the package is not what it pretends to be, i. e., it has been falsified—for example, because some malicious person tries to pass a "Trojan" package off as the original.

RPM safeguards you against both scenarios: with

```
$ rpm --checksig /tmp/openssh-3.5p1-107.i586.rpm
/tmp/openssh-3.5p1-107.i586.rpm: md5 gpg OK
```

an MD5 checksum of the package is compared to the checksum contained within itself, which guarantees the proper transmission of the package. In addition, the signature within the package, which was created using the private PGP or GPG key of the package preparer, is checked using the package preparer's public key. This guarantees that the correct package has arrived.

Should the MD5 checksum be correct but not the signature, the output looks correspondingly different:

```
$ rpm --checksig /tmp/openssh-3.5p1-107.i586.rpm
/tmp/openssh-3.5p1-107.i586.rpm: md5 GPG NOT OK
```

Of course your distributor's public key must be available on your system for the signature checks.

Post-Installation Verification RPM lets you compare certain values within the RPM database to the file system. This is done by means of the -V ("verify") mode; instead of one or more internal package names, this mode can use all specifications made available for the query mode.

```
# rpm -V openssh
.......T c /etc/init.d/sshd
S.5....T c /etc/pam.d/sshd
S.5....T c /etc/ssh/ssh_config
SM5....T c /etc/ssh/sshd_config
.M......   /usr/bin/ssh
```

This output contains all files for which at least one "required" value from the database differs from the "actual" value within the file system: a "." signifies agreement, while a letter indicates a deviation. The following checks are performed: access mode and file type (M), owner and group (U, G); for symbolic links, the path of the referenced file (L); for device files, major and minor device numbers (D); for plain files the size (S), modification time (T), and content (5). Since configuration files are unlikely to remain in their original state, they are labeled with a c.

Even though the verification of installed packages using RPM cannot replace an "intrusion detection system" (why should an intruder not modify the RPM database as well?), it can be useful to limit the damage, e. g., after a hard disk crash.

13.2.5 The rpm2cpio Program

RPM packages are essentially cpio archives with a prepended "header". You can use this fact to extract individual files from an RPM package without having to install the package first. Simply convert the RPM package to a cpio archive using the rpm2cpio program, and feed the archive into cpio. Since rpm2cpio works as a filter, you can easily connect the two programs using a pipe:

```
$ rpm2cpio hello-2.4-1.fc10.i386.rpm \
> | cpio -idv ./usr/share/man/man1/hello.1.gz
./usr/share/man/man1/hello.1.gz
387 blocks
$ zcat usr/share/man/man1/hello.1.gz | head
.\" DO NOT MODIFY THIS FILE!  It was generated by help2man 1.35.
.TH HELLO "1" "December 2008" "hello 2.4" "User Commands"
.SH NAME
hello \- friendly greeting program
◁◁◁◁◁
```

Exercises

13.1 [2] Use rpm2cpio and cpio to display the list of files contained in an RPM package.

13.3 YUM

13.3.1 Overview

The rpm program is useful but does have its limits. As a basic tool it can install packages that are available as files or URLs, but, for example, does not help with locating appropriate, possibly installable packages. Many RPM-based distributions use YUM (short for “Yellow Dog Updater, Modified”, after the distribution for which the program was originally developed) to enable access to package sources (repositories) available on the Internet or on CD-ROM.

package sources

In RPM-based distributions, YUM takes up approximately the same “ecological niche” occupied by apt-get in Debian GNU/Linux and its derivatives.

YUM is usually controlled via the command line, but the “yum shell” command starts a “YUM shell” where you can enter multiple YUM commands interactively.

13.3.2 Package Repositories

YUM introduces the concept of *package repositories*. A package repository is a set of RPM packages that is available via the network and allows the installation of packages with YUM. The “yum repolist” command outputs a list of configured package repositories:

```
$ yum repolist
Loaded plugins: refresh-packagekit
repo id       repo name                 status
fedora        Fedora 10 - i386          enabled: 11416
updates       Fedora 10 - i386 - Updates enabled:  3324
repolist: 14740
```

“yum repolist disabled” yields a list of known but disabled repositories:

```
$ yum repolist disabled
Loaded plugins: refresh-packagekit
repo id          repo name                    status
fedora-debuginfo Fedora 10 - i386 - Debug     disabled
fedora-source    Fedora 10 - Source           disabled
```

```
rawhide           Fedora - Rawhide - Development disabled
◁◁◁◁◁
```

To enable a repository, you need to give the `--enablerepo=` option, followed by the “repo ID” from the list. This only works in connection with a “genuine” `yum` command; `repolist` is fairly innocuos:

```
$ yum --enablerepo=rawhide repolist
Loaded plugins: refresh-packagekit
rawhide                                      | 3.4 kB   00:00
rawhide/primary_db                           | 7.2 MB   00:14
repo id        repo name                       status
fedora         Fedora 10 - i386                enabled: 11416
rawhide        Fedora - Rawhide - Development  enabled: 12410
updates        Fedora 10 - i386 - Updates      enabled:  3324
repolist: 27150
```

You can disable a repository using the `--disablerepo` option.

Repositories are most conveniently made known to YUM by means of configuration files in the `/etc/yum.repos.d` directory. (You could also enter them into `/etc/yum.conf` directly, but this is more inconvenient to manage.)

YUM keeps itself current as far as the content of repositories is concerned. There is no equivalent to the Debian tools’ “`apt-get update`”.

13.3.3 Installing and Removing Packages Using YUM

To install a new package using YUM, you merely need to know its name. YUM checks whether the active repositories contain an appropriately-named package, resolves any dependencies the package may have, downloads the package and possibly other packages that it depends upon, and installs all of them:

```
# yum install hello
Setting up Install Process
Parsing package install arguments
Resolving Dependencies
--> Running transaction check
---> Package hello.i386 0:2.4-1.fc10 set to be updated
--> Finished Dependency Resolution

Dependencies Resolved

=============================================================
 Package       Arch      Version         Repository     Size
=============================================================
Installing:
 hello         i386      2.4-1.fc10      updates        68 k

Transaction Sum
=============================================================
Install      1 Package(s)
Update       0 Package(s)
Remove       0 Package(s)

Total download size: 68 k
Is this ok [y/N]: y
Downloading Packages:
hello-2.4-1.fc10.i386.rpm                    |  68 kB     00:00
```

```
===================== Entering rpm code =======================
Running rpm_check_debug
Running Transaction Test
Finished Transaction Test
Transaction Test Succeeded
Running Transaction
  Installing     : hello                                    1/1
===================== Leaving rpm code ========================

Installed:
  hello.i386 0:2.4-1.fc10

Complete!
```

YUM accepts not just simple package names, but also package names with architecture specifications, version numbers, and release numbers. Check yum(8) to find the allowable formats.

Removing packages is just as simple:

```
# yum remove hello
```

This will also remove packages that this package depends upon—as long as these are not required by another installed package, anyway.

Instead of "yum remove" you can also say "yum erase"—the two are equivalent.

You can update packages using "yum update":

```
# yum update hello
```

checks whether a newer version of the package is available and installs that if this is the case. YUM takes care that all dependencies are resolved. "yum update" without a package name attempts to update all installed packages.

When the --obsoletes is specified (the default case), yum tries to handle the case where one package has been replaced by another (of a different name). This makes full upgrades of the whole distribution easier to perform.

"yum upgrade" is the same as "yum update --obsoletes"—but saves some typing in the case that you have switched off the obsoletes option in the configuration.

YUM supports the idea of "package groups", i. e., packages that together are useful for a certain task. The available package groups can be displayed using "yum grouplist":

```
$ yum grouplist
Loaded plugins: refresh-packagekit
Setting up Group Process
Installed Groups:
   Administration Tools
   Authoring and Publishing
   Base
   Dial-up Networking Support
   Editors
◁◁◁◁◁
```

If you want to know which packages a group consists of, use "yum groupinfo":

```
$ yum groupinfo 'Printing Support'
Loaded plugins: refresh-packagekit
Setting up Group Process

Group: Printing Support
 Description: Install these tools to enable the system to▷
 ◁ print or act as a print server.
 Mandatory Packages:
   cups
   ghostscript
 Default Packages:
   a2ps
   bluez-cups
   enscript
◁◁◁◁◁
```

A group is considered "installed" if all its "mandatory" packages are installed. Besides these there are "default packages" and "optional packages".

"yum groupinstall" lets you install the packages of a group. The configuration option group_package_types determines which class package will actually be installed—usually the "mandatory" and the "default packages".

"yum groupremove" removes all packages of a group, *without* taking into account package classes (group_package_types is ignored). Note that packages can belong to more than one group at the same time, so they may be missing from group X after having been removed along with group Y.

13.3.4 Information About Packages

The "yum list" command is available to find out which packages exist:

```
$ yum list gcc
Loaded plugins: refresh-packagekit
Installed Packages
gcc.i386            4.3.2-7             installed
```

You can also give a search pattern (it is best to put it inside quotes so the shell will not mess with it):

```
$ yum list "gcc*"
Loaded plugins: refresh-packagekit
Installed Packages
gcc.i386            4.3.2-7             installed
gcc-c++.i386        4.3.2-7             installed
Availabe Packages
gcc-gfortran.i386   4.3.2-7             fedora
gcc-gnat.i386       4.3.2-7             fedora
◁◁◁◁◁
```

The "installed packages" are installed on the local system, while the "available packages" can be fetched from repositories. The repository offering the package is displayed on the far right.

To restrict the search to locally installed, or uninstalled but available, packages, you can use "yum list installed" or "yum list available":

```
$ yum list installed "gcc*"
Loaded plugins: refresh-packagekit
```

```
Installed Packages
gcc.i386            4.3.2-7               installed
gcc-c++.i386        4.3.2-7               installed
$ yum list available "gcc*"
Loaded plugins: refresh-packagekit
Available Packages
gcc-gfortran.i386     4.3.2-7             fedora
gcc-gnat.i386       4.3.2-7               fedora
⊲⊲⊲⊲⊲
```

"yum list updates" lists the packages that are installed and for which updates are available, while "yum list recent" lists the packages that have "recently" arrived in a repository. "yum list extras" points out packages that are installed locally but are *not* available from any repository.

To find out more about a package, use "yum info":

```
$ yum info hello
Loaded plugins: refresh-packagekit
Installed Packages
Name       : hello
Arch       : i386
Version    : 2.4
Release    : 1.fc10
Size       : 186 k
Repo       : installed
Summary    : Prints a Familiar, Friendly Greeting
URL        : http://www.gnu.org/software/hello/
License    : GPLv3+ and GFDL and BSD and Public Domain
Description: Hello prints a friendly greeting. It also serves as a
           : sample GNU package, showing practices that may be
           : useful for GNU projects.
```

The advantage over "rpm -qi" is that "yum info" also works for packages that are not installed locally but are available from a repository.

You con otherwise use "yum info" like "yum list"—"yum info installed", for example, displays detailed information about *all* installed packages.

Using "yum search", you can search for all packages in whose name or description a given string occurs:

```
$ yum search mysql
Loaded plugins: refresh-packagekit
============================ Matched: mysql ========================
dovecot-mysql.i386 : MySQL backend for dovecot
koffice-kexi-driver-mysql.i386 : Mysql-driver for kexi
libgda-mysql.i386 : MySQL provider for libgda
⊲⊲⊲⊲⊲
```

Unfortunately, the resulting list is unsorted and a little difficult to read. yum uses boldface to emphasise the places where the search string occurs.

You can examine a package's dependencies using "yum deplist":

```
$ yum deplist gcc
Loaded plugins: refresh-packagekit
Finding dependencies:
package: gcc.i386 4.3.2-7
```

```
  dependency: binutils >= 2.17.50.0.17-3
   provider: binutils.i386 2.18.50.0.9-7.fc10
  dependency: libc.so.6(GLIBC_2.3)
   provider: glibc.i386 2.9-2
◁◁◁◁◁
```

13.3.5 Downloading Packages

If you want to download a package from a repository but do not want to install it outright, you can use the `yumdownloader` program. A command like

```
$ yumdownloader --destdir /tmp hello
```

searches the repositories for the `hello` package just like YUM would and downloads the corresponding file to the `/tmp` directory.

The `--resolve` option causes dependencies to be resolved and any other missing packages to be downloaded as well—but only those that are not installed on the local system.

With the `--urls` option, nothing is downloaded at all, but `yumdownloader` outputs the URLs of the packages it would otherwise have downloaded.

With the `--source` option, `yumdownloader` downloads source RPMs instead of binary RPMs.

Commands in this Chapter

cpio	File archive manager	`cpio`(1)	204
rpm	Package management tool used by various Linux distributions (Red Hat, SUSE, …)	`rpm`(8)	200
rpm2cpio	Converts RPM packages to `cpio` archives	`rpm2cpio`(1)	204
yum	Convenient RPM package maintenance tool	`yum`(8)	205

Summary

- RPM is a system for Linux software package management which is used by various distributions such as those by Red Hat and Novell/SUSE.
- YUM is a front-end for `rpm` that gives access to package repositories over the network.

A

Sample Solutions

This appendix contains sample solutions for selected exercises.

1.1 Access control applies to normal users but not the administrator. `root` may do anything! The `root` account should only be used to execute commands that really require `root`'s privileges, e. g., to partition the disk, to create file systems, to add user accounts, or to change system configuration files. All other actions should be performed from an unprivileged account. This includes invoking administration commands to gather information (where possible) and unpacking `tar` archives.

1.2 As `root` you may do anything, therefore it is very easy to damage the system, e. g., through inadvertently mistyped commands.

1.3 This question aims at a comparison to other operating systems. Depending on the system in question, there are no access controls at all (DOS, Windows 95/98) or different methods for access control (Windows NT/2000/XP or Windows Vista). Accordingly, the former do not support administrator access (as opposed to normal user access), while the latter even allow the creation of several administrator accounts.

1.4 Basically you can log in as `root`, or create a UID 0 shell using `su`. The latter method is better, e. g., because the change including the former UID is logged.

1.5 The shell prompt often looks different. In addition, the `id` command may help.

1.6 You can either log in directly or `su`. For frequent changes, it is a good idea to log in on two consoles at the same time, and obtain a `root` shell using `su` on one. Alternatively, you could open several terminal windows on a GUI.

1.7 You will generally find the log entry in `/var/log/messages`.

1.10 The obvious advantage is that administration is possible from anywhere, if necessary by using an internet-enabled cell phone on the beach, and without having to have access to specialised hardware or software. The obvious disadvantage is that you need to secure access to the administration tool very carefully, in order to prevent unbidden guests to "misconfigure" your system (or worse). This may imply that (the obvious advantage notwithstanding) you may be able to provide the administration tool only from within the local network, or that you should secure access to it using strong cryptography (e. g., SSL with client certificates). If you consider deploying Webmin in your company, you should discuss the possibility of external access *very carefully* with the appropriate decision makers and/or corporate data protection officers in order to avoid extremely dire consequences that could hit you if problems appear. Consider yourself warned.

2.1 By their respective numerical UIDs and GIDs.

2.2 This works but is not necessarily a good idea. As far as the system is concerned, the two are a single user, i. e., all files and processes with that UID belong to both user names.

2.3 A pseudo-user's UID is used by programs in order to obtain particular well-defined access rights.

2.4 Whoever is in group `disk` has block-level read and write permission to the system's disks. With knowledge of the file system structure it is easy to make a copy of `/bin/sh` into a SUID `root` shell (section 3.5) by changing the file system metadata directly on disk. Thus, group `disk` membership is tantamount to `root` privileges; you should put nobody into the `disk` group whom you would not want to tell the `root` password outright.

2.5 You will usually find an "`x`". This is a hint that the password that would usually be stored there is indeed stored in another file, namely `/etc/shadow`, which unlike the former file is readable only for `root`.

2.6 There are basically two possibilities:

1. Nothing. In this case the system should turn you away after you entered your password, since no user account corresponds to the all-uppercase user name.
2. From now on, the system talks to you in uppercase letters only. In this case your Linux system assumes that you are sitting in front of an absolutely antediluvial terminal (1970s vintage or so) that does not support lowercase letters, and kindly switches its processing of input and output data such that uppercase letters in the input are interpreted as lowercase, and lowercase letters in the output are displayed as uppercase. Today this is of limited benefit (except if you work in a computer museum), and you should log out as quickly again as possible before your head explodes. Since this behaviour is so atavistic, not every Linux distribution goes along with it, though.

2.7 Use `getent`, `cut`, and `sort` to generate lists of user names for the databases, and `comm` to compare the two lists.

2.8 Use the `passwd` command if you're logged in as user `joe`, or "`passwd joe`" as `root`. In `joe`'s entry in the `/etc/shadow` file there should be a different value in the second field, and the date of the last password change (field 3) should show the current date (in what unit?)

2.9 As `root`, you set a new password for him using "`passwd dumbo`", as you cannot retrieve his old one even though you are the administrator.

2.10 Use the command "`passwd -n 7 -x 14 -w 2 joe`". You can verify the settings using "`passwd -S joe`".

2.11 Use the `useradd` command to create the user, ”`usermod -u`” to modify the UID. Instead of a user name, the files should display a UID as their owner, since no user name is known for that UID …

2.12 For each of the three user accounts there should be one line in `/etc/passwd` and one in `/etc/shadow`. To work with the accounts, you do not necessarily need a password (you can use `su` as `root`), but if you want to login you do. You can create a file without a home directory by placing it in `/tmp` (in case you forgot—a home directory for a new user would however be a good thing).

2.13 Use the `userdel` command to delete the account. To remove the files, use the ”`find / -uid` ⟨*UID*⟩ `-delete`” command.

2.14 If you use ”`usermod -u`”, you must reassign the user's file to the new UID, for example by means of ”`find / -uid` ⟨*UID*⟩ `-exec chown test1 {} \;`” or (more efficiently) ”`chown -R --from=`⟨*UID*⟩ `test1 /`”. In each case, ⟨*UID*⟩ is the (numerical) former UID.

2.15 You can either edit `/etc/passwd` using `vipw` or else call `usermod`.

2.16 Groups make it possible to give specific privileges to groups [sic!] of users. You could, for example, add all HR employees to a single group and assign that group a working directory on a file server. Besides, groups can help organise access rights to certain peripherals (e. g., by means of the groups `disk`, `audio`, or `video`).

2.17 Use the ”`mkdir` ⟨*directory*⟩” command to create the directory and ”`chgrp` ⟨*groupname*⟩ ⟨*directory*⟩” to assign that directory to the group. You should also set the SGID bit to ensure that newly created files belong to the group as well.

2.18 Use the following commands:

```
# groupadd test
# gpasswd -a test1 test
Adding user test1 to group test
# gpasswd -a test2 test
Adding user test2 to group test
# gpasswd test
Changing the password for group test
New Password:x9q.Rt/y
Re-enter new password:x9q.Rt/y
```

To change groups, use the ”`newgrp test`” command. You will be asked for the password only if you are not a member of the group in question.

3.1 A new file is assigned to your current primary group. You can't assign a file to a group that you are not a member of—unless you are `root`.

3.3 077 and ”‘`u=rwx,go=`”’, respectively.

3.5 This is the SUID or SGID bit. The bits cause a process to assume the UID/GID of the executable file rather than that of the executing user. You can see the bits using “`ls -l`”. Of course you may change all the permissions on your own files. However, at least the SUID bit only makes sense on binary executable files, not shell scripts and the like.

3.6 One of the two following (equivalent) commands will serve:

```
$ umask 007
$ umask -S u=rwx,g=rwx
```

You may perhaps ask yourself why this umask contains `x` bits. They are indeed irrelevant for files, as files are not created executable by default. However it might be the case that subdirectories are desired in the project directory, and it makes sense to endow these with permissions that allow them to be used reasonably.

3.7 The so-called "sticky bit" on a directory implies that only the owner of a file (or the owner of the directory) may delete or rename it. You will find it, e. g., on the `/tmp` directory.

3.9 This doesn't work with the `bash` shell (at least not without further trickery). We can't speak for other shells here.

3.11 You cannot do this with `chattr` alone, since various attributes can be displayed with `lsattr` but not set with `chattr`. Read up on the details in `chattr`(1).—In addition, some attributes are only defined for "plain" files while others are only defined for directories; you will, for example, find it difficult to make the `D` and `E` attributes visible for the same "file system object" at the same time. (The `E` attribute is to do with transparent compression, which cannot be used on directories, while `D` only applies to directories—write operations to such directories will be performed synchronously.)

4.1 In the directory of a process below `/proc` there is a file called `environ` which contains the environment variables of that process. You can output this file using `cat`. The only blemish is that the variables in this file are separated using zero bytes, which looks messy on the screen; for convenience, you might use something like "`tr "\0" "\n" </proc/4711/environ`" to display the environment.

4.2 Funnily enough, the limit is not documented in any obvious placees. In `/usr/include/linux/threads.h` on a Linux 2.6 kernel, the constant `PID_MAX_LIMIT` is defined with a value of 32768; this is the lowest value that will by default *not* be assigned to processes. You can query the actual value in `/proc/sys/kernel/pid_max` (or even change it—the maximum for 32-bit platforms is actually 32768, while on 64-bit systems you may set an arbitrary value of up to 2^{22}, which is approximately 4 million).

The PIDs assigned to processes rise monotonically at first. When the above-mentioned limit is reached, assignment starts again with lower PIDs, where PIDs that are still being used by processes are of course not given again to others. Many low PIDs are assigned to long-running daemons during the boot process, and for this reason after the limit has been reached, the search for unused PIDs starts again not at PID 1 but at PID 300. (The details are in the `kernel/pid_namespace.c` file within the Linux source code.)

4.4 As we said, zombies arise when the parent process does not pick up the return code of a child process. Thus, to create a zombie you must start a child process and then prevent the parent process from picking up its return code, for example by stopping it by means of a signal. Try something like

```
$ sh
$ echo $$                                                    In the subshell
12345
$ sleep 20
                                                         In a different window:
```

```
$ kill -STOP 12345
                                                                Wait
$ ps u | grep sleep
joe 12346 0.0  0.0  3612  456 pts/2  Z 18:19 0:00 sleep 20
```

4.5 Consult `ps`(1).

4.6 Try

```
$ ps -o pid,ppid,state,cmd
```

4.7 Usually `SIGCHLD` ("child process finished"—sometimes called `SIGCLD`), `SIGURG` (urgent data was received on a network connection) and `SIGWINCH` (the size of the window for a text-based program was changed). These three events are so inane that the process should not be terminated on their account.

4.8 Something like

```
$ pgrep -u hugo
```

should suffice.

4.10 Use, e. g., the "`renice -10` ⟨*PID*⟩" command. You can only specify negative nice values as `root`.

6.2 `sda1`, `sda2`, `sda5`, `sda6`, and `sdb1`, `sdb5`, `sdb6`, `sdb7`.

7.2 Use `tune2fs` with the `-c`, `-u` and `-m` options.

7.3 `mkreiserfs /dev/sdb5`

7.6 `/etc/fstab` contains all frequently-used file systems and their mount points, while `/etc/mtab` contains those file systems that are actually mounted at the moment.

8.1 The boot loader can be placed inside the MBR, in another (partition) boot sector, or a file on disk. In the two latter cases, you will need another boot loader that can "chain load" the Linux boot loader. Be that as it may, for a Linux system you absolutely need a boot loader that can boot a Linux kernel, such as GRUB (there are others).

8.3 Assign a password preventing the unauthorised entry of kernel parameters. With GRUB Legacy, e. g., using

```
password --md5 ⟨encrypted keyword⟩
```

`lock` helps with the password request for a specific operating system.

9.3 You can display the previous and current runlevel using `runlevel`. If the previous runlevel is "`N`" that means "none"—the system started into the current runlevel. To change, say "`init 2`", then "`runlevel`" again to check.

9.4 A possible entry for the `inittab` file might be

```
aa:A:ondemand:/bin/date >/tmp/runlevel-a.txt
```

This entry should write the current time to the mentioned file if you activate it using “`telinit A`”. Don’t forget the “`telinit q`” to make `init` reread its configuration file.

9.5 Call the `syslog` init script with the `restart` or `reload` parameters.

9.6 For example, by using “`chkconfig -l`” (on a SUSE or Red Hat system).

9.7 It is tempting just to remove the symbolic links from the runlevel directory in question. However, depending on the distribution, they may reappear after the next automated change. So if your distribution uses a tool like `chkconfig` or `insserv` you had better use that.

9.8 You should be prepared for the system asking for the `root` password.

9.10 Use the

```
# shutdown -h +15 'This is just a test'
```

command; everything that you pass to `shutdown` after the delay will be sent to your users as a broadcast message. To cancel the shutdown, you can either interrupt the program using the Ctrl+c key combination (if you started `shutdown` in the foreground), or give the “`shutdown -c`” command.

9.10 The file name will be sent as the message.

10.4 The unit file does not need to be modified in order to express dependencies. This makes the automatic installation and, in particular, deinstallation of units as part of software packages easier (e. g., in the context of a distribution-specific package management tool) and allows the seamless updating of unit files by a distribution.

10.10 There is no exact equivalent because systemd does not use the runlevel concept. You can, however, display all currently active targets:

```
# systemctl list-units -t target
```

10.11 “`systemctl kill`” guarantees that the signal will only be sent to processes belonging to the unit in question. The other two commands send the signal to all processes whose name happens to be `example`.

10.13 You can’t (“`systemctl mask`” outputs an error message). You must deactivate the service and then remove, move, or rename the unit file.

11.3 You can find out using a shell script like

```
for f in /usr/bin/*
do
    printf "%4u %s\n" $(ldd $f 2>/dev/null | wc -l) $f
done | sort -nr | head
```

(the `2>/dev/null` suppresses the error message that `ldd` outputs if you feed it a shell script rather than an executable program.) On the author’s system, `mplayer` is at the top of the list, sporting an impressive 118 libraries.

11.4 This would be a major security hole, since you could attempt to use a private directory in LD_LIBRARY_PATH to sneak a "Trojan horse" version of a common library into the program. For example, you can assume with near certainty that a program such as passwd will call a C function like open(). If you should succeed in arranging for this function to be called not from libc but your own library, you would essentially be able to do whatever you wanted using root privileges. This of course is not desirable (in the greater scheme of things, anyway).

12.11 Try

```
# dpkg-reconfigure debconf
```

13.1 This is most easily done using something like

```
$ rpm2cpio ⟨package⟩ | cpio -t
```

B

LPIC-1 Certification

B.1 Overview

The *Linux Professional Institute* (LPI) is a vendor-independent non-profit organization dedicated to furthering the professional use of Linux. One aspect of the LPI's work concerns the creation and delivery of distribution-independent certification exams, for example for Linux professionals. These exams are available world-wide and enjoy considerable respect among Linux professionals and employers.

Through LPIC-1 certification you can demonstrate basic Linux skills, as required, e. g., for system administrators, developers, consultants, or user support professionals. The certification is targeted towards Linux users with 1 to 3 years of experience and consists of two exams, LPI-101 and LPI-102. These are offered as computer-based multiple-choice and fill-in-the-blanks tests in all Pearson VUE and Thomson Prometric test centres. On its web pages at `http://www.lpi.org/`, the LPI publishes **objectives** outlining the content of the exams.

objectives

This training manual is part of Linup Front GmbH's curriculum for preparation of the LPI-101 exam and covers part of the official examination objectives. Refer to the tables below for details. An important observation in this context is that the LPIC-1 objectives are not suitable or intended to serve as a didactic outline for an introductory course for Linux. For this reason, our curriculum is not strictly geared towards the exams or objectives as in "Take classes x and y, sit exam p, then take classes a and b and sit exam q." This approach leads many prospective students to the assumption that, being complete Linux novices, they could book n days of training and then be prepared for the LPIC-1 exams. Experience shows that this does not work in practice, since the LPI exams are deviously constructed such that intensive courses and exam-centred "swotting" do not really help.

Accordingly, our curriculum is meant to give you a solid basic knowledge of Linux by means of a didactically reasonable course structure, and to enable you as a participant to work independently with the system. LPIC-1 certification is not a primary goal or a goal in itself, but a natural consequence of your newly-obtained knowledge and experience.

B.2 Exam LPI-101

The following table displays the objectives for the LPI-101 exam and the materials covering these objectives. The numbers in the columns for the individual manuals refer to the chapters containing the material in question.

No	Wt	Title	GRD1	ADM1
101.1	2	Determine and configure hardware settings	–	5–6
101.2	3	Boot the system	–	8–10
101.3	3	Change runlevels/boot targets and shutdown or reboot system	–	9–10
102.1	2	Design hard disk layout	–	6
102.2	2	Install a boot manager	–	8
102.3	1	Manage shared libraries	–	11
102.4	3	Use Debian package management	–	12
102.5	3	Use RPM and YUM package management	–	13
103.1	4	Work on the command line	3–4	–
103.2	3	Process text streams using filters	8	–
103.3	4	Perform basic file management	6, 11	7.3
103.4	4	Use streams, pipes and redirects	8	–
103.5	4	Create, monitor and kill processes	–	4
103.6	2	Modify process execution priorities	–	4
103.7	2	Search text files using regular expressions	7–8	–
103.8	3	Perform basic file editing operations using vi	5, 7	–
104.1	2	Create partitions and filesystems	–	6–7
104.2	2	Maintain the integrity of filesystems	–	7
104.3	3	Control mounting and unmounting of filesystems	–	7
104.4	1	Manage disk quotas	–	7.4
104.5	3	Manage file permissions and ownership	–	3
104.6	2	Create and change hard and symbolic links	6	–
104.7	2	Find system files and place files in the correct location	6, 10	–

B.3 Exam LPI-102

The following table displays the objectives for the LPI-102 exam and the materials covering these objectives. The numbers in the columns for the individual manuals refer to the chapters containing the material in question.

No	Wt	Title	ADM1	GRD2	ADM2
105.1	4	Customize and use the shell environment	–	1–2	–
105.2	4	Customize or write simple scripts	–	2–5	–
105.3	2	SQL data management	–	8	–
106.1	2	Install and configure X11	–	11	–
106.2	1	Setup a display manager	–	11	–
106.3	1	Accessibility	–	12	–
107.1	5	Manage user and group accounts and related system files	2	–	–
107.2	4	Automate system administration tasks by scheduling jobs	–	9	–
107.3	3	Localisation and internationalisation	–	10	–
108.1	3	Maintain system time	–	–	8
108.2	3	System logging	–	–	1–2
108.3	3	Mail Transfer Agent (MTA) basics	–	–	11
108.4	2	Manage printers and printing	–	–	9
109.1	4	Fundamentals of internet protocols	–	–	3–4
109.2	4	Basic network configuration	–	–	4–5, 7
109.3	4	Basic network troubleshooting	–	–	4–5, 7
109.4	2	Configure client side DNS	–	–	4
110.1	3	Perform security administration tasks	2	–	4–5, 13
110.2	3	Setup host security	2	–	4, 6–7, 13
110.3	3	Securing data with encryption	–	–	10, 12

B.4 LPI Objectives In This Manual

101.1 Determine and configure hardware settings

Weight 2

Description Candidates should be able to determine and configure fundamental system hardware.

Key Knowledge Areas

- Enable and disable integrated peripherals
- Configure systems with or without external peripherals such as keyboards
- Differentiate between the various types of mass storage devices
- Know the differences between coldplug and hotplug devices
- Determine hardware resources for devices
- Tools and utilities to list various hardware information (e.g. `lsusb`, `lspci`, etc.)
- Tools and utilities to manipulate USB devices
- Conceptual understanding of sysfs, udev, dbus

The following is a partial list of the used files, terms and utilities:

- `/sys/`
- `/proc/`
- `/dev/`
- `modprobe`
- `lsmod`
- `lspci`
- `lsusb`

101.2 Boot the system

Weight 3

Description Candidates should be able to guide the system through the booting process.

Key Knowledge Areas

- Provide common commands to the boot loader and options to the kernel at boot time
- Demonstrate knowledge of the boot sequence from BIOS to boot completion
- Understanding of SysVinit and systemd
- Awareness of Upstart
- Check boot events in the log files

The following is a partial list of the used files, terms and utilities:

- `dmesg`
- BIOS
- bootloader
- kernel
- initramfs
- `init`
- SysVinit
- systemd

101.3 Change runlevels/boot targets and shutdown or reboot system

Weight 3

Description Candidates should be able to manage the SysVinit runlevel or systemd boot target of the system. This objective includes changing to single user mode, shutdown or rebooting the system. Candidates should be able to alert users before switching runlevels/boot targets and properly terminate processes. This objective also includes setting the default SysVinit runlevel or systemd boot target. It also includes awareness of Upstart as an alternative to SysVinit or systemd.

Key Knowledge Areas

- Set the default runlevel or boot target
- Change between runlevels/boot targets including single user mode
- Shutdown and reboot from the command line
- Alert users before switching runlevels/boot targets or other major system events
- Properly terminate processes

The following is a partial list of the used files, terms and utilities:

- `/etc/inittab`
- `shutdown`
- `init`
- `/etc/init.d/`
- `telinit`
- systemd
- `systemctl`
- `/etc/systemd/`
- `/usr/lib/systemd/`
- `wall`

102.1 Design hard disk layout

Weight 2

Description Candidates should be able to design a disk partitioning scheme for a Linux system.

Key Knowledge Areas

- Allocate filesystems and swap space to separate partitions or disks
- Tailor the design to the intended use of the system
- Ensure the `/boot` partition conforms to the hardware architecture requirements for booting
- Knowledge of basic features of LVM

The following is a partial list of the used files, terms and utilities:

- `/` (root) filesystem
- `/var` filesystem
- `/home` filesystem
- `/boot` filesystem
- swap space
- mount points
- partitions

102.2 Install a boot manager

Weight 2

Description Candidates should be able to select, install and configure a boot manager.

Key Knowledge Areas

- Providing alternative boot locations and backup boot options

- Install and configure a boot loader such as GRUB Legacy
- Perform basic configuration changes for GRUB 2
- Interact with the boot loader

The following is a partial list of the used files, terms and utilities:

- `menu.lst`, `grub.cfg` and `grub.conf`
- `grub-install`
- `grub-mkconfig`
- MBR

102.3 Manage shared libraries

Weight 1

Description Candidates should be able to determine the shared libraries that executable programs depend on and install them when necessary.

Key Knowledge Areas

- Identify shared libraries
- Identify the typical locations of system libraries
- Load shared libraries

The following is a partial list of the used files, terms and utilities:

- `ldd`
- `ldconfig`
- `/etc/ld.so.conf`
- `LD_LIBRARY_PATH`

102.4 Use Debian package management

Weight 3

Description Candidates should be able to perform package management using the Debian package tools.

Key Knowledge Areas

- Install, upgrade and uninstall Debian binary packages
- Find packages containing specific files or libraries which may or may not be installed
- Obtain package information like version, content, dependencies, package integrity and installation status (whether or not the package is installed)

The following is a partial list of the used files, terms and utilities:

- `/etc/apt/sources.list`
- `dpkg`
- `dpkg-reconfigure`
- `apt-get`
- `apt-cache`
- `aptitude`

102.5 Use RPM and YUM package management

Weight 3

Description Candidates should be able to perform package management using RPM and YUM tools.

Key Knowledge Areas

- Install, re-install, upgrade and remove packages using RPM and YUM
- Obtain information on RPM packages such as version, status, dependencies, integrity and signatures
- Determine what files a package provides, as well as find which package a specific file comes from

The following is a partial list of the used files, terms and utilities:

- `rpm`
- `rpm2cpio`
- `/etc/yum.conf`
- `/etc/yum.repos.d/`
- `yum`
- `yumdownloader`

103.3 Perform basic file management

Weight 4

Description Candidates should be able to use the basic Linux commands to manage files and directories.

Key Knowledge Areas

- Copy, move and remove files and directories individually
- Copy multiple files and directories recursively
- Remove files and directories recursively
- Use simple and advanced wildcard specifications in commands
- Using find to locate and act on files based on type, size, or time
- Usage of tar, cpio and dd

The following is a partial list of the used files, terms and utilities:

- `cp`
- `find`
- `mkdir`
- `mv`
- `ls`
- `rm`
- `rmdir`
- `touch`
- `tar`
- `cpio`
- `dd`
- `file`
- `gzip`
- `gunzip`
- `bzip2`
- `xz`
- file globbing

103.5 Create, monitor and kill processes

Weight 4

Description Candidates should be able to perform basic process management.

Key Knowledge Areas

- Run jobs in the foreground and background
- Signal a program to continue running after logout
- Monitor active processes

- Select and sort processes for display
- Send signals to processes

The following is a partial list of the used files, terms and utilities:

- `&`
- `bg`
- `fg`
- `jobs`
- `kill`
- `nohup`
- `ps`
- `top`
- `free`
- `uptime`
- `pgrep`
- `pkill`
- `killall`
- `screen`

103.6 Modify process execution priorities

Weight 2

Description Candidates should be able to manage process execution priorities.

Key Knowledge Areas

- Know the default priority of a job that is created
- Run a program with higher or lower priority than the default
- Change the priority of a running process

The following is a partial list of the used files, terms and utilities:

- `nice`
- `ps`
- `renice`
- `top`

104.1 Create partitions and filesystems

Weight 2

Description Candidates should be able to configure disk partitions and then create filesystems on media such as hard disks. This includes the handling of swap partitions.

Key Knowledge Areas

- Manage MBR partition tables
- Use various mkfs commands to create various filesystems such as:
  - ext2/ext3/ext4
  - XFS
  - VFAT
- Awareness of ReiserFS and Btrfs
- Basic knowledge of gdisk and parted with GPT

The following is a partial list of the used files, terms and utilities:

- `fdisk`
- `gdisk`
- `parted`
- `mkfs`
- `mkswap`

104.2 Maintain the integrity of filesystems

Weight 2

Description Candidates should be able to maintain a standard filesystem, as well as the extra data associated with a journaling filesystem.

Key Knowledge Areas

- Verify the integrity of filesystems
- Monitor free space and inodes
- Repair simple filesystem problems

The following is a partial list of the used files, terms and utilities:

- `du`
- `df`
- `fsck`
- `e2fsck`
- `mke2fs`
- `debugfs`
- `dumpe2fs`
- `tune2fs`
- XFS tools (such as `xfs_metadump` and `xfs_info`)

104.3 Control mounting and unmounting of filesystems

Weight 3

Description Candidates should be able to configure the mounting of a filesystem.

Key Knowledge Areas

- Manually mount and unmount filesystems
- Configure filesystem mounting on bootup
- Configure user mountable removable filesystems

The following is a partial list of the used files, terms and utilities:

- `/etc/fstab`
- `/media/`
- `mount`
- `umount`

104.4 Manage disk quotas

Weight 1

Description Candidates should be able to manage disk quotas for users.

Key Knowledge Areas:

- Set up a disk quota for a filesystem
- Edit, check and generate user quota reports

The following is a partial list of the used files, terms and utilities:

- `quota`
- `edquota`
- `repquota`
- `quotaon`

104.5 Manage file permissions and ownership

Weight 3

Description Candidates should be able to control file access through the proper use of permissions and ownerships.

Key Knowledge Areas

- Manage access permissions on regular and special files as well as directories
- Use access modes such as suid, sgid and the sticky bit to maintain security
- Know how to change the file creation mask
- Use the group field to grant file access to group members

The following is a partial list of the used files, terms and utilities:

- `chmod`
- `umask`
- `chown`
- `chgrp`

107.1 Manage user and group accounts and related system files

Weight 5

Description Candidates should be able to add, remove, suspend and change user accounts.

Key Knowledge Areas

- Add, modify and remove users and groups
- Manage user/group info in password/group databases
- Create and manage special purpose and limited accounts

The following is a partial list of the used files, terms and utilities:

- `/etc/passwd`
- `/etc/shadow`
- `/etc/group`
- `/etc/skel/`
- `chage`
- `getent`
- `groupadd`
- `groupdel`
- `groupmod`
- `passwd`
- `useradd`
- `userdel`
- `usermod`

110.1 Perform security administration tasks

Weight 3

Description Candidates should know how to review system configuration to ensure host security in accordance with local security policies.

Key Knowledge Areas

- Audit a system to find files with the suid/sgid bit set
- Set or change user passwords and password aging information
- Being able to use nmap and netstat to discover open ports on a system
- Set up limits on user logins, processes and memory usage

- Determine which users have logged in to the system or are currently logged in
- Basic sudo configuration and usage

The following is a partial list of the used files, terms and utilities:

- `find`
- `passwd`
- `fuser`
- `lsof`
- `nmap`
- `chage`
- `netstat`
- `sudo`
- `/etc/sudoers`
- `su`
- `usermod`
- `ulimit`
- `who`, `w`, `last`

110.2 Setup host security

Weight 3

Description Candidates should know how to set up a basic level of host security.

Key Knowledge Areas

- Awareness of shadow passwords and how they work
- Turn off network services not in use
- Understand the role of TCP wrappers

The following is a partial list of the used files, terms and utilities:

- `/etc/nologin`
- `/etc/passwd`
- `/etc/shadow`
- `/etc/xinetd.d/`
- `/etc/xinetd.conf`
- `/etc/inetd.d/`
- `/etc/inetd.conf`
- `/etc/inittab`
- `/etc/init.d/`
- `/etc/hosts.allow`
- `/etc/hosts.deny`

C

Command Index

This appendix summarises all commands explained in the manual and points to their documentation as well as the places in the text where the commands have been introduced.

Index

This index points to the most important key words in this document. Particularly important places for the individual key words are emphasised by **bold** type. Sorting takes place according to letters only; "~/.bashrc" is therefore placed under "B".