Linux 学习笔记

Linux Notes

theqiong.com

谨献给...。

λ

目　录

Part I

Introduction

第 1 章

UNIX

Unix (officially trademarked as UNIX) is a multitasking, multi-user computer operating system that exists in many variants. The original Unix was developed at AT&T's Bell Labs research center by Ken Thompson, Dennis Ritchie, and others.[1] From the power user's or programmer's perspective, Unix systems are characterized by a modular design that is sometimes called the "Unix philosophy", meaning the OS provides a set of simple tools that each perform a limited, well-defined function, with a unified filesystem as the main means of communication[1] and a shell scripting and command language to combine the tools to perform complex workflows.

The C programming language was designed by Dennis Ritchie as a systems programming language for Unix, allowing for portability beyond the initial PDP-11 development platform and the use of Unix on a plethora of computing platforms.

During the late 1970s and 1980s, Unix developed into a standard operating system for academia. AT&T tried to commercialize it by licensing the OS to third-party vendors, leading to a variety of both academic (e.g., BSD) and commercial variants of Unix (such as Xenix) and eventually to the "Unix wars" between groups of vendors. AT&T finally sold its rights in Unix to Novell in the early 1990s.

The Open Group, an industry standards consortium, now owns the UNIX trademark and allows its use for certified operating systems compliant with its standard, the Single UNIX Specification. Other operating systems that emulate Unix to some extent may be called Unix-like, although the Open Group disapproves of this term. The term Unix is also often used informally to denote any operating system that closely resembles the trademarked system. The most common version of Unix (bearing certification) is Apple's OS X, while Linux is the most popular non-certified workalike.

The Unix operating system was conceived and implemented in 1969 at AT&T's Bell Laboratories in the United States by Ken Thompson, Dennis Ritchie, Douglas McIlroy, and Joe Ossanna. It was first released in 1971, and initially, was written entirely in assembly language, a common practice at the time. Later, in a key pioneering approach in 1973, Unix was re-written in the programming language C by Dennis Ritchie (with exceptions to the kernel and I/O). The availability of an operating system written in a high-level language allowed easier portability to different computer platforms.

With AT&T being required to license the operating system's source code to anyone who asked (due to an earlier antitrust case forbidding them from entering the computer business), Unix grew quickly and became widely adopted by academic institutions and businesses. In 1984, AT&T divested itself of Bell Labs. Free of the legal obligation requiring free licensing, Bell Labs began selling Unix as a proprietary product.

Now, the Open Group holds the definition of what a UNIX system is and its associated trademark in trust for the industry.

In 1994 Novell (who had acquired the UNIX systems business of AT&T/USL) decided to get out of that business. Rather than sell the business as a single entity, Novell transferred the rights to the UNIX trademark and the specification (that subsequently became the Single UNIX Specification) to The Open Group (at the time X/Open Company). Subsequently, it sold the source code and the product implementation (UNIXWARE) to SCO. The Open Group also owns the trademark UNIXWARE.

Today, the definition of UNIX ® takes the form of the worldwide Single UNIX Specification integrating X/Open Company's XPG4, IEEE's POSIX Standards and ISO C. Through continual evolution, the Single UNIX Specification

is the defacto and dejure standard definition for the UNIX system application programming interfaces. As the owner of the UNIX trademark, The Open Group has separated the UNIX trademark from any actual code stream itself, thus allowing multiple implementations. Since the introduction of the Single UNIX Specification, there has been a single, open, consensus specification that defines the requirements for a conformant UNIX system.

There is also a mark, or brand, that is used to identify those products that have been certified as conforming to the Single UNIX Specification, initially UNIX 93, followed subsequently by UNIX 95, UNIX 98 and now UNIX 03.

The Open Group is committed to working with the community to further the development of standards conformant systems by evolving and maintaining the Single UNIX Specification and participation in other related standards efforts. Recent examples of this are making the standard freely available on the web, permitting reuse of the standard in open source documentation projects, providing test tools, developing the POSIX and LSB certification programs.

1.1 Overview

Originally, Unix was meant to be a programmer's workbench to be used for developing software to be run on multiple platforms[6] more than to be used to run application software. The system grew larger as the operating system started spreading in the academic circle, as users added their own tools to the system and shared them with colleagues.

Unix was designed to be portable, multi-tasking and multi-user in a time-sharing configuration. Unix systems are characterized by various concepts: the use of plain text for storing data; a hierarchical file system; treating devices and certain types of inter-process communication (IPC) as files; and the use of a large number of software tools, small programs that can be strung together through a command line interpreter using pipes, as opposed to using a single monolithic program that includes all of the same functionality. These concepts are collectively known as the "Unix philosophy." Brian Kernighan and Rob Pike summarize this in The Unix Programming Environment as "the idea that the power of a system comes more from the relationships among programs than from the programs themselves."

Unix operating systems are widely used in servers, workstations, and mobile devices.[9] The Unix environment and the client–server program model were essential elements in the development of the Internet and the reshaping of computing as centered in networks rather than in individual computers.

Both Unix and the C programming language were developed by AT&T and distributed to government and academic institutions, which led to both being ported to a wider variety of machine families than any other operating system. As a result, Unix became synonymous with open systems.[citation needed] Under Unix, the operating system consists of many utilities along with the master control program, the kernel. The kernel provides services to start and stop programs, handles the file system and other common "low level" tasks that most programs share, and schedules access to avoid conflicts when programs try to access the same resource or device simultaneously. To mediate such access, the kernel has special rights, reflected in the division between user-space and kernel-space.

The microkernel concept was introduced in an effort to reverse the trend towards larger kernels and return to a system in which most tasks were completed by smaller utilities. In an era when a standard computer consisted of a hard disk for storage and a data terminal for input and output (I/O), the Unix file model worked quite well, as most I/O was linear. However, modern systems include networking and other new devices. As graphical user interfaces developed, the file model proved inadequate to the task of handling asynchronous events such as those generated by a mouse. In the 1980s, non-blocking I/O and the set of inter-process communication mechanisms were augmented with Unix domain sockets, shared memory, message queues, and semaphores. Functions such as network protocols were moved out of the kernel.

概述

UNIX 操作系统(UNIX)是一种计算机操作系统,具有多任务、多用户的特征。UNIX 操作系统于 1969 年由美国 AT&T 公司的贝尔实验室实现。Unix 的前身为 Multics,贝尔实验室参与了这个操作系统的研发,但因为开发速度太慢,贝尔实验室决定放弃这个计划。贝尔实验室的工程师,汤普逊和里奇,在此时自行开发了 Unix。

Unix 因为其安全可靠,高效强大的特点在服务器领域得到了广泛的应用。直到 GNU/Linux 流行开始前,Unix 也是科学计算、大型机、超级计算机等所用操作系统的主流。现在其仍然被应用于一些对稳定性要求极高的数据中心之上。

UNIX 最早由肯·汤普逊(Ken Thompson),丹尼斯·里奇(Dennis Ritchie),道格拉斯·麦克罗伊(Douglas McIlroy),和 Joe Ossanna 于 1969 年在 AT&T 贝尔实验室开发,并于 1971 年首次发布,最初是完全用汇编语言编写,这是当时的一种普遍的做法。

后来,在 1973 年用一个重要的开拓性的方法,UNIX 被丹尼斯·里奇用编程语言 C(内核和 I/O 例外)重新编写。高级语言编写的操作系统具有的可用性,允许移植到不同的计算机平台更容易。

此后的 10 年,Unix 在学术机构和大型企业中得到了广泛的应用,当时的 UNIX 拥有者 AT&T 公司以低廉甚至免费的许可将 Unix 源码授权给学术机构做研究或教学之用,许多机构在此源码基础上加以扩充和改进,形成了所谓的“Unix 变种”。

这些“Unix 变种”反过来也促进了 Unix 的发展,其中最著名的变种之一是由加州大学柏克莱分校开发的柏克莱软件包 (BSD) 产品。BSD 使用主版本加次版本的方法标识,如 4.2、4.3BSD,在原始版本的基础上还有派生版本,这些版本通常有自己的名字,如 4.3BSD-Net/1,4.3BSD-Net/2 等。

Richard Stallman 创建了 GNU 项目,要创建一个能够自由发布的类 UNIX 系统。20 年来,这个项目不断发展壮大,包含了越来越多的内容。现在,GNU 项目开发的产品,比如 Emacs、GCC 等已经成为各种其他自由发布的类 UNIX 产品中的核心角色。

1990 年,Linus Torvalds 决定编写一个自己的 Minix 内核,初名为 Linus' Minix,意为 Linus 的 Minix 内核,后来改名为 Linux,此内核于 1991 年正式发布,并逐渐引起人们的注意。当 GNU 软件与 Linux 内核结合后,GNU 软件构成了这个 POSIX 兼容操作系统 GNU/Linux 的基础。今天 GNU/Linux 已经成为发展最为活跃的自由/开放源码的类 Unix 操作系统。

1994 年,BSD Unix 走上了复兴[1]的道路,其开发也走向了几个不同的方向,并最终导致了 FreeBSD、OpenBSD、NetBSD 和 DragonFlyBSD 的出现。

后来 AT&T 意识到了 Unix 的商业价值,不再将 Unix 源码授权给学术机构,并对之前的 Unix 及其变种声明了版权权利。BSD 在 Unix 的历史发展中具有相当大的影响力,被很多商业厂家采用,成为很多商用 Unix 的基础。其不断增大的影响力终于引起了 AT&T 的关注,于是开始了一场持久的版权官司,这场官司一直打到 AT&T 将自己的 Unix 系统实验室卖掉,新接手的 Novell 采取了一种比较开明的做法,允许柏克莱分校自由发布自己的 Unix 变种,但是前提是必须将来自于 AT&T 的代码完全删除,于是诞生了 4.4 BSD Lite 版,由于这个版本不存在法律问题,4.4 BSD Lite 成为了现代柏克莱软件包的基础版本。

尽管后来,非商业版的 Unix 系统又经过了很多演变,但其中有不少最终都是创建在 BSD 版本上(Linux、Minix 等系统除外)。所以从这个角度上,4.4 BSD 又是所有自由版本 Unix 的基础,它们和 System V 及 Linux 等共同构成 Unix 操作系统这片璀璨的星空。有很多大公司在取得了 Unix 的授权之后,开发了自己的 Unix 产品,比如国际商业机器股份有限公司的 AIX、惠普公司的 HP-UX、太阳微系统的 Solaris 和硅谷图形公司的 IRIX。

[1] Although not released until 1992 due to legal complications, development of 386BSD, from which NetBSD, OpenBSD and FreeBSD descended, predated that of Linux. Linus Torvalds has said that if 386BSD had been available at the time, he probably would not have created Linux.

386BSD 因为法律问题直到 1992 年还没有发布,NetBSD 和 FreeBSD 是 386BSD 的后裔,早于 Linux。林纳斯·托瓦兹曾说,当时如果有可用的 386BSD,他就可能不会编写 Linux。

现在，UNIX 不仅仅是一个操作系统，更是一种生活方式。经过几十年的发展，UNIX 在技术上日臻成熟的过程中，她独特的设计哲学和美学也深深地吸引了一大批技术人员，他们在维护、开发、使用 UNIX 的同时，UNIX 也影响了他们的思考方式和看待世界的角度。

UNIX 重要的设计原则包括：

- 简洁至上（KISS 原则）
- 提供机制而非策略

从 1980 年代开始，POSIX——一个开放的操作系统标准就在制定中，IEEE 制定的 POSIX 标准（ISO/IEC 9945）现在是 UNIX 系统的基础部分。

目前 UNIX 的商标权由国际开放标准组织（The Open Group，缩写为 TOG）所拥有，只有符合单一 UNIX 规范的 UNIX 系统才能使用 UNIX 这个名称，否则只能称为类 UNIX（UNIX-like）。

国际开放标准组织（又译为国际标准化组织）是以制定电脑架构的共通标准为目的而成立的国际性非营利组织，在英国登记注册。在 1996 年，由 X/Open 与开源软件基金会（Open Software Foundation）合组而成，拥有 UNIX 的商标权，它制定并且发布了单一 UNIX 规范（Single UNIX Specification）。

这里，单一 UNIX 规范（Single UNIX Specification，缩写为 SUS）是一套 UNIX 系统的统一规格书。它扩充了 POSIX 标准，定义了标准 UNIX 操作系统。最初由 IEEE 与 The Open Group 所提出，目前由 Austin Group 负责维持。

- 1980 年代

  在 1980 年代中，开始有人提出计划，想要统一不同 Unix 操作系统的接口。
- 1988 年：POSIX

  1988 年，这些标准被汇整为 IEEE 1003（ISO/IEC 9945），也就是 POSIX。
- 1990 年代：Spec 1170

  共通应用程式接口规格（Common API Specification），又称为 Spec 1170。
- 1997 年：单一 UNIX 规范第二版

  单一 UNIX 规范第二版（Single UNIX Specification version 2）。
- 2001 年：POSIX:2001，单一 UNIX 规范第三版
- 2004 年：POSIX:2004
- 2008 年：POSIX:2008

通过 UNIX 认证的操作系统包括：AIX、HP/UX、OS X、Reliant UNIX、SCO、Solaris、Tru64 UNIX、z/OS 等。

1.2 Histroy

The history of Unix dates back to the mid-1960s when the Massachusetts Institute of Technology, AT&T Bell Labs, and General Electric were developing an experimental time sharing operating system called Multics for the GE-645 mainframe. Multics introduced many innovations, but had many problems.

Bell Labs, frustrated by the size and complexity of Multics but not the aims, slowly pulled out of the project. Their last researchers to leave Multics, Ken Thompson, Dennis Ritchie, M. D. McIlroy, and J. F. Ossanna, decided to redo the work on a much smaller scale.

In 1979, Dennis Ritchie described their vision for Unix:

```
What we wanted to preserve was not just a good environment in which to do programmir
```

While Ken Thompson still had access to the Multics environment, he wrote simulations for the new file and paging system on it. He also programmed a game called Space Travel, but the game needed a more efficient and less expensive machine to run on, and eventually he found a little-used PDP-7 at Bell Labs.[3] On this PDP-7, in 1969, a team of Bell Labs researchers led by Thompson and Ritchie, including Rudd Canaday, developed a hierarchical file system, the concepts of computer processes and device files, a command-line interpreter, and some small utility programs.

The first production instance of Unix was installed in early 1972 at New York Telephone Co. Systems Development Center, under the direction of Dan Gielan. In 1972, Unix was rewritten in the C programming language.[12] The migration from assembly to the higher-level language C, resulted in much more portable software, requiring only a relatively small amount of machine-dependent code to be replaced when porting Unix to other computing platforms. Bell Labs produced several versions of Unix that are collectively referred to as Research Unix.

During the late 1970s and early 1980s, the influence of Unix in academic circles led to large-scale adoption of Unix (BSD and System V) by commercial startups, some of the most notable of which are Sequent, HP-UX, Solaris, AIX, and Xenix. In the late 1980s, System V Release 4 (SVR4) was developed by AT&T Unix System Laboratories and Sun Microsystems. SVR4 was subsequently adopted by many commercial Unix vendors.

In the 1990s, Unix-like systems grew in popularity as Linux and BSD distributions were developed through collaboration by a worldwide network of programmers. Later, Apple also released Darwin, which became the core of the OS X operating system.

1.2.1 1970s

In 1970, Peter Neumann coined the project name UNICS (UNiplexed Information and Computing Service) as a pun on Multics (Multiplexed Information and Computer Services): the new operating system was an emasculated Multics.

Up to that point, there had been no financial support from Bell Labs. When the Computer Science Research Group wanted to use Unix on a machine much larger than the PDP-7, Thompson and Ritchie managed to trade the promise of adding text processing capabilities to Unix, for a PDP-11/20 machine. This led to some financial support from Bell. For the first time in 1970, the Unix operating system was officially named and ran on the PDP-11/20. A text formatting program called roff and a text editor were added. All three were written in PDP-11/20 assembly language. Bell Labs used this initial text processing system, consisting of Unix, roff, and the editor, for text processing of patent applications. Roff soon evolved into troff, the first electronic publishing program with full typesetting capability. The UNIX Programmer's Manual was published on 3 November 1971.

The first commercial instance of Unix worldwide was installed in early 1972 at New York Telephone Co. Systems

Development Center, under the direction of Dan Gielan. An Operational Support System was developed entirely in assembly language by Neil Groundwater that survived nearly 7 years, without change.

In 1972, Unix was rewritten in the C programming language, contrary to the general notion at the time "that something as complex as an operating system, which must deal with time-critical events, has to be written exclusively in assembly language". The migration from assembly to the higher-level language C, resulted in much more portable software, requiring only a relatively small amount of machine-dependent code to be replaced when porting Unix to other computing platforms.

Under a 1956 consent decree in settlement of an antitrust case, AT&T (the parent organization of Bell Labs) had been forbidden from entering the computer business[citation needed]. Unix could not, therefore, be turned into a product. Indeed, under the terms of the decree, Bell Labs was required to license its non-telephone technology to anyone who asked. Ken Thompson quietly began answering requests by shipping out tapes and disks, each accompanied by —according to legend —a note signed, "Love, Ken".

AT&T made Unix available to universities and commercial firms, as well as the United States government, under licenses. The licenses included all source code including the machine-dependent parts of the kernel, which were written in PDP-11 assembly language. Copies of the annotated Unix kernel sources circulated widely in the late 1970s in the form of a much-copied book by John Lions of the University of New South Wales, the Lions' Commentary on UNIX 6th Edition, with Source Code, which led to considerable use of Unix as an educational example.

Versions of the Unix system were determined by editions of its user manuals. For example, "Fifth Edition UNIX" and "UNIX Version 5" have both been used to designate the same version. Development expanded, with Versions 4, 5, and 6 being released by 1975. These versions added the concept of pipes, which led to the development of a more modular code base, and quicker development cycles. Version 5, and especially Version 6, led to a plethora of different Unix versions both inside and outside Bell Labs, including PWB/UNIX and the first commercial Unix, IS/1. As more of Unix was rewritten in C, portability also increased. A group at the University of Wollongong ported Unix to the Interdata 7/32. Bell Labs developed several ports for research purposes and internal use at AT&T. Target machines included an Intel 8086-based computer (with custom-built MMU) and the UNIVAC 1100.

In May 1975, ARPA documented the benefits of the Unix time-sharing system which "presents several interesting capabilities" as an ARPA network mini-host in RFC 681.

At the time Unix required a license from Bell Laboratories that at $20,000(US) was very expensive for non-university users, while an educational license cost just $150. It was noted that Bell was "open to suggestions" for an ARPANET-wide license.

Specific features found beneficial were:

- Local processing facilities.
- Compilers.
- Editor.
- Document preparation system.
- Efficient file system and access control.
- Mountable and de-mountable volumes.
- Unified treatment of peripherals as special files.
- The network control program (NCP) was integrated within the Unix file system.
- Network connections treated as special files which can be accessed through standard Unix I/O calls.
- The system closes all files on program exit.
- "desirable to minimize the amount of code added to the basic Unix kernel".

In 1978, UNIX/32V was released for DEC's then new VAX system. By this time, over 600 machines were running Unix in some form. Version 7 Unix, the last version of Research Unix to be released widely, was released

in 1979. In Version 7, the number of system calls was only around 50, although later Unix and Unix-like systems would add many more later:

```
Version 7 of the Research UNIX System provided about 50 system calls, 4.4BSD provide
```

Research Unix versions 8, 9 and 10 were developed through the 1980s but were only released to a few universities, though they did generate papers describing the new work. This research led to the development of Plan 9 from Bell Labs, a new portable distributed system.

1.2.2 1980s

For internal use, Bell had developed multiple versions of Unix, such as CB UNIX (with improved support for databases) and PWB/UNIX, the ”Programmer's Workbench”, aimed at large groups of programmers. It advertised the latter version, as well as 32V and V7, stating that ”more than 800 systems are already in use outside the Bell System” in 1980,[9] and ”more than 2000” the following year.[10] AT&T licensed UNIX System III, based largely on Version 7, for commercial use, the first version launching in 1982. This also included support for the VAX. AT&T continued to issue licenses for older Unix versions. To end the confusion between all its differing internal versions, AT&T combined them into UNIX System V Release 1. This introduced a few features such as the vi editor and curses from the Berkeley Software Distribution of Unix developed at the University of California, Berkeley. This also included support for the Western Electric 3B series of machines. AT&T provided support for System III and System V through the Unix Support Group (USG), and these systems were sometimes referred to as USG Unix.

In 1983, the U.S. Department of Justice settled its second antitrust case against AT&T and broke up the Bell System. This relieved AT&T of the 1956 consent decree that had prevented them from turning Unix into a product. AT&T promptly rushed to commercialize Unix System V, a move that nearly killed Unix.[6] The GNU Project was founded in the same year by Richard Stallman.

Since the newer commercial UNIX licensing terms were not as favorable for academic use as the older versions of Unix, the Berkeley researchers continued to develop BSD Unix as an alternative to UNIX System III and V. Many contributions to Unix first appeared in BSD releases, notably the C shell with job control (modelled on ITS). Perhaps the most important aspect of the BSD development effort was the addition of TCP/IP network code to the mainstream Unix kernel. The BSD effort produced several significant releases that contained network code: 4.1cBSD, 4.2BSD, 4.3BSD, 4.3BSD-Tahoe (”Tahoe” being the nickname of the Computer Consoles Inc. Power 6/32 architecture that was the first non-DEC release of the BSD kernel), Net/1, 4.3BSD-Reno (to match the ”Tahoe” naming, and that the release was something of a gamble), Net/2, 4.4BSD, and 4.4BSD-lite. The network code found in these releases is the ancestor of much TCP/IP network code in use today, including code that was later released in AT&T System V UNIX and early versions of Microsoft Windows. The accompanying Berkeley sockets API is a de facto standard for networking APIs and has been copied on many platforms.

Other companies began to offer commercial versions of the UNIX System for their own mini-computers and workstations. Many of these new Unix flavors were developed from the System V base under a license from AT&T; others were based on BSD. One of the leading developers of BSD, Bill Joy, went on to co-found Sun Microsystems in 1982 and created SunOS for their workstation computers. In 1980, Microsoft announced its first Unix for 16-bit microcomputers called Xenix, which the Santa Cruz Operation (SCO) ported to the Intel 8086 processor in 1983, and eventually branched Xenix into SCO UNIX in 1989.

During this period (before PC compatible computers with MS-DOS became dominant), industry observers expected that UNIX, with its portability and rich capabilities, was likely to become the industry standard operating system for microcomputers.[11] In 1984, several companies established the X/Open consortium with the goal of creating an open system specification based on UNIX. Despite early progress, the standardization effort collapsed

into the "Unix wars", with various companies forming rival standardization groups. The most successful Unix-related standard turned out to be the IEEE's POSIX specification, designed as a compromise API readily implemented on both BSD and System V platforms, published in 1988 and soon mandated by the United States government for many of its own systems.

AT&T added various features into UNIX System V, such as file locking, system administration, STREAMS, new forms of IPC, the Remote File System and TLI. AT&T cooperated with Sun Microsystems and between 1987 and 1989, merged features from Xenix, BSD, SunOS, and System V into System V Release 4 (SVR4), independently of X/Open. This new release consolidated all the previous features into one package, and heralded the end of competing versions. It also increased licensing fees.

During this time a number of vendors including Digital Equipment, Sun, Addamax and others began building trusted versions of UNIX for high security applications, mostly designed for military and law enforcement applications.

1.2.3 1990s

In 1990, the Open Software Foundation released OSF/1, their standard Unix implementation, based on Mach and BSD. The Foundation was started in 1988 and was funded by several Unix-related companies that wished to counteract the collaboration of AT&T and Sun on SVR4. Subsequently, AT&T and another group of licensees formed the group UNIX International in order to counteract OSF. This escalation of conflict between competing vendors again gave rise to the phrase Unix wars.

In 1991, a group of BSD developers (Donn Seeley, Mike Karels, Bill Jolitz, and Trent Hein) left the University of California to found Berkeley Software Design, Inc (BSDI). BSDI produced a fully functional commercial version of BSD Unix for the inexpensive and ubiquitous Intel platform, which started a wave of interest in the use of inexpensive hardware for production computing. Shortly after it was founded, Bill Jolitz left BSDI to pursue distribution of 386BSD, the free software ancestor of FreeBSD, OpenBSD, and NetBSD.

In 1991, Linus Torvalds began work on Linux, a Unix-Like system which he based on MINIX that initially ran on IBM PC compatible computers.

By 1993, most commercial vendors had changed their variants of Unix to be based on System V with many BSD features added. The creation of the Common Open Software Environment (COSE) initiative that year, by the major players in Unix, marked the end of the most notorious phase of the Unix wars, and was followed by the merger of UI and OSF in 1994. The new combined entity retained the OSF name and stopped work on OSF/1. By that time the only vendor using it was Digital Equipment Corporation, which continued its own development, rebranding their product Digital UNIX in early 1995.

Shortly after UNIX System V Release 4 was produced, AT&T sold all its rights to UNIX to Novell. Dennis Ritchie likened this sale to the Biblical story of Esau selling his birthright for the mess of pottage.[12] Novell developed its own version, UnixWare, merging its NetWare with UNIX System V Release 4. Novell tried to use this as a marketing tool against Windows NT, but their core markets suffered considerably.

In 1993, Novell decided to transfer the UNIX trademark and certification rights to the X/Open Consortium.[13] In 1996, X/Open merged with OSF, creating the Open Group. Various standards by the Open Group now define what is and what is not a UNIX operating system, notably the post-1998 Single UNIX Specification.

In 1995, the business of administering and supporting the existing UNIX licenses, plus rights to further develop the System V code base, were sold by Novell to the Santa Cruz Operation.[14] Whether Novell also sold the copyrights is currently the subject of litigation (see below).

In 1997, Apple Computer sought a new foundation for its Macintosh operating system and chose NEXTSTEP, an operating system developed by NeXT. The core operating system, which was based on BSD and the Mach kernel, was renamed Darwin after Apple acquired it. The deployment of Darwin in Mac OS X makes it, according to a

statement made by an Apple employee at a USENIX conference, the most widely used Unix-based system in the desktop computer market.

In 1998, a confidential memo at Microsoft stated, ”Linux is on track to eventually own the x86 UNIX market,” and further predicted, ”I believe that Linux – moreso than NT – will be the biggest threat to SCO in the near future.”

1.2.4 2000s

In 2000, SCO sold its entire UNIX business and assets to Caldera Systems, which later changed its name to The SCO Group.

The bursting of the dot-com bubble (2001–2003) led to significant consolidation of versions of Unix. Of the many commercial variants of Unix that were born in the 1980s, only Solaris, HP-UX, and AIX were still doing relatively well in the market, though SGI's IRIX persisted for quite some time. Of these, Solaris had the largest market share in 2005.

In 2003, the SCO Group started legal action against various users and vendors of Linux. SCO had alleged that Linux contained copyrighted Unix code now owned by the SCO Group. Other allegations included trade-secret violations by IBM, or contract violations by former Santa Cruz customers who had since converted to Linux. However, Novell disputed the SCO Group's claim to hold copyright on the UNIX source base. According to Novell, SCO (and hence the SCO Group) are effectively franchise operators for Novell, which also retained the core copyrights, veto rights over future licensing activities of SCO, and 95% of the licensing revenue. The SCO Group disagreed with this, and the dispute resulted in the SCO v. Novell lawsuit. On 10 August 2007, a major portion of the case was decided in Novell's favor (that Novell had the copyright to UNIX, and that the SCO Group had improperly kept money that was due to Novell). The court also ruled that ”SCO is obligated to recognize Novell's waiver of SCO's claims against IBM and Sequent”. After the ruling, Novell announced they have no interest in suing people over Unix and stated, ”We don't believe there is Unix in Linux”. SCO successfully got the 10th Circuit Court of Appeals to partially overturn this decision on 24 August 2009 which sent the lawsuit back to the courts for a jury trial.

On 30 March 2010, following a jury trial, Novell, and not The SCO Group, was ”unanimously” to be the owner of the UNIX and UnixWare copyrights. The SCO Group, through bankruptcy trustee Edward Cahn, decided to continue the lawsuit against IBM for causing a decline in SCO revenues.

In 2005, Sun Microsystems released the bulk of its Solaris system code (based on UNIX System V Release 4) into an open source project called OpenSolaris. New Sun OS technologies, notably the ZFS file system, were first released as open source code via the OpenSolaris project. Soon afterwards, OpenSolaris spawned several non-Sun distributions. In 2010, after Oracle acquired Sun, OpenSolaris was officially discontinued, but the development of derivatives continued.

历史

早期的计算机都是用于军事或者是高科技用途以及学术研究。非但如此，早期的计算机还很难使用，除了命令周期并不快之外操作接口也不友好。在那个时候的输入设备只有卡片阅读机，输出设备只有打印机，用户也无法与操作系统互动(这时期的操作系统称为多道批处理操作系统)。

程序设计者在写程序时，必须要将程序相关的信息在读卡纸上打孔，然后再将读卡纸插入读卡机来将信息输入主机中运算。如果程序有某个地方写错，加上主机少，用户众多，光是等待，就耗去很多的时间了。

后来可以使用键盘来进行信息的输入/输出，不过毕竟主机数量太少，只能是大家轮流等待使用。在 1960 年代初期 MIT 开发了所谓的“兼容分时系统”(Compatible Time-Sharing System, CTSS)，它可以让大型主机通过提供数个终端(terminal)以连线进入主机来使用主机的资源。

CTSS 可以说是近代操作系统的鼻祖，它可以让多个用户在某一段时间内分别使用 CPU 的资源，感觉上用户会觉得大家是同时使用该主机的资源。但是事实上是 CPU 在每个用户的工作之间进行切换，但在当时 CTSS 是划时代的技术。

这样一来，无论主机在哪里，只要在终端进行操作就可利用主机提供的功能了。不过需要注意的是，此时终端只具有输入/输出的功能，本身完全不具备任何运算或者软件安装的能力，而且比较先进的主机大概也只能提供不到 30 个终端机而已。

为了更加强化大型主机的系统，让主机的资源可以提供更多用户来使用，所以在 1965 年前后由贝尔实验室(Bell Lab.)、MIT 及 GE 共同发起了 Multics[2]的计划，Multics 目的是想要让大型主机可以提供 300 个以上的终端连线使用。不过到了 1969 年前后，计划进度落后，资金也短缺，所以该计划就宣告失败[3]。

在认为 Multics 计划不可能成功之后，贝尔研究室就退出了该计划，不过原本参与 Multics 计划的人员中已经从该计划当中获得一些点子，Ken Thompson 就是其中一位。

Thompson 因为自己的需要，希望开发一个小的操作系统以满足自己的需求。当时有一台 DEC (Digital Equipment Corporation) 的 PDP-7 没人使用，于是他就准备针对这台主机进行操作系统核心程序的编写。本来 Thompson 是没时间的，有趣的是在 1969 年八月份左右，刚好 Thompson 的妻儿探亲去了，于是他有了额外的一个月的时间好好的待在家将一些构想实现出来。

经过四个星期的奋斗，Thompson 终于以汇编语言写出了一个核心程序，同时包括一些核心工具程序以及一个小小的文件系统，那个系统就是 UNIX 的原型。

当时 Thompson 将 Multics 庞大的复杂系统简化了不少，于是同实验室的朋友都戏称这个系统为：Unics。

Thompson 的这个文件系统有两个重要的概念，分别是：

1. 所有的程序或系统设备都是文件；
2. 不管构建编辑器还是附属文件，所写的程序只有一个目的，且要有效的完成目标。

这些概念在后来对于 Linux 的开发有相当重要的影响。

Thompson 写的那个操作系统在贝尔实验室内部广为流传并且经过多次改版。但是，比较重要的改版则发生在 1973 年。

UNIX 本来是以汇编语言编写的，后来因为系统移植与效率的需求，该系统又被用 B 语言改写。不过效率依旧不是很好。后来 Dennis Ritchie 将 B 语言重新改写成 C 语言，C 语言可以在不同的机器上面运行，而 Ritchie 等人再以 C 语言重新改写与编译 Unics，最后开发出 UNIX 的正式版本。

在这个时候需要特别注意的是，贝尔实验室是隶属于 AT&T 的，只是 AT&T 当时忙于其他商业活动，此外 UNIX 在这个时期的开发者都是贝尔实验室的工程师，这些工程师对于程序当然相当有研究，

[2]注：Multics 有复杂、多数的意思

[3]最终 Multics 还是成功地开发出了他们的系统，参考http://www.multicians.org/

所以 UNIX 在此时是不容易被一般人所接受的。此外也需要特别强调，由于 UNIX 是以较高阶的 C 语言写的，相对于汇编语言需要与硬件有密切的配合，C 语言与硬件平台的相关性就没有那么大了，所以这个改变也使得 UNIX 很容易被移植到不同的机器上。

由于 UNIX 的高度可移植性与效率，加上当时并没有版权的纠纷，所以让很多商业公司开始了 UNIX 操作系统的开发，例如 AT&T 自家的 System V、IBM 的 AIX 以及 HP 与 DEC 等公司，并且他们都推出了自家的主机搭配自己的 UNIX 操作系统。

操作系统的核心（kernel）必须要跟硬件配合以提供及控制硬件的资源进行良好的工作，而在早期每一家生产计算机硬件的公司还没有所谓的“标准协议”的概念，所以每一个计算机公司生产的硬件自然就不相同，因此他们必须要为自己的计算机硬件开发合适的 UNIX 系统，例如 Sun、Cray 与 HP 就是这一种情况。

他们开发出来的 UNIX 操作系统以及内含的相关软件并没有办法在其他的硬件架构下工作的，而且由于 UNIX 强调的是多用户、多任务的环境，但早期的 286 个人计算机架构下的 CPU 是没有能力处理多任务的作业的，因此没有厂商针对个人计算机设计 UNIX 系统，于是在早期并没有出现支持个人计算机的 UNIX 操作系统。

每一家公司的 UNIX 虽然在架构上面大同小异，但是却真的仅能支持自身的硬件，所以早先的 UNIX 只能与服务器（Server）或者是大型工作站（Workstation）划上等号。

但是这个高度开放的 UNIX 系统在 1979 年有了重大的转折，因为 AT&T 由于商业上和在当时现实环境下的考虑想将 UNIX 的版权收回去，因此在 AT&T 在 1979 年发行的第七版 UNIX 中，特别提到了“不可对学生提供源代码”的严格限制，同时也造成 UNIX 业界之间的紧张气氛并且也引发了很多的商业纠纷。

另外，到了 1979 年时 AT&T 推出 System V 第七版 UNIX 后，这一版最重要的特色是可以支持 x86 架构的个人计算机系统，也就是说 System V 可以在个人计算机上安装和运行了。

1.3 Standards

Beginning in the late 1980s, an open operating system standardization effort now known as POSIX provided a common baseline for all operating systems; IEEE based POSIX around the common structure of the major competing variants of the Unix system, publishing the first POSIX standard in 1988. In the early 1990s, a separate but very similar effort was started by an industry consortium, the Common Open Software Environment (COSE) initiative, which eventually became the Single UNIX Specification administered by The Open Group. Starting in 1998, the Open Group and IEEE started the Austin Group, to provide a common definition of POSIX and the Single UNIX Specification.

In 1999, in an effort towards compatibility, several Unix system vendors agreed on SVR4's Executable and Linkable Format (ELF) as the standard for binary and object code files. The common format allows substantial binary compatibility among Unix systems operating on the same CPU architecture.

The Filesystem Hierarchy Standard was created to provide a reference directory layout for Unix-like operating systems, and has mainly been used in Linux.

1.4 Components

The Unix system is composed of several components[?] that are normally packaged together. By including – in addition to the kernel of an operating system – the development environment, libraries, documents, and the portable, modifiable source-code for all of these components, Unix was a self-contained software system. This was one of the key reasons it emerged as an important teaching and learning tool and has had such a broad influence.

The inclusion of these components did not make the system large – the original V7 UNIX distribution, consisting of copies of all of the compiled binaries plus all of the source code and documentation occupied less than 10MB, and arrived on a single 9-track magnetic tape. The printed documentation, typeset from the on-line sources, was contained in two volumes.

The names and filesystem locations of the Unix components have changed substantially across the history of the system. Nonetheless, the V7 implementation is considered by many to have the canonical early structure:

- Kernel – source code in /usr/sys, composed of several sub-components:
  - conf – configuration and machine-dependent parts, including boot code
  - dev – device drivers for control of hardware (and some pseudo-hardware)
  - sys – operating system "kernel", handling memory management, process scheduling, system calls, etc.
  - h – header files, defining key structures within the system and important system-specific invariables
- Development Environment – Early versions of Unix contained a development environment sufficient to recreate the entire system from source code:
  - cc – C language compiler (first appeared in V3 Unix)
  - as – machine-language assembler for the machine
  - ld – linker, for combining object files
  - lib – object-code libraries (installed in /lib or /usr/lib). libc, the system library with C run-time support, was the primary library, but there have always been additional libraries for such things as mathematical functions (libm) or database access. V7 Unix introduced the first version of the modern "Standard I/O" library stdio as part of the system library. Later implementations increased the number of libraries significantly.
  - make – build manager (introduced in PWB/UNIX), for effectively automating the build process
  - include – header files for software development, defining standard interfaces and system invariants
  - Other languages – V7 Unix contained a Fortran-77 compiler, a programmable arbitrary-precision calculator (bc, dc), and the awk scripting language, and later versions and implementations contain many other

language compilers and toolsets. Early BSD releases included Pascal tools, and many modern Unix systems also include the GNU Compiler Collection as well as or instead of a proprietary compiler system.
    - Other tools – including an object-code archive manager (ar), symbol-table lister (nm), compiler-development tools (e.g. lex & yacc), and debugging tools.
- Commands – Unix makes little distinction between commands (user-level programs) for system operation and maintenance (e.g. cron), commands of general utility (e.g. grep), and more general-purpose applications such as the text formatting and typesetting package. Nonetheless, some major categories are:
    - sh – The "shell" programmable command line interpreter, the primary user interface on Unix before window systems appeared, and even afterward (within a "command window").
    - Utilities – the core tool kit of the Unix command set, including cp, ls, grep, find and many others. Subcategories include:
        * System utilities – administrative tools such as mkfs, fsck, and many others.
        * User utilities – environment management tools such as passwd, kill, and others.
    - Document formatting – Unix systems were used from the outset for document preparation and typesetting systems, and included many related programs such as nroff, troff, tbl, eqn, refer, and pic. Some modern Unix systems also include packages such as TeX and Ghostscript.
    - Graphics – The plot subsystem provided facilities for producing simple vector plots in a device-independent format, with device-specific interpreters to display such files. Modern Unix systems also generally include X11 as a standard windowing system and GUI, and many support OpenGL.
    - Communications – Early Unix systems contained no inter-system communication, but did include the inter-user communication programs mail and write. V7 introduced the early inter-system communication system UUCP, and systems beginning with BSD release 4.1c included TCP/IP utilities.
- Documentation – Unix was the first operating system to include all of its documentation online in machine-readable form. The documentation included:
    - man – manual pages for each command, library component, system call, header file, etc.
    - doc – longer documents detailing major subsystems, such as the C language and troff

This is a list of Unix utilities as specified by IEEE Std 1003.1-2008, which is part of the Single UNIX Specification (SUS). These utilities can be found on Unix operating systems and most Unix-like operating systems.

Table 1.1: IEEE Std 1003.1-2008 utilities

admin	SCCS	Create and administer SCCS files
alias	Misc	Define or display aliases
ar	Misc	Create and maintain library archives
asa	Text processing	Interpret carriage-control characters
at	Process management	Execute commands at a later time
awk	Text processing	Pattern scanning and processing language
basename	Filesystem	Return non-directory portion of a pathname; see also dirname
batch	Process management	Schedule commands to be executed in a batch queue
bc	Misc	Arbitrary-precision arithmetic language
bg	Process management	Run jobs in the background
c99	C programming	Compile standard C programs
cal	Misc	Print a calendar
cat	Filesystem	Concatenate and print files
cd	Filesystem	Change the working directory

Name	Category	Description
cflow	C programming	Generate a C-language flowgraph
chgrp	Filesystem	Change the file group ownership
chmod	Filesystem	Change the file modes/attributes/permissions
chown	Filesystem	Change the file ownership
cksum	Filesystem	Write file checksums and sizes
cmp	Filesystem	Compare two files; see also diff
comm	Text processing	Select or reject lines common to two files
command	Shell programming	Execute a simple command
compress	Filesystem	Compress data
cp	Filesystem	Copy files
crontab	Misc	Schedule periodic background work
csplit	Text processing	Split files based on context
ctags	C programming	Create a tags file
cut	Text processing	Cut out selected fields of each line of a file
cxref	C programming	Generate a C-language program cross-reference table
date	Misc	Display the date and time
dd	Filesystem	Convert and copy a file
delta	SCCS	Make a delta (change) to an SCCS file
df	Filesystem	Report free disk space
diff	Text processing	Compare two files; see also cmp
dirname	Filesystem	Return the directory portion of a pathname; see also basename
du	Filesystem	Estimate file space usage
echo	Shell programming	Write arguments to standard output
ed	Text processing	The standard text editor
env	Misc	Set the environment for command invocation
ex	Text processing	Text editor
expand	Text processing	Convert tabs to spaces
expr	Shell programming	Evaluate arguments as an expression
false	Shell programming	Return false value
fc	Misc	Process the command history list
fg	Process management	Run jobs in the foreground
file	Filesystem	Determine file type
find	Filesystem	Find files
fold	Text processing	Filter for folding lines
fort77	FORTRAN77 programming	FORTRAN compiler
fuser	Process management	List process IDs of all processes that have one or more files open
gencat	Misc	Generate a formatted message catalog
get	SCCS	Get a version of an SCCS file
getconf	Misc	Get configuration values
getopts	Shell programming	Parse utility options
grep	Misc	Search text for a pattern

Name	Category	Description
hash	Misc	hash database access method
head	Text processing	Copy the first part of files
iconv	Text processing	Codeset conversion
id	Misc	Return user identity
ipcrm	Misc	Remove a message queue, semaphore set, or shared memory segment identifier
ipcs	Misc	Report interprocess communication facilities status
jobs	Process management	Display status of jobs in the current session
join	Text processing	Merges two sorted text files based on the presence of a common field
kill	Process management	Terminate or signal processes
lex	C programming	Generate programs for lexical tasks
link	Filesystem	Create a hard link to a file
ln	Filesystem	Link files
locale	Misc	Get locale-specific information
localedef	Misc	Define locale environment
logger	Shell programming	Log messages
logname	Misc	Return the user's login name
lp	Text processing	Send files to a printer
ls	Filesystem	List directory contents
m4	Misc	Macro processor
mailx	Misc	Process messages
make	Programming	Maintain, update, and regenerate groups of programs
man	Misc	Display system documentation
mesg	Misc	Permit or deny messages
mkdir	Filesystem	Make directories
mkfifo	Filesystem	Make FIFO special files
more	Text processing	Display files on a page-by-page basis
mv	Filesystem	Move files
newgrp	Misc	Change to a new group (functionaliy similar to sg)
nice	Process management	Invoke a utility with an altered nice value
nl	Text processing	Line numbering filter
nm	C programming	Write the name list of an object file
nohup	Process management	Invoke a utility immune to hangups
od	Misc	Dump files in various formats
paste	Text processing	Merge corresponding or subsequent lines of files
patch	Text processing	Apply changes to files
pathchk	Filesystem	Check pathnames
pax	Misc	Portable archive interchange
pr	Text processing	Print files
printf	Shell programming	Write formatted output
prs	SCCS	Print an SCCS file

Name	Category	Description
ps	Process management	Report process status
pwd	Filesystem	Print working directory - Return working directory name
qalter	Batch utilities	Alter batch job
qdel	Batch utilities	Delete batch jobs
qhold	Batch utilities	Hold batch jobs
qmove	Batch utilities	Move batch jobs
qmsg	Batch utilities	Send message to batch jobs
qrerun	Batch utilities	Rerun batch jobs
qrls	Batch utilities	Release batch jobs
qselect	Batch utilities	Select batch jobs
qsig	Batch utilities	Signal batch jobs
qstat	Batch utilities	Show status of batch jobs
qsub	Batch utilities	Submit a script
read	Shell programming	Read a line from standard input
renice	Process management	Set nice values of running processes
rm	Filesystem	Remove directory entries
rmdel	SCCS	Remove a delta from an SCCS file
rmdir	Filesystem	Remove directories
sact	SCCS	Print current SCCS file-editing activity
sccs	SCCS	Front end for the SCCS subsystem
sed	Text processing	Stream editor
sh[4]	Shell programming	Shell, the standard command language interpreter
sleep	Shell programming	Suspend execution for an interval
sort	Text processing	Sort, merge, or sequence check text files
split	Misc	Split files into pieces
strings	C programming	Find printable strings in files
strip	C programming	Remove unnecessary information from executable files
stty	Misc	Set the options for a terminal
tabs	Misc	Set terminal tabs
tail	Text processing	Copy the last part of a file
talk	Misc	Talk to another user
tee	Shell programming	Duplicate the standard output
test	Shell programming	Evaluate expression
time	Process management	Time a simple command
touch	Filesystem	Change file access and modification times
tput	Misc	Change terminal characteristics
tr	Text processing	Translate characters
true	Shell programming	Return true value
tsort	Text processing	Topological sort
tty	Misc	Return user's terminal name
type	Misc	Displays how a name would be interpreted if used as a command

[4] In earlier versions, sh was either the Thompson shell or the PWB shell.

Name	Category	Description
ulimit	Misc	Set or report file size limit
umask	Misc	Get or set the file mode creation mask
unalias	Misc	Remove alias definitions
uname	Misc	Return system name
uncompress	Misc	Expand compressed data
unexpand	Text processing	Convert spaces to tabs
unget	SCCS	Undo a previous get of an SCCS file
uniq	Text processing	Report or filter out repeated lines in a file
unlink	Filesystem	Call the unlink function
uucp	Network	System-to-system copy
uudecode	Network	Decode a binary file
uuencode	Network	Encode a binary file
uustat	Network	uucp status inquiry and job control
uux	Process management	Remote command execution
val	SCCS	Validate SCCS files
vi	Text processing	Screen-oriented (visual) display editor
wait	Process management	Await process completion
wc	Text processing	Line, word and byte or character count
what	SCCS	Identify SCCS files
who	System administration	Display who is on the system
write	Misc	Write to another user's terminal
xargs	Shell programming	Construct argument lists and invoke utility
yacc	C programming	Yet another compiler compiler
zcat	Text processing	Expand and concatenate data

Unix stores system time values as the number of seconds from midnight 1 January 1970 (the "Unix Epoch") in variables of type time_t, historically defined as "signed long". On 19 January 2038 on 32 bit Unix systems, the current time will roll over from a zero followed by 31 ones (0x7FFFFFFF) to a one followed by 31 zeros (0x80000000), which will reset time to the year 1901 or 1970, depending on implementation, because that toggles the sign bit.

Since times before 1970 are rarely represented in Unix time, one possible solution that is compatible with existing binary formats would be to redefine time_t as "unsigned 32-bit integer". However, such a kludge merely postpones the problem to 7 February 2106, and could introduce bugs in software that computes time differences.

Some Unix versions have already addressed this. For example, in Solaris and Linux in 64-bit mode, time_t is 64 bits long, meaning that the OS itself and 64-bit applications will correctly handle dates for some 292 billion years. Existing 32-bit applications using a 32-bit time_t continue to work on 64-bit Solaris systems but are still prone to the 2038 problem. Some vendors have introduced an alternative 64-bit type and corresponding API, without addressing uses of the standard time_t. The NetBSD Project decided to instead bump time_t to 64-bit in its 6th major release for both 32-bit and 64-bit architectures, supporting 32-bit time_t in applications compiled for a former NetBSD release via its binary compatibility layer.

UNIX 组成

本列表中的 UNIX 实用程序由 IEEE Std 1003.1-2008 定义，是单一 UNIX 规范（SUS）的一部分，这些实用程序可以在 UNIX 操作系统和绝大多数类 UNIX 操作系统中找到。

Table 1.2: IEEE 标准 1003.1-2008 实用程序

名称	分类	描述
admin	源代码控制系统	创建和管理源代码控制系统文件
alias	其他	定义或者显示别名
ar	其他	生成并维护函数库
asa	文字处理	Interpret carriage-control characters
at	进程管理	在设定时间执行命令
awk	文字处理	模式扫描和处理语言
basename	文件系统	输入文件完整路径，只返回其文件名
batch	进程管理	按队列执行 at 命令
bc	其他	计算器编程语言
bg	进程管理	后台运行作业
c99	C 语言编程	标准 C 语言编译器
cal	其他	输出日历
cat	文件系统	连接和输出文件
cd	文件系统	改变工作目录
cflow	C 语言编程	生成 C 语言流程图
chgrp	文件系统	改变文件组拥有者
chmod	文件系统	改变文件权限
chown	文件系统	改变文件所有者
cksum	文件系统	计算文件校验和和大小
clear	文件系统	清除屏幕
cmp	文件系统	比较 2 个文件
comm	文字处理	按行比较两个已排序文件
command	Shell 编程	执行简单命令
compress	文件系统	压缩数据
cp	文件系统	复制文件
crontab	其他	设制定期运行的后台程序
csplit	文字处理	基于内容分割文件
ctags	C 语言编程	创建 C 语言的标记（tag）文件
cut	Shell 编程	选择文本中每行的特定区域
cxref	C 语言编程	生成 C 语言程序交叉引用表
date	其他	输出日期和时间
dd	文件系统	转换或复制文件
delta	源代码控制系统	为源代码控制系统生成差异文件
df	文件系统	报告磁盘剩余空间
diff	文字处理	比较 2 个文件
dirname	文件系统	返回路径的目录
du	文件系统	计算磁盘占用空间

名称	分类	描述
echo	Shell 编程	输出命令参数到标准输出
ed	文字处理	标准文本编辑器
env	其他	为命令设置环境变量
ex	文字处理	文字编辑器
expand	文字处理	转换跳格为空格
expr	Shell 编程	计算表达式的值
false	Shell 编程	返回假值
fc	其他	处理命令行历史
fg	进程管理	在前台运行命令
file	文件系统	判断文件类型
find	文件系统	查找文件
fold	文字处理	回折每行文本到特定宽度
fort77	FORTRAN77 编程	FORTRAN 编译器
fuser	进程管理	列出所有打开文件的进程的进程号
gencat	其他	生成一个格式化的消息目录
get	源代码控制系统	取得源代码控制系统文件某个版本
getconf	其他	查询系统配置变量
getopts	Shell 编程	解析命令行选项参数
grep	其他	根据模式搜索文字
hash	其他	提示或者报告程序位置
head	文字处理	显示文件开头几行
iconv	文字处理	转换字符集
id	其他	返回用户标示符
ipcrm	其他	删除消息队列,信号集或者共享内存段标识
ipcs	其他	显示进程间通信的状态
jobs	进程管理	显示当前会话中任务状态
join	文字处理	关系型数据库操作
kill	进程管理	结束进程或向进程发信号
lex	C 语言编程	为词法分析程器审查功能程序
link	文件系统	创建文件硬链接
ln	文件系统	创建文件链接
locale	其他	获得本地信息
localedef	其他	定义本地环境变量
logger	Shell 编程	记录消息日志
logname	其他	返回当前登陆用户名
lp	文字处理	发送文件到打印机
ls	文件系统	列出目录内容
m4	其他	宏处理器
mailx	其他	发送电子邮件
make	编程	维护一整套代码库,组织编译
man	其他	显示系统文档
mesg	其他	允许或者拒绝消息

名称	分类	描述
mkdir	文件系统	创建目录
mkfifo	文件系统	生成 FIFO 类型文件
more	文字处理	逐页显示文件
mv	文件系统	移动文件
newgrp	其他	登陆到其他用户组
nice	进程管理	用新的 nice 值运行程序
nl	文字处理	加行号显示文本
nm	C 语言编程	显示目标文件的符号表
nohup	进程管理	运行一个忽略 SIGHUP 信号的程序
od	其他	将文件以八进制或其他进制输出
paste	文字处理	合并文件
patch	文字处理	将改变写入文件
pathchk	文件系统	检验路径名
pax	其他	Portable archive interchange
pr	文字处理	打印文件
printf	Shell 编程	格式化输出
prs	源代码控制系统	打印源代码控制系统文件
ps	进程管理	报告进程状态
pwd	文件系统	输出当前目录
qalter	批处理实用程序	Alter 批处理任务
qdel	批处理实用程序	删除批处理任务
qhold	批处理实用程序	暂停批处理任务
qmove	批处理实用程序	移动批处理任务
qmsg	批处理实用程序	向批处理任务发送消息
qrerun	批处理实用程序	返回批处理任务
qrls	批处理实用程序	释放批处理任务
qselect	批处理实用程序	选择批处理任务
qsig	批处理实用程序	发信号给批处理任务
qstat	批处理实用程序	显示批处理任务状态
qsub	批处理实用程序	提交脚本
read	Shell 编程	从标准输入读取一行
renice	进程管理	设置进程的 nice 值
rm	文件系统	删除整个目录
rmdel	源代码控制系统	从 SCCS 文件中删除差异
rmdir	文件系统	删除空目录
sact	源代码控制系统	显示 SCCS 文件正在进行的编辑
sccs	源代码控制系统	源代码控制系统前端
sed	文字处理	流编辑器
sh[5]	Shell 编程	Shell, 标准命令语言解析器
sleep	Shell 编程	延时
sort	文字处理	文本排序

[5]早期版本 sh 可能是 Thompson shell 或者 PWB shell。

名称	分类	描述
split	其他	分割文件
strings	C 语言编程	查找文件中可打印字符串
strip	C 语言编程	从可执行文件中移除无用信息
stty	其他	设置终端选项
tabs	其他	定义终端跳格
tail	文字处理	显示文件结尾
talk	其他	与另外用户对话
tee	Shell 编程	从标准输入读入，写到标准输出
test	Shell 编程	计算表达式
time	进程管理	计算一个命令的执行时间
touch	文件系统	改变文件访问和修改时间
tput	其他	改变终端字符
tr	文字处理	翻译字符
true	Shell 编程	返回真值
tsort	文字处理	拓扑排序
tty	其他	返回用户终端名
type	其他	显示命令类型
ulimit	其他	设置或显示文件限制
umask	其他	设置或显示文件生成掩码
unalias	其他	移除别名定义
uname	其他	返回系统名
uncompress	其他	解压缩数据
unexpand	文字处理	转换空格为制表符
unget	源代码控制系统	回退之前从源代码控制系统获得的文件
uniq	文字处理	报告或者删除文件中重复行
unlink	文件系统	调用未链接函数
uucp	网络	系统间拷贝
uudecode	网络	解码二进制文件
uuencode	网络	编码二进制文件
uustat	网络	uucp 状态查询和作业控制
uux	进程管理	远程命令调用
val	源代码控制系统	验证 SCCS 文件
vi	文字处理	面向屏幕的可视化编辑器
wait	进程管理	等待进程结束
wc	文字处理	字、行字节或者字符计数
what	源代码控制系统	鉴别源代码控制系统文件
who	系统管理	显示登录用户
write	其他	输出到另一个用户终端
xargs	Shell 编程	从输入列表中执行命令
yacc	C 语言编程	用来生成编译器的编译器
zcat	文字处理	显示或连接 zip 压缩的文件

1.5 Daemons

This is a list of Unix daemons that are found on various Unix-like operating systems. Unix daemons typically have a name ending with a d.

Table 1.3: IEEE Std 1003.1-2008 utilities

init	The Unix program which spawns all other processes.
biod	Works in cooperation with the remote nfsd to handle client NFS requests.
crond	Time-based job scheduler, runs jobs in the background.
dhcpd	Dynamically configure TCP/IP information for clients.
fingerd	Provides a network interface for the finger protocol, as used by the finger command.
ftpd	Services FTP requests from a remote system.
httpd	Web server daemon.
inetd	Listens for network connection requests. If a request is accepted, it can launch a background daemon to handle the request, was known as the super server for this reason. Some systems use the replacement command xinetd.
lpd	The line printer daemon that manages printer spooling.
nfsd	Processes NFS operation requests from client systems. Historically each nfsd daemon handled one request at a time, so it was normal to start multiple copies.
ntpd	Network Time Protocol daemon that manages clock synchronization across the network. xntpd implements the version 3 standard of NTP.
portmap/rpcbind	Provides information to allow ONC RPC clients to contact ONC RPC servers
sshd	Listens for secure shell requests from clients.
sendmail	SMTP daemon.
swapper	Copies process regions to swap space in order to reclaim physical pages of memory for the kernel. Also called sched.
syslogd	System logger process that collects various system messages.
syncd	Periodically keeps the file systems synchronized with system memory.
xfsd	Serve X11 fonts to remote clients.
vhand	Releases pages of memory for use by other processes. Also known as the “page stealing daemon”
ypbind	Find the server for an NIS domain and store the information in a file.

第 2 章

GNU

The GNU Project, started in 1983 by Richard Stallman, had the goal of creating a “complete UNIX-compatible software system” composed entirely of free software. “free” in that everyone who received a copy would be free to use, study, modify, and redistribute it.

Its work began in 1984. Later, in 1985, Stallman started the Free Software Foundation and wrote the GNU General Public License (GNU GPL) in 1989. By the early 1990s, many of the programs required in an operating system (such as libraries, compilers, text editors, a UNIX shell, and a windowing system) were completed, although low-level elements such as device drivers, daemons, and the kernel were stalled and incomplete.

Now, the system's basic components include the GNU Compiler Collection (GCC), the GNU C library (glibc), and GNU Core Utilities (coreutils), but also the GNU Debugger (GDB), GNU Binary Utilities (binutils), the bash shell and the GNOME desktop environment. GNU developers have contributed Linux ports of GNU applications and utilities, which are now also widely used on other operating systems such as BSD variants, Solaris and Mac OS X.

Many GNU programs have been ported to other operating systems, including proprietary platforms such as Microsoft Windows and Mac OS X. Compared to their proprietary Unix counterparts, GNU programs have also been shown to be more reliable

The GNU project's own kernel development project, GNU Hurd, had not produced a working kernel, but in 1991 Linus Torvalds released the Linux kernel as free software under the GNU General Public License. In addition to their use in the Linux operating system, many GNU packages – such as the GNU Compiler Collection (and the rest of the GNU toolchain), the GNU C library and the GNU core utilities – have gone on to play central roles in other free Unix systems as well.

GNU is a Unix-like computer operating system developed by the GNU Project. It is composed wholly of free software. It is based on the GNU Hurd kernel and is intended to be a ”complete Unix-compatible software system”[1] [4][5] GNU is a recursive acronym for ”GNU's Not Unix!”, chosen because GNU's design is Unix-like, but differs from Unix by being free software and containing no Unix code.

Development of GNU was initiated by Richard Stallman in 1983 and was the original focus of the Free Software Foundation (FSF), but no stable release of GNU yet exists as of January 2014. However, non-GNU kernels, most famously the Linux kernel, can also be used with GNU software. Stallman views GNU as a ”technical means to a social end.”

Linux distributions, consisting of the Linux kernel and large collections of compatible software have become popular both with individual users and in business. Popular distributions include Red Hat Enterprise Linux, Fedora, SUSE Linux Enterprise, openSUSE, Debian GNU/Linux, Ubuntu, Linux Mint, Mandriva Linux, Slackware Linux, MEPIS, and Gentoo.

A free derivative of BSD Unix, 386BSD, was released in 1992 and led to the NetBSD and FreeBSD projects. With the 1994 settlement of a lawsuit brought against the University of California and Berkeley Software Design Inc. (USL v. BSDi) by UNIX Systems Laboratories, it was clarified that Berkeley had the right to distribute BSD Unix for free, if it so desired. Since then, BSD Unix has been developed in several different product branches, including OpenBSD and DragonFly BSD.

Linux and BSD are increasingly filling the market needs traditionally served by proprietary Unix operating

systems, as well as expanding into new markets such as the consumer desktop and mobile and embedded devices. Because of the modular design of the Unix model, sharing components is relatively common; consequently, most or all Unix and Unix-like systems include at least some BSD code, and some systems also include GNU utilities in their distributions.

In a 1999 interview, Dennis Ritchie clarified his opinion, that Linux and BSD operating systems are a continuation of the basis of the Unix design, and as derivatives of Unix:

> I think the Linux phenomenon is quite delightful, because it draws so strongly on the basis that Unix provided. Linux seems to be the among the healthiest of the direct Unix derivatives, though there are also the various BSD systems as well as the more official offerings from the workstation and mainframe manufacturers.

In the same interview, he states that he views both Unix and Linux as "the continuation of ideas that were started by Ken and me and many others, many years ago."

OpenSolaris is a relatively recent addition to the list of operating systems based on free software licenses marked as such by FSF and OSI. It includes a number of derivatives that combines CDDL-licensed kernel and system tools and also GNU userland and is currently the only open-source System V derivative available.

Linus Torvalds has said that if the GNU kernel had been available at the time (1991), he would not have decided to write his own.

2.1 FSF

Richard Mathew Stallman(史托曼)在 1984 年发起的 GNU 计划对于自由软件风潮具有不可磨灭的作用。目前很多自由软件几乎均直接或间接受益于 GNU 计划。

GNU("Gnu's Not Unix") 是一个由 GNU 计划推动的类 UNIX 的操作系统,目标是创建一个完全兼容于 UNIX 的自由软件环境。1983 年,理查德·马修·斯托曼创立了 GNU 计划,目标是为了开发一个完全自由的类 UNIX 操作系统。

在 1985 年,理查德·马修·斯托曼发起自由软件基金会(FSF)并且在 1989 年撰写了 GPL 协议。1990 年代早期,GNU 开始大量的产生或收集各种系统所必备的组件,包括库、编译器、调试工具、文本编辑器、网页服务器以及一个 UNIX 的用户界面(UNIX shell)——但是像一些底层环境,如硬件驱动、守护进程运行内核(kernel)仍然不完整和陷于停顿。

GNU 计划中是在马赫微核 (Mach microkernel) 的架构之上开发系统内核,也就是所谓的 GNU Hurd,但是这个基于 Mach 的设计异常复杂,发展进度则相对缓慢。最近一个 GNU 系统版本,是于 2011 年 4 月 1 日发布的 GNU 0.401,采用 GNU Hurd 作为操作系统内核。但直到 2013 年为止,都还没有稳定版本发布。

其他的内核,最著名的是 Linux kernel[1],也被应用在 GNU 系统中。

Stallman(生于 1953 年,在网络上自称的 ID 为 RMS)在 1971 年的时候进入骇客(hacker)圈中相当出名的人工智慧实验室(AI Lab.),这个时候的骇客专指计算机功力很强的人而非破坏计算机的怪客(cracker)。

当时的骇客圈对于软件的着眼点几乎都是在"分享",所以并没有专利方面的困扰,这个特色对于史托曼的影响很大。不过后来由于管理层的问题导致实验室的优秀骇客离开该实验室,并且进入其他商业公司继续开发优秀的软件。但史托曼并不服输,仍然继续在原来的实验室开发新的程序与软件,后来他发现自己一个人无法完成所有的工作,于是想成立一个开放的团体来共同努力。

1983 年以后,因为实验室硬件的更换使得史托曼无法继续以原有的硬件与操作系统继续自由程序的编写,而且他进一步发现过去他所使用的 LISP 操作系统是 MIT 的专利软件,是无法共享的,这对

[1]林纳斯·托瓦兹曾说过如果 GNU 内核在 1991 年时可以用,他不会自己去写一个。

于想要成立一个开放团体的史托曼是个阻碍,于是他便放弃了 LISP 这个系统。后来他接触到 UNIX 这个系统并且发现 UNIX 在理论与实际上都可以在不同的机器间进行移植,于是他开始转而使用 UNIX 系统。

因为 LISP 与 UNIX 是不同的系统,所以他原本已经编写完的软件是无法在 UNIX 上面运行的,为此他就开始将软件移植到 UNIX 上,并且为了让软件可以在不同的平台上运行,史托曼将其开发的软件均编写成可以移植的。

1984 年史托曼开始了 GNU 计划,这个计划的目的是想要建立一个自由的开放的 UNIX 操作系统(Free UNIX),但是在当时的 GNU 是仅有自己一个人单打独斗的史托曼,但又不能不做这个计划,于是史托曼反其道而行——"既然操作系统太复杂,我就先写可以在 UNIX 上面运行的程序",于是史托曼便开始了程序的编写。在写作期间为了不让自己惹上官司,他绝对不看专利软件的源代码。为了这个计划他开始使用原本 UNIX 上面跑的软件,并自行编写功能与 UNIX 原有专利软件相仿的软件。

但不论是什么软件,都得要编译成二进制文件(binary file)后才能够执行,因此他便开始编写 C 语言编译器,那就是后来的 GNU C(gcc)。这一点相当的重要,因为 C 语言编译器版本众多但都是专利软件,如果他开发的 C 编译器足够好,效率足够高,那么将会尽早的让 GNU 计划出现在大众眼前。

> 计算机仅认识 0/1 的信息,但是人类只能识别纯文本的信息(就是 ASCII 文件格式),但是计算机又不认识 ASCII 格式的文本

但开始时并不顺利,为此他先转而将 Emacs 编辑器写成可以在 UNIX 上运行的软件并公开源代码,因为 Emacs 太优秀了因此很多人便直接向他购买。此时 Internet 尚未流行,所以史托曼便通过将 Emacs 以磁带(tape)出售赚了一点钱,进而开始全力编写其他软件并且成立了自由软件基金会(FSF, Free Software Foundation),并请更多工程师与志愿者编写软件才终于完成了 GCC,这比 Emacs 还更有帮助。

此外,他还编写了更多可以被调用的 C 函数库(GNU C library)以及可以被用来操作操作系统的基本界面 BASH Shell,这些都在 1990 年左右完成。

> 如果纯粹使用文本编辑器来编辑程序的话,那么程序语法写错时,只能利用编译时产生的错误信息来修订了,这样实在很没有效率。Em

到了 1985 年,为了避免 GNU 所开发的自由软件被其他人所利用而成为专利软件,所以他与律师草拟了有名的通用公共许可证(General Public License, GPL),并且称为 copyleft(相对于专利软件的 copyright)。在这里,必须要说明的是,由于有 GNU 所开发的几个重要软件,如:

1. Emacs
2. GNU C(GCC)
3. GNU C Library(glibc)
4. Bash Shell

这样后来很多的软件开发者可以通过这些基础的工具来进行程序开发,进一步壮大了自由软件团体,这是很重要的。不过对于 GNU 的最初构想"建立一个自由的 UNIX 操作系统"来说,有这些优秀的程序是仍无法满足,因为当下并没有"自由的 UNIX 核心"存在,所以这些软件仍只能在那些有专利的 UNIX 平台上工作,一直到 Linux 的出现。

2.2 GPL

1984 年创立 GNU 计划与 FSF 基金会的 Stallman 先生认为,写程序最大的快乐就是让大家使用自己开发的良好的软件,虽然程序是想要分享给大家使用的,不过每个人所使用的计算机软硬件并不相同,这样该程序的源代码(Source Code)就应该要同时发布,这样才能方便大家修改从而适用于每个人的计算机。这个将源代码发布的举动,就称为 Open Source。

此外史托曼同时认为,如果您将您程序的 Source Code 享出来时,若该程序是很优秀的,那么将会有很多人使用,而每个人对于该程序都可以查看 Source Code,无形之中就会有一帮人帮您除错,这个程序将会越来越壮大,越来越优秀。

而为了避免自己的开发出来的 Open Source 的自由软件被拿去做成专利软件, Stallman 同时将 GNU 与 FSF 开发出来的软件都挂上 GPL 的版权协议。

Linux 操作系统是基于 GPL 许可证授权下的,该许可证可防止开源软件被转换为封闭源代码软件及确保源代码的可用性,GPL 许可证的目的就是防止二进制包成为唯一的软件发行源[?]。GPL 的核心观念是“版权制度是促进社会进步的手段,版权本身不是自然权力。”

> 其实 GNU 是 GNU's Not UNIX 的缩写,意思是说,GNU 并不是 UNIX,那么 GNU 又是什么呢?就是 GNU's Not UNIX。如
>
> 另外,所谓的 source 是程序开发者写出的原始程序代码,Open Source 就是软件在发布时同时将作者的源代码一起公布。

GPL(GNU General Public License)这个版权宣告对于作者有何好处?

首先,Stallman 对 GPL 一直是强调 Free 的,这个 Free 的意思是这样的:

“Free software” is a matter of liberty, not price. To understand the concept, you should think of “free speech”, not “free beer”. “Free software” refers to the users freedom to run, copy, distribute, study, change, and improve the software.

大意是说,Free Software(自由软件)是一种自由的权力,并非是“价格”。举例来说,你可以拥有自由呼吸的权力、你拥有自由发表言论的权力,但是这并不代表您可以到处喝“免费的啤酒(free beer)”,也就是说,自由软件的重点并不是指“免费”的,而是指具有“自由度,freedom”的软件。

史托曼进一步说明了自由度的意义是:用户可以自由的执行、复制、再发行、学习、修改与强化自由软件。如此一来,你所拿到的软件可能原先只能在 UNIX 上面运行,但是经过源代码的修改之后,你将可以让它在 Linux 或者是 Windows 上面来运行。总之,一个软件挂上了 GPL 版权宣告之后,它自然就成了自由软件。

自由软件具有下面的特色:

1. 取得软件与源代码:您可以根据自己的需求来执行这个自由软件;
2. 复制:您可以自由的复制该软件;
3. 修改:您可以将取得的源代码进进程序修改工作,使之适合您的工作;
4. 再发行:您可以将您修改过的程序,再度的自由发行,而不会与原先的编写者冲突;
5. 回馈:您应该将您修改过的程序代码回馈于社群。

但请特别留意,您所修改的任何一个自由软件都不应该也不能这样:

1. 修改授权:您不能将一个 GPL 授权的自由软件,在您修改后而将它取消 GPL 授权;
2. 单纯销售:您不能单纯的销售自由软件。

也就是说,既然 GPL 是站在互助互利的角度上去开发的,您自然不应该将大家的成果占为己有而取消 GPL 授权,因此当然不可以将一个 GPL 软件的授权取消,即使您已经对该软件进行大幅度的修改,那么自由软件也不能销售吗?

当然不是!自由软件是可以销售的,不过不可以仅销售该软件,应同时搭配售后服务与相关手册。

很多人还是有疑问,目前不是有很多 Linux 开发商吗?为何他们可以销售 Linux 这个 GPL 授权的软件?原因很简单,因为他们大多都是销售“售后服务”,所以他们所使用的自由软件,都可以在他们的网站上面下载(当然,每个厂商他们自己开发的工具软件就不是 GPL 的授权软件了)。但是,您可以购买他们的 Linux 光盘,如果您购买了光盘,他们会提供相关的手册说明文件,同时也会提供您数年不等的谘询、售后服务、软件升级与其他协助工作等等的附加价值。

所以说,目前自由软件工作者所赖以维生的,几乎都是在“服务”这个领域。毕竟自由软件并不是每个人都会编写,有人有需要您的自由软件时,他就会请求您的协助,此时,您就可以通过服务来收费,这样来说,自由软件确实还是具有商业空间的。

很多人对于 GPL 授权一直很疑惑,对于 GPL 的商业行为更是无法接受。关于这一点,这里还是要再次的申明,GPL 是可以从事商

上面提到的大多是与用户有关的项目,那么 GPL 对于自由软件的作者有何好处呢?

大致的优点有这些:

1. 软件安全性较佳;
2. 软件执行效率较佳;
3. 软件除错时间较短;
4. 贡献的源代码远永都存在。

这是因为既然是 Open Source 的自由软件,那么程序代码将会有很多人帮您查看,如此一来,程序的漏洞发现与程序的优化将会很快,所以在安全性与效率上面,自由软件一点都不输给商业软件。此外因为 GPL 授权中,修改者并不能修改授权,因此您如果曾经贡献过程序代码,您的贡献将会一直流传下去。

不过,GPL 对于程序开发者的优点是相当多的,不过对于不熟悉程序的一般人来说,GPL 的优点其实不太容易看出来。

首先,虽然它是随手可得的自由软件,不过您也必须要会使用基本的编译器。这对于一般人来说并不容易,对于不懂程序的人来说,商业公司是一个很快速的解决之道,而对于我们广大的读者群来说,认识了/学习了 Linux 与自由软件的相关技巧后,对于未来确实是有很不错的帮助。

第 3 章

BSD

3.1 Overview

Berkeley Software Distribution (BSD, sometimes called Berkeley Unix) is a Unix operating system derivative developed and distributed by the Computer Systems Research Group (CSRG) of the University of California, Berkeley, from 1977 to 1995. Today the term "BSD" is often used non-specifically to refer to any of the BSD descendants which together form a branch of the family of Unix-like operating systems. Operating systems derived from the original BSD code remain actively developed and widely used.

Historically, BSD has been considered a branch of UNIX—"BSD UNIX", because it shared the initial codebase and design with the original AT&T UNIX operating system. In the 1980s, BSD was widely adopted by vendors of workstation-class systems in the form of proprietary UNIX variants such as DEC ULTRIX and Sun Microsystems SunOS. This can be attributed to the ease with which it could be licensed, and the familiarity the founders of many technology companies of the time had with it.

Although these proprietary BSD derivatives were largely superseded by the UNIX System V Release 4 and OSF/1 systems in the 1990s (both of which incorporated BSD code and are the basis of other modern Unix systems), later BSD releases provided a basis for several open source development projects, e.g. FreeBSD, NetBSD, OpenBSD or DragonFly BSD, that are ongoing. These, in turn, have been incorporated in whole or in part in modern proprietary operating systems, e.g. the TCP/IP (IPv4 only) networking code in Microsoft Windows and a part of the foundation of Apple's OS X.

Berkeley's Unix was the first Unix to include libraries supporting the Internet Protocol stacks: Berkeley sockets. (A Unix implementation of IP's predecessor, the ARPAnet's NCP, with FTP and Telnet clients, had been produced at U. Illinois in 1975, and was available at Berkeley. However, the memory scarcity on the PDP-11 forced a complicated design and performance problems.)

By integrating sockets with the Unix operating system's file descriptors, it became almost as easy to read and write data across a network as it was to access a disk. The AT&T laboratory eventually released their own STREAMS library, which incorporated much of the same functionality in a software stack with a different architecture, but the wide distribution of the existing sockets library reduced the impact of the new API. Early versions of BSD were used to form Sun Microsystems' SunOS, founding the first wave of popular Unix workstations.

Today, BSD continues to serve as a technological testbed for academic organizations. It can be found in use in numerous commercial and free products, and, increasingly, in embedded devices. The general quality of its source code, as well as its documentation (especially reference manual pages, commonly referred to as man pages), make it well-suited for many purposes.

The permissive nature of the BSD license allows companies to distribute derived products as proprietary software without exposing source code and sometimes intellectual property to competitors. Searching for strings containing "University of California, Berkeley" in the documentation of products, in the static data sections of binaries and ROMs, or as part of other information about a software program, will often show BSD code has been used. This permissiveness also makes BSD code suitable for use in open source products, and the license is compatible with many other open source licenses. The permissive nature of the BSD license also allows derivative works of code released originally under the BSD license to become less permissive with time.

BSD operating systems can run much native software of several other operating systems on the same architecture, using a binary compatibility layer. Much simpler and faster than emulation, this allows, for instance, applications intended for Linux to be run at effectively full speed. This makes BSDs not only suitable for server environments, but also for workstation ones, given the increasing availability of commercial or closed-source software for Linux only. This also allows administrators to migrate legacy commercial applications, which may have only supported commercial Unix variants, to a more modern operating system, retaining the functionality of such applications until they can be replaced by a better alternative.

Current BSD operating system variants support many of the common IEEE, ANSI, ISO, and POSIX standards, while retaining most of the traditional BSD behavior. Like AT&T Unix, the BSD kernel is monolithic, meaning that device drivers in the kernel run in privileged mode, as part of the core of the operating system.

3.2 Histroy

The earliest distributions of Unix from Bell Labs in the 1970s included the source code to the operating system, allowing researchers at universities to modify and extend Unix. The first Unix system at Berkeley was a PDP-11 installed in 1974, and the computer science department used it for extensive research thereafter. Other universities became interested in the software at Berkeley, and so in 1977 Bill Joy, then a graduate student at Berkeley, started compiling the first Berkeley Software Distribution (1BSD), which was released on March 9, 1978.[1] 1BSD was an add-on to Sixth Edition Unix rather than a complete operating system in its own right; its main components were a Pascal compiler and Joy's ex line editor.

The Second Berkeley Software Distribution (2BSD), released in May, 1979,[2] included updated versions of the 1BSD software as well as two new programs by Joy that persist on Unix systems to this day: the vi text editor (a visual version of ex) and the C shell.

Later releases of 2BSD contained ports of changes to the VAX-based releases of BSD back to the PDP-11 architecture. 2.9BSD from 1983 included code from 4.1cBSD, and was the first release that was a full OS (a modified Version 7 Unix) rather than a set of applications and patches. The most recent release, 2.11BSD, was first released in 1992. As of 2008, maintenance updates from volunteers are still continuing, with patch 447 being released on December 31, 2008.

A VAX computer was installed at Berkeley in 1978, but the port of Unix to the VAX architecture, UNIX/32V, did not take advantage of the VAX's virtual memory capabilities. The kernel of 32V was largely rewritten by Berkeley students to include a virtual memory implementation, and a complete operating system including the new kernel, ports of the 2BSD utilities to the VAX, and the utilities from 32V was released as 3BSD at the end of 1979. 3BSD was also alternatively called Virtual VAX/UNIX or VMUNIX (for Virtual Memory Unix), and BSD kernel images were normally called /vmunix until 4.4BSD. The success of 3BSD was a major factor in the Defense Advanced Research Projects Agency's (DARPA) decision to fund Berkeley's Computer Systems Research Group (CSRG), which would develop a standard Unix platform for future DARPA research in the VLSI Project. CSRG released 4BSD, containing numerous improvements to the 3BSD system, in October 1980. According to Quarterman et al.,

> 4BSD was the operating system of choice for VAXs from the beginning until the release of

4BSD (November 1980) offered a number of enhancements over 3BSD, notably job control in the previously released csh, delivermail (the antecedent of sendmail), "reliable" signals, and the Curses programming library.

4.1BSD (June 1981) was a response to criticisms of BSD's performance relative to the dominant VAX operating system, VMS. The 4.1BSD kernel was systematically tuned up by Bill Joy until it could perform as well as VMS on several benchmarks. (The release would have been called 5BSD, but after objections from AT&T the name was

changed; AT&T feared confusion with AT&T's UNIX System V.[5] One early, never-released test version was in fact called 4.5BSD.)

4.2BSD would take over two years to implement and contained several major overhauls. Before its official release came three intermediate versions: 4.1a incorporated a modified version of BBN's preliminary TCP/IP implementation; 4.1b included the new Berkeley Fast File System, implemented by Marshall Kirk McKusick; and 4.1c was an interim release during the last few months of 4.2BSD's development. Back at Bell Labs, 4.1cBSD became the basis of the Eight Edition of Research Unix, and a commercially supported version was available from mtXinu.

To guide the design of 4.2BSD, Duane Adams of DARPA formed a "steering committee" consisting of Bob Fabry, Bill Joy and Sam Leffler from UCB, Alan Nemeth and Rob Gurwitz from BBN, Dennis Ritchie from Bell Labs, Keith Lantz from Stanford, Rick Rashid from Carnegie-Mellon, Bert Halstead from MIT, Dan Lynch from ISI, and Gerald J. Popek of UCLA. The committee met from April 1981 to June 1983.

Apart from the Fast File System, several features from outside contributors were accepted, including disk quotas and job control. Sun Microsystems provided testing on its Motorola 68000 machines prior to release, ensuring portability of the system.

The official 4.2BSD release came in August 1983. It was notable as the first version released after the 1982 departure of Bill Joy to co-found Sun Microsystems; Mike Karels and Marshall Kirk McKusick took on leadership roles within the project from that point forward. On a lighter note, it also marked the debut of BSD's daemon mascot in a drawing by John Lasseter that appeared on the cover of the printed manuals distributed by USENIX.

4.3BSD was released in June 1986. Its main changes were to improve the performance of many of the new contributions of 4.2BSD that had not been as heavily tuned as the 4.1BSD code. Prior to the release, BSD's implementation of TCP/IP had diverged considerably from BBN's official implementation. After several months of testing, DARPA determined that the 4.2BSD version was superior and would remain in 4.3BSD.

After 4.3BSD, it was determined that BSD would move away from the aging VAX platform. The Power 6/32 platform (codenamed "Tahoe") developed by Computer Consoles Inc. seemed promising at the time, but was abandoned by its developers shortly thereafter. Nonetheless, the 4.3BSD-Tahoe port (June 1988) proved valuable, as it led to a separation of machine-dependent and machine-independent code in BSD which would improve the system's future portability.

Apart from portability, the CSRG worked on an implementation of the OSI network protocol stack, improvements to the kernel virtual memory system and (with Van Jacobson of LBL) new TCP/IP algorithms to accommodate the growth of the internet.

Until then, all versions of BSD incorporated proprietary AT&T Unix code and were, therefore, subject to an AT&T software license. Source code licenses had become very expensive and several outside parties had expressed interest in a separate release of the networking code, which had been developed entirely outside AT&T and would not be subject to the licensing requirement. This led to Networking Release 1 (Net/1), which was made available to non-licensees of AT&T code and was freely redistributable under the terms of the BSD license. It was released in June 1989.

4.3BSD-Reno came in early 1990. It was an interim release during the early development of 4.4BSD, and its use was considered a "gamble", hence the naming after the gambling center of Reno, Nevada. This release was explicitly moving towards POSIX compliance,[6] and, according to some, away from the BSD philosophy (as POSIX is very much based on System V, and Reno was quite bloated compared to previous releases). Among the new features were an NFS implementation from the University of Guelph and support for the HP 9000 range of computers, originating in the University of Utah's "HPBSD" port.

In August 2006, Information Week magazine rated 4.3BSD as the "Greatest Software Ever Written".[8] They commented: "BSD 4.3 represents the single biggest theoretical undergirder of the Internet."

After Net/1, BSD developer Keith Bostic proposed that more non-AT&T sections of the BSD system be released under the same license as Net/1. To this end, he started a project to reimplement most of the standard Unix utilities

without using the AT&T code. For example, vi, which had been based on the original Unix version of ed, was rewritten as nvi (new vi). Within eighteen months, all of the AT&T utilities had been replaced, and it was determined that only a few AT&T files remained in the kernel. These files were removed, and the result was the June 1991 release of Networking Release 2 (Net/2), a nearly complete operating system that was freely distributable. Net/2 was the basis for two separate ports of BSD to the Intel 80386 architecture: the free 386BSD by William Jolitz and the proprietary BSD/386 (later renamed BSD/OS) by Berkeley Software Design (BSDi). 386BSD itself was short-lived, but became the initial code base of the NetBSD and FreeBSD projects that were started shortly thereafter.

BSDi soon found itself in legal trouble with AT&T's Unix System Laboratories (USL) subsidiary, then the owners of the System V copyright and the Unix trademark. The USL v. BSDi lawsuit was filed in 1992 and led to an injunction on the distribution of Net/2 until the validity of USL's copyright claims on the source could be determined. The lawsuit slowed development of the free-software descendants of BSD for nearly two years while their legal status was in question, and as a result systems based on the Linux kernel, which did not have such legal ambiguity, gained greater support. Although not released until 1992, development of 386BSD predated that of Linux. Linus Torvalds has said that if 386BSD or the GNU kernel had been available at the time, he probably would not have created Linux.

The lawsuit was settled in January 1994, largely in Berkeley's favor. Of the 18,000 files in the Berkeley distribution, only three had to be removed and 70 modified to show USL copyright notices. A further condition of the settlement was that USL would not file further lawsuits against users and distributors of the Berkeley-owned code in the upcoming 4.4BSD release.

In June 1994, 4.4BSD was released in two forms: the freely distributable 4.4BSD-Lite contained no AT&T source, whereas 4.4BSD-Encumbered was available, as earlier releases had been, only to AT&T licensees.

The final release from Berkeley was 1995's 4.4BSD-Lite Release 2, after which the CSRG was dissolved and development of BSD at Berkeley ceased. Since then, several variants based directly or indirectly on 4.4BSD-Lite (such as FreeBSD, NetBSD, OpenBSD and DragonFly BSD) have been maintained. In addition, the permissive nature of the BSD license has allowed many other operating systems, both free and proprietary, to incorporate BSD code. For example, Microsoft Windows has used BSD-derived code in its implementation of TCP/IP[citation needed] and bundles recompiled versions of BSD's command-line networking tools since Windows 2000.[11] Also Darwin, the system on which Apple's Mac OS X is built, is a derivative of 4.4BSD-Lite2 and FreeBSD. Various commercial UNIX operating systems, such as Solaris, also contain varying amounts of BSD code.

BSD（Berkeley Software Distribution，伯克利软件套件）是一个 UNIX 的衍生系统 (UNIX 变种)，1970 年代由伯克利加州大学（University of California, Berkeley）的学生比尔·乔伊开创。

> BSD 也被用来代表其衍生出的各种套件，BSD 系统更为类似于 UNIX。事实上 BSD 就是传统 UNIX 的直接衍生品，而 Linux 则是一个松散的基于 UNIX 衍生品（Minix）而新创建的一个操作系统内核及其工具。

BSD 常被当作工作站级别的 Unix 系统，这得归功于 BSD 使用授权非常地宽松，许多 1980 年代成立的计算机公司，不少都从 BSD 中获益，比较著名的例子如 DEC 的 Ultrix，以及 Sun 公司的 SunOS。1990 年代，BSD 很大程度上被 System V 4.x 版以及 OSF/1 系统所取代，但其开源版本被采用，促进了因特网的开发。

虽然贝尔实验室隶属于 AT&T，但是 AT&T 此时对于 UNIX 是采取开放的态度，此外 UNIX 是以高阶的 C 语言编写成的，理论上是具有可移植性的。所以只要取得 UNIX 的源代码，并且针对主机的特性修改源代码（Source Code）就可能将 UNIX 移植到另一台不同的主机上了，所以在 1973 年以后 UNIX 便得以与学术界合作开发，最重要的接触就是与加州柏克莱（Berkeley）大学的合作。

柏克莱大学的 Bill Joy 在取得了 UNIX 的核心源代码后，自行修改成适合自己机器的版本，并且同时增加了很多工具软件与编译程序，最终将其命名为 Berkeley Software distribution（BSD）[1]。这个 BSD 是 UNIX 很重要的一个分支，Bill Joy 也是 Sun 公司的创办者，Sun 公司即是以 BSD 开发的核心进行自己的商业 UNIX 版本[2]——SPARC 的开发的，后来可以安装在 x86 硬件架构上面 FreeBSD 也是由 BSD 改版而来。

BSD 开创了现代计算机的潮流。伯克利的 Unix 率先包含了库，以支持互联网协议栈（Stack）、伯克利套接字（sockets）。通过将套接字与 Unix 操作系统的文件描述符相整合，库用户通过计算机网络读写数据，跟直接在磁盘上操作一样容易。AT&T 实验室最后也发布了他们的 STREAMS 库，在软件栈中引入了类似的功能，虽然结构层有所改进，但由于套接字库已经使用广泛，另外由于少了对开放套接字的轮询功能（类似于伯克利库中的 select 调用），使得将软件移植到这个新的 API 很困难。

> BSD 项目维护的是整个操作系统，而 Linux 则只是主要集中在单一的内核上面。这点确实是需要注意的，虽然现在这两个系统上都运行着许多相同的软件。

跟 AT&T Unix 一样，BSD 也采用单内核，这意味着内核中的设备驱动，在核心态下运行，从而作为操作系统的核心部分。

时至今日，BSD 仍在学术机构，乃至许多商业或自由产品的高科技实验中，继续被用作试验平台，甚至在嵌入式设备中，其使用也在增长。由于 BSD 设计出众，代码编写清晰，包括它的文档（特别是参考文档，常被称为“man pages”），使得这样的系统，几乎成了程序员眼中的乐土。

许多公司都使用 BSD 衍生出的代码，如此便可以支持他们的知识产权，许多自由软件，如 Linux、GNU 工程都遵照 GNU General Public License，与之相比，BSD 许可协议要更加灵活。当然，这也导致人们的机器上运行着一些 BSD 软件，但自己却并不知情。有兴趣的话，可以试着找找符号“University of California, Berkeley”，比如在产品文档内，二进制代码中的静态数据段，或者 ROM 中，还有通过一些产品的用户界面看看“about”（关于）内容。

有意思的是，通过一个二进制兼容层（compatibility layer），在 BSD 操作系统上可以运行相同构架下其他操作系统上的原程序。这比模拟器要快得多，通过这个方法，针对 Linux 的应用程序，也可以在 BSD 上全速运行。所以，BSD 不仅适合作为服务器，也可作为工作站来使用，众所周知，现在针对 Linux 的商业或封闭源码软件越来越多了。管理员也可以将一些原本只用于商业 UNIX 变种的专属软件，转

[1] 目前被称为纯种的 UNIX 指的就是 AT&T System V 以及 BSD 这两套。
[2] BSD 的早期版本被用作组建 Sun 公司的 SunOS，造就了 Unix 工作站的第一波热潮。

移到 BSD，这样在保持原有功能的同时，操作系统更趋现代，可继续使用这些软件，直到有更好的替代。

最初的 UNIX 套件源自 1970 年代的贝尔实验室，操作系统中包含源码，这样研究人员以及大学都可以参与修改扩充。1974 年，第一个伯克利的 Unix 系统被安装在 PDP-11 机器上，计算机科学系而后将其用作扩展研究。

其他大学开始对伯克利的软件感兴趣，在 1977 年，伯克利的研究生 Bill Joy 将程序整理到磁带上，作为 first Berkeley Software Distribution（1BSD）发行。1BSD 被作为第六版 Unix 系列，而不是单独的操作系统。主要程序包括 Pascal 编译器，以及 Joy 的 ex 行编辑器。

Second Berkeley Software Distribution（2BSD）于 1978 年发布，除了对 1BSD 中的软件进行升级，还包括了 Joy 写的两个新程序：vi 文本编辑器（ex 的可视版本），以及 C Shell。这两个新添的程序，在 Unix 系统中至今仍被使用。

2BSD 以后的版本逐渐从 PDP-11 结构向 VAX 计算机移植。最新的 2.11BSD 于 1992 年发布，更新维护一直持续到 2003 年。

1978 年，伯克利安装了第一台 VAX 计算机，但将 Unix 移植到 VAX 构架的 UNIX/32V，并没有利用 VAX 虚拟内存的能力。伯克利的学生重写了 32V 的大部分内核，以实现虚拟内存的支持。1979 年，3BSD 诞生了，这个新系统完整包括了一个新内核、从 2BSD 移植到 VAX 的工具，还有 32V 原来的工具。

3BSD 的成功使得 Defense Advanced Research Projects Agency（DARPA，美国国防部高级研究规划署）决定资助伯克利的 Computer Systems Research Group（CSRG，计算机系统研究组），以开发一个 Unix 标准平台，以供 DARPA 未来的研究。1980 年 10 月，CSRG 发布了 4BSD，该版本对 3BSD 有许多改进。

相较于 VAX 机器的主流系统 VMS，用户对 BSD 时有批评，1981 年 6 月，终于发布了 4.1BSD。Bill Joy 大幅度提高了 4.1BSD 内核的性能，可以跟 VMS 在多个平台上媲美。为了避免与 AT&T 的 UNIX System V（UNIX 第五版）混淆，这个版本没有取名为 5BSD。

以后 4.2BSD 历经两年，实现了多项重大改进后才得以问世。之前有三个中间版本相继推出：4.1a 引入了修改版的 BBN 预试中 TCP/IP；4.1b 引入了由 Marshall Kirk McKusick 实现的新型 Berkeley Fast File System（FFS）；4.1c 是 4.2BSD 开发最后几个月的过渡版。

1983 年 8 月，4.2BSD 正式发布。这是 1982 年 Bill Joy 离开前去创建 Sun 公司后的第一个版本，此后 Mike Karels 和 Marshall Kirk McKusick 一直负责领导该项目。值得一提的是，这次 BSD 小恶魔正式出场，最初是 Marshall Kirk McKusick 的画作，出现在打印好的文档封面上，由 USENIX 发行。

1986 年 6 月，4.3BSD 发布。该版本主要是将 4.2BSD 的许多新贡献作性能上的提高，原来的 4.1BSD 没有很好地协调。在该版本之前，BSD 的 TCP/IP 实现已经跟 BBN 的官方实现有较大差异。经过数月测试后，DARPA 认为 4.2BSD 更合适，所以在 4.3BSD 中作了保留。

4.3BSD 后，BSD 逐渐抛开老式的 VAX 平台。Computer Consoles 有限公司开发的 Power 6/32 平台（代号为”Tahoe”），当时看来大有可为，但不久即被他们的开发员所遗弃。然后，1988 年 6 月移植的 4.3BSD-Tahoe 却表现不俗，BSD 将依赖于机器跟不依赖于机器的代码分离，为未来系统的可移植性打下了良好的基础。

到此为止，所有的 BSD 版本混合了专属的 AT&T UNIX 代码，这样继续使用就需要从 AT&T 获得许可证。源码许可证当时非常地昂贵，几个其他组织对单独的网络代码版感兴趣，完全独立于 AT&T，这样就可不受许可证的支配。1989 年 6 月，Networking Release 1（Net/1）诞生了，没有 AT&T 授权也能使用，可遵照 BSD 许可证进行自由再发布。

1990 年初，推出了 4.3BSD-Reno。该版本是 4.4BSD 早期开发的过渡版，使用该版本被戏称为是一种赌博，因为 Reno 就是内华达州的赌城雷诺。

Net/1 以后，Keith Bostic 提议，BSD 系统中应该有更多的非 AT&T 部分，以 Net/1 的协议发布。随后，他开始一个项目，着手重新实现一些 Unix 标准工具，其中不使用原来的 AT&T 代码。例如，Vi，也就是基于最初 UNIX 上 ed 的编辑器，被重写为 nvi（new vi）。18 个月后，所有 AT&T 的工具被替换，剩下的只是存留在内核的一些 AT&T 文件。残余文件被剔除后，1991 年 6 月，Net/2 诞生了，这是一个全新的操作系统，并且可以自由发布。

Net/2 成为 Intel 80386 构架上两种移植的主要组成，包括由 William Jolitz 负责，自由的 386BSD；以及专属的 BSD/OS，由 Berkeley Software Design（BSDi）负责。386BSD 本身虽然短命，但不久即成为 NetBSD 和 FreeBSD 原始代码的基础。

BSDi 很快就与 AT&T 的 UNIX Systems Laboratories（USL）附属公司产生了法律纠纷，后者将拥有 System V 版权，以及 UNIX 商标。1992 年，USL 正式对 BSDi 提起诉讼，这导致 Net/2 发布被中止，直到其源码能够被鉴定为符合 USL 的版权。

由于最后判决悬而未决，这桩法律诉讼将 BSD 后裔的开发（特别是自由软件）延迟了两年，这导致没有法律问题的 Linux 内核获得了极大的支持。Linux 跟 386BSD 的开发几乎同时起步，林纳斯·托瓦兹曾说，当时如果有基于 386 的自由 Unix-like 操作系统，他就可能不会创造 Linux。尽管无法预料这给以后的软件业究竟造成了什么样的影响，但有一点可以肯定，Linux 更加丰富了这块土壤。

这桩诉讼在 1994 年 1 月了结，更多地满足了伯克利的利益。伯克利套件的 18,000 个文件中，只有 3 个文件要求删除，另有 70 个文件要求修改，并显示 USL 的版权说明。这项调解另外要求，USL 不得对下面的 4.4BSD 提起诉讼，不管是用户还是伯克利代码的分发者。

1994 年 6 月，4.4BSD 以两种形式发布：可自由再发布的 4.4BSD-Lite，不包含 AT&T 源码；另有 4.4BSD-Encumbered，跟以前的版本一样，遵照 AT&T 的许可证。

伯克利的最终版本是 1995 年的 4.4BSD-Lite Release 2，而后 CSRG 解散，在伯克利的 BSD 开发告一段落。在这之后，几种基于 4.4BSD 的套件（比如 FreeBSD、OpenBSD 和 NetBSD）得以继续维护。

另外，由于 BSD 许可证的宽容，许多其他的操作系统，不管是自由还是专属，都采用了 BSD 的代码。例如，Microsoft Windows 在 TCP/IP 的实现上引入了 BSD 代码；经过重新编译，在当前 Windows 版本中，还采用了许多 BSD 命令行下的网络工具。当前的 BSD 操作系统变种支持各种通用标准，包括 IEEE、ANSI、ISO 以及 POSIX，同时保持了传统 BSD 的良好风范。

> 与 GPL 相比，BSD 许可证的限制则要少得多，它甚至允许二进制包成为唯一的发行源。这就是 GPL 与 BSD 的核心差异，可以这样理解：GPL 许可证让您有权拥有任何你想要使用该软件的方法，但你必须确保提供源代码给下一个使用它的人（包括你对它的改变部分），而 BSD 许可证并不是要求你必须那么做。

由于 BSD 的开发方式的原因，可以利用 SD 下是用 `freebsd-update fetch update` 命令），或者也可以下载整个源代码树，然后通过编译来升级。在 Linux 中也可以通过内置的包管理系统来升级系统。前者（BSD）仅更新基本系统，而后者（Linux）则会升级整个系统。不过 BSD 升级到最新的基本系统并不意味着所有的附加软件包也将会被更新，而 Linux 升级的时候，所有的软件包都会被升级。

除了最基本的系统软件，FreeBSD 还提供了一个拥有成千上万广受欢迎的程序组成的软件的 Ports Collection，包括从 http(WWW) 服务器到游戏、程序设计语言、编辑器等。完整的 Ports Collection 大约需要 500 MB 的存储空间，所有的只提供对原始代码的“修正”，这使得我们能够容易地更新软件，并且减少了老旧的 1.0 Ports Collection 对硬盘空间的浪费。

> BSD 是一个包括众多工具的基本系统，包括 libc 在内都是基本系统的一部分，这些组件都被作为一个基本系统被一起开发和打包。

要编译一个 port，只要切换到想要安装的程序的目录，输入 `make install`，然后让系统去做剩下的事情。要编译的每一个程序完整的原始代码可以从 CD-ROM 或本地 FTP 获得，所以只需要编译想要的软件具备足够的磁盘空间。几乎大多数的软件都提供了事先编译好的“package”以方便安装，对于那些不希望从源代码编译他们自己的 ports 的人只要使用一个简单的命令 (`pkg_add`) 就可以安装。

第 4 章

Solaris

Solaris 原先是太阳微系统公司研制的类 Unix 操作系统，在 Sun 公司被 Oracle 并购后，称作 Oracle Solaris。目前最新版为 Solaris 11。早期的 Solaris 是由 BSDUnix 发展而来。这是因为太阳公司的创始人之一，比尔·乔伊（Bill Joy）来自柏克莱加州大学（U.C.Berkeley）。但是随着时间的推移，Solaris 现在在接口上正在逐渐向 System V 靠拢。2005 年 6 月 14 日，Sun 公司将正在开发中的 Solaris 11 的源代码以 CDDL 许可开放，这一开放版本就是 OpenSolaris。2010 年 8 月 23 日，OpenSolaris 项目被 Oracle 中止。2011 年 11 月 9 日，发布 Solaris 11。

Sun 的操作系统最初叫做 SunOS[1]，SunOS 5.0 开始，SUN 的操作系统开发开始转向 System V 4，并且有了新的名字叫做 Solaris 2.0；Solaris 2.6 以后，SUN 删除了版本号中的“2”，因此，SunOS 5.10 就叫做 Solaris 10。Solaris 的早期版本后来又被重命名为 Solaris 1.x. 所以”SunOS” 这个词被用做专指 Solaris 操作系统的内核，因此 Solaris 被认为是由 SunOS，图形化的桌面计算环境，以及它网络增强部分组成。

Solaris 运行在两个平台：Intel x86 及 SPARC/UltraSPARC。后者是太阳工作站使用的处理器。因此，Solaris 在 SPARC 上拥有强大的处理能力和硬件支持，同时 Intel x86 上的性能也正在得到改善。对两个平台，Solaris 屏蔽了底层平台差异，为用户提供了尽可能一样的使用体验。早期的 Solaris 主要用于 Sun 工作站上。不过，随着 Sun 让 Solaris 可以免费下载和 OpenSolaris 的发布，Solaris/OpenSolaris 除了作为服务器／工作站的用途外，已经开始可以作为 Desktop 用途。目前各大软件、应用程序厂商对 SPARC 平台的支持尚算良好，但对 x86 平台的 Solaris 多半都不支持。这也是 x86 用户面临的困境之一。

最新发布的 Solaris10 包含若干创新技术，包括 ZFS、DTrace、Solaris Zones（Container）、预测性自愈等。其中一些以往只可能在专业服务器等具有相关硬件的大型机器上才可能得到支持，但 Solaris10 使得任何一台普通 PC 都可以具有这些能力。

Solaris 支持多种系统架构：SPARC、x86 及 x64（即 AMD64 及 EM64T 处理器）。在版本 2.5.1 的时候，Solaris 曾经一度被移植到 PowerPC 架构，但是后来又在这一版本正式发布时被删去。与 Linux 相比，Solaris 可以更有效地支持对称多处理器，即 SMP 架构。Sun 同时宣布将在 Solaris 10 的后续版本中提供 Linux 运行环境，允许 Linux 二进制程序直接在 Solaris x86 和 x64 系统上运行，目前，这一技术已通过 Solaris Zone 的一个特殊实现（BrandZ）得到支持。

Solaris 传统上与基于 Sun SPARC 处理器的硬件体系结构结合紧密，在设计上和市场上经常捆绑在一起，整个软硬件系统的可靠性和性能也因此大大增强。然而 SPARC 系统的成本和价格通常要高于 PC 类的产品，这成为 Solaris 进一步普及的障碍。可喜的是，Solaris 对 x86 体系结构的支持正得到大大加强，特别是 Solaris 10 已经能很好地支持 x64(AMD64/EM64T) 架构。Sun 公司已推出自行设计的基于 AMD64 的工作站和服务器，并随机附带 Solaris 10。

第一个 Solaris 的桌面环境是 OpenWindows。紧接着是 Solaris 2.5 的 CDE。在 Solaris 10 中，Sun 又推出了基于 GNOME 的 Java Desktop System，另外也支持 KDE、XFCE、WindowMaker 等。Solaris 11 采用 GNOME。

Solaris 已开放其部分源代码，但是由于 Sun 公司的源代码许可证，Solaris “不是” 自由软件，而 OpenSolaris 才是。Solaris 和 OpenSolaris 为两个“不同”的操作系统。

Solaris 的大多数源代码已经在 CDDL 的许可下在 OpenSolaris 开源项目中发布 [5]。二进制和源代

[1]SunOS 仍旧用来称呼 Solaris 的核心。SunOS 的版本号是以 5.Solaris 版本号来表示。例如，最新的 Solaris 发布版本，Solaris 10，在 SunOS 5.10 上运行。Solaris 的 man 手册是以 SunOS 为标记的，启动的时候也显示它，但是 “SunOS” 这个词不再用于 Sun 的市场文档中。

码目前都可以被下载和许可而无需任何费用。Sun 的 Common Development and Distribution License 被选择用做 OpenSolaris 的许可，并通过了 Open Source Initiative 评审和批准，但其授权条款与时下流行的 GPL 互不兼容。

OpenSolaris 于 2005 年 6 月 14 日正式启动，源代码来自当时的 Solaris 开发版本。目前 OpenSolaris 项目已被并吞 Sun 公司的 Oracle 中止，由社区发起的 Illumos 计划继承。

第 5 章

UNIX-like

5.1 MINIX

MINIX is an inexpensive minimal UNIX-like operating system, designed for education in computer science, written by Andrew S. Tanenbaum. Starting with version 3 in 2005, MINIX became free and was redesigned for ”serious” use.

MINIX is a Unix-like computer operating system based on a microkernel architecture created by Andrew S. Tanenbaum for educational purposes; MINIX also inspired the creation of the Linux kernel.

MINIX (from ”mini-Unix”) was first released in 1987, with its complete source code made available to universities for study in courses and research. It has been free and open source software since it was re-licensed under the BSD license in April 2000.

5.1.1 Histroy

Andrew S. Tanenbaum created MINIX at Vrije Universiteit in Amsterdam to exemplify the principles conveyed in his textbook, Operating Systems: Design and Implementation (1987).

An abridged 12,000 lines of the C source code of the kernel, memory manager, and file system of MINIX 1.0 are printed in the book. Prentice-Hall also released MINIX source code and binaries on floppy disk with a reference manual. MINIX 1 was system-call compatible with Seventh Edition Unix.

Tanenbaum originally developed MINIX for compatibility with the IBM PC and IBM PC/AT microcomputers available at the time.

MINIX 1.5, released in 1991, included support for MicroChannel IBM PS/2 systems and was also ported to the Motorola 68000 and SPARC architectures, supporting the Atari ST, Commodore Amiga, Apple Macintosh and Sun SPARCstation computer platforms. There were also unofficial ports to Intel 386 PC compatibles (in 32-bit protected mode), National Semiconductor NS32532, ARM and INMOS transputer processors. Meiko Scientific used an early version of MINIX as the basis for the MeikOS operating system for its transputer-based Computing Surface parallel computers. A version of MINIX running as a user process under SunOS and Solaris was also available, a simulator called SMX.

Demand for the 68k-based architectures waned, however, and MINIX 2.0, released in 1997, was only available for the x86 and Solaris-hosted SPARC architectures. It was the subject of the second edition of Tanenbaum's textbook, co-written with Albert Woodhull and was distributed on a CD-ROM included with the book. MINIX 2.0 added POSIX.1 compliance, support for 386 and later processors in 32-bit mode and replaced the Amoeba network protocols included in MINIX 1.5 with a TCP/IP stack. Unofficial ports of MINIX 2.0.2 to the 68020-based ISICAD Prisma 700 workstation[4] and the Hitachi SH3-based HP Jornada 680/690 PDA[5] were also developed.

Minix-vmd is a variant of MINIX 2 for Intel IA-32-compatible processors, created by two Vrije Universiteit researchers, which adds virtual memory and support for the X Window System.

MINIX 3 was publicly announced on 24 October 2005 by Andrew Tanenbaum during his keynote speech on top of the ACM Symposium Operating Systems Principles conference. Although it still serves as an example for the

new edition of Tanenbaum and Woodhull's textbook, it is comprehensively redesigned to be "usable as a serious system on resource-limited and embedded computers and for applications requiring high reliability."[6]

MINIX 3 currently supports only IA-32 architecture PC compatible systems. It is available in a Live CD format that allows it to be used on a computer without installing it on the hard drive, and in versions compatible with hardware emulation/virtualization systems, including Bochs, QEMU, VMware Workstation/Fusion, VirtualBox and Microsoft Virtual PC.

Version 3.1.5 was released 5 November 2009. It contains X11, emacs, vi, cc, gcc, perl, python, ash, bash, zsh, ftp, ssh, telnet, pine, and over 400 other common Unix utility programs. With the addition of X11, this version marks the transition away from a text-only system. It can also withstand driver crashes. In many cases it can automatically replace drivers without affecting running processes. This feature will be improved in future releases. In this way, MINIX is self-healing and can be used in applications demanding high reliability. MINIX 3 also has support for virtual memory management, making it suitable for desktop OS use.[7] Desktop applications such as Firefox and OpenOffice.org are not yet available for MINIX 3 however.

With the creation of MINIX 3, and its transition to a graphical interface, some commercial software and hardware developers have started to implement some systems with MINIX in the late 2000s.

As of version 3.2.0, the userland was mostly replaced by that of NetBSD and support from pkgsrc became possible, increasing the available software applications that MINIX can use. Clang replaced the previous compiler (with GCC optionally supported), and GDB, the GNU debugger, was ported.

5.1.2 Licensing

At the time of its original development, the license for MINIX was considered to be rather liberal. Its licensing fee was very small ($69) compared to those of other operating systems. Although Tanenbaum wished for MINIX to be as accessible as possible to students, his publisher was not prepared to offer material (such as the source code) that could be copied freely, so a restrictive license requiring a nominal fee (included in the price of Tanenbaum's book) was applied as a compromise. This prevented the use of MINIX as the basis for a freely distributed software system.

When free and open source Unix-like operating systems such as Linux and 386BSD became available in the early 1990s, many volunteer software developers abandoned MINIX in favor of these. In April 2000, MINIX became free/open source software under a permissive free software license,[13] but by this time other operating systems had surpassed its capabilities, and it remained primarily an operating system for students and hobbyists.

Minix 原来是荷兰阿姆斯特丹自由大学计算机科学系的塔能鲍姆教授(Prof. Andrew S. Tanenbaum)所发展的一个类 Unix 操作系统。

MINIX 是一个轻量的小型类 UNIX 操作系统,采用微核心设计,而且它启发了 Linux 操作系统的开发,作者是安德鲁·斯图尔特·塔能鲍姆。从第三版开始,MINIX 是自由软件,而且被“严重的”重新设计。

MINIX 取自 Mini UNIX 的缩写。与 Xinu、Idris、Coherent 和 Uniflex 等类 Unix 操作系统类似,派生自 Version 7 Unix,但并没有使用任何 AT&T 的代码。第一版于 1987 年发布,只需要购买它的磁片,就提供完整的源代码给大学系所与学生,做为授课及学习之用。2000 年 4 月,重新以 BSD 许可证发布,成为开放源代码软件。

1979 年的版权声明中影响最大的当然就是高校与 UNIX 相关的教学人员,因为 AT&T 的政策改变,在 Version 7 Unix 推出之后,发布新的使用条款,将 UNIX 源代码私有化,在大学中不再能使用 UNIX 源代码。由于没有 UNIX kernel 的源代码便无法教导学生认识 UNIX 了,这问题对于 Andrew Tanenbaum(谭宁邦)教授来说实在是很伤脑筋的。不过既然 1979 年的 UNIX 第七版可以在 Intel 的 x86 架构上进行移植,那么是否意味着可以将 UNIX 改写并移植到 x86 上面呢?

谭宁邦教授于是自己动手开发了 Minix 这个 UNIX-like 的核心程序。为了避免版权纠纷,谭宁邦在不使用任何 AT&T 的源代码前提下开发了 MINIX,并且强调 MINIX 必须能够与 UNIX 相容才行。

谭宁邦在 1984 年开始编写核心程序,到了 1986 年终于完成,并于次年出版 Minix 相关书籍同时与新闻组相结合。这个 Minix 版本比较有趣的地方是,它并不是完全免费的,无法在网络上提供下载,必须要通过磁盘/磁带购买才行,虽然真的很便宜,不过因为没有在网络上流传,所以 Minix 的传播速度并不快。

Minix 以 C 语言写成,与 Version 7 Unix 兼容,全部的代码共约 12,000 行,并置于他的著作《操作系统:设计与实现》(Operating Systems: Design and Implementation, ISBN 0-13-637331-3)的附录里作为示例。Minix 的系统要求在当时来说非常简单,只要三片磁片就可以起动。Minix 原始是设计给 1980 年代到 1990 年代的 IBM PC 和 IBM PC/AT 兼容计算机上运行,主要运作于 16-bits 的 Intel 8080 平台,以软盘起动。此外购买时随磁盘还会附上 Minix 的源代码,这意味着用户可以学习 Minix 的核心程序设计概念,这个特色对于 Linux 的初始开发阶段有很大的帮助。

因为开发者仅有谭宁邦教授,加上谭宁邦始终认为 Minix 主要用在教育用途上面,所以对于 Minix 是点到为止,所以 Minix 很受欢迎没错,不过用户的要求/需求的声音可能就没有办法上升到比较高的地方。

1.5 版也有移植到已 Motorola 68000 系列 CPU 为基础的计算机上(如 Atari ST, Amiga, 和早期的 Apple Macintosh)和以 SPARC 为基础的机器(如升阳(Sun)公司的工作站)。

1997 年,随着教科书改版,塔能鲍姆发布 Minix 2,在这版中,它改成可以在 Intel 80386 等 x86 平台上运作,从硬盘上开机。2004 年,塔能鲍姆重新架构与设计了整个系统,更进一步的将程序模块化,推出 Minix 3。

全套 Minix 除了启动的部份以汇编语言编写以外,其他大部份都是纯粹用 C 语言编写。分为:内核、内存管理及文件系统三部份。

Minix 在设计之初,为了使程序简化,它将程序模块化,如文件系统与存储器管理,都不是在操作系统内核中运作,而是在用户空间运作。至 Minix 3 时,连 IO 设备都被移到用户空间运作。另一个特点,则是 Minix 主要目的是用于教学,因此代码撰写上极力重视简洁与可读性。

在授权方式上,Minix 的版权声明在早期被认为是相当自由的:塔能鲍姆教授在希望拿 Minix 作为一个公开的教材与出版社希望保护代码著作权的平衡下,它只要求一个相当低的授权费。但因为它并不是一个开放源代码的授权方案,所以志愿工作者在以 GPL 方式散布的 Linux 核心出现后就多转向 Linux 平台。而 Unix 也在 BSD 与 AT&T 达成协议后,出现了以 BSD 许可证授权散布的 FreeBSD 开放平台。Minix 虽然在 2000 年改用 BSD 许可证授权,但这时其它的操作系统在功能上大幅超越了它,而它失去了发展成一个广泛使用的操作系统的机会,只留下,如塔能鲍姆教授原来期望的,作为一个开

放的教材的用途。直到 Minix 3 出现后，Minix 才又改头换面，现在它是一个面向小型系统的可靠操作系统。

5.2 FreeBSD

FreeBSD 是一种自由的类 Unix 操作系统，是由经过 BSD、386BSD 和 4.4BSD 发展而来的类 Unix 的一个重要分支[1]。FreeBSD 拥有超过 200 名活跃开发者和上千名贡献者。

FreeBSD 被认为是自由操作系统中的不知名的巨人。它不是 Unix，但如 Unix 一样运行，兼容 POSIX。作为一个操作系统，FreeBSD 被认为相当稳健可靠。

FreeBSD 是以一个完善的操作系统的定位来做开发。其核心、驱动程序以及所有的用户层 (Userland) 应用程序 (比方说是 Shell) 均由同一源代码版本控制系统保存 (目前使用 Subversion 并与 CVS 兼容)。相较于另一知名的的操作系统 Linux，其核心为一组开发人员设计，而用户应用程序则交由他人开发(例如 GNU 计划)，最后再由其他团体集成并包装成 Linux 包。

FreeBSD 默认是无桌面环境的命令行界面，想要使用桌面环境必须自行安装，或是使用 PC-BSD 之类的桌面发布版。

FreeBSD 发展采用 Core Team 的方式。Core Team 的成员决定整个 FreeBSD 计划的大方向，对于开发者间的问题有最后的决定权，其他的开发者也可以提交建议或是他们修改过的代码，但是 Core Team 保留最终的决定权，决定是否将这功能放进 FreeBSD。这种方式与 Linux 发展大相径庭。

- Contributor
  也可以说是 Submitter，无 FreeBSD 的 CVS 的访问权限，但是可以通过其它的方式，例如提交 Problem Reports 或是在 Mailing list 上面参与讨论，来对 FreeBSD 做出贡献。
- Committer
  有对 FreeBSD 的 CVS 及 Subversion 访问的权限，可以将他的代码或是文件送到版本库里面。一个 committer 必须要在过去的 12 个月中有 commit 的动作。而一个活跃的 committer 指在每个月至少都有一次以上的 commit 动作。
  虽然说没有必要限制一个有 commit 权限的人只能在代码树中可以访问的地方，但是如果一个 committer 要在他／她没有做出贡献或是不熟悉的地方，他／她必须要读那个地方的历史记录，还有 MAINTAINER 文件，确认这个部份的维护者对于更改这边的代码有没有什么特殊的要求。
- Core Team
  Core Team 成员由 committer 互相推选出来，是整个 FreeBSD 计划的领导人，他们提升活跃的 contributor 成为 committer，还有可以指派“Hat”(指在计划中负责一些特定工作或领域的人)，也是对于决定整个计划的大方向的最后仲裁者。在 2004 年 7 月 1 日，core team 有 9 位成员，而 core team 选举每两年举行一次。

FreeBSD 的发展始于 1993 年，取源于 386BSD。然而，因为 386BSD 源代码的合理性受到质疑以及 Novell (当时 UNIX 的版权拥有者) 与柏克莱接连而来的诉讼，FreeBSD 在 1995 年 1 月发布的 2.0-RELEASE 中以柏克莱加州大学的 4.4BSD-Lite Release 全面改写。

FreeBSD 2.0 最值得注意的部份也许是对卡内基美隆大学的 Mach Virtual Memory 系统翻修以及 FreeBSD Ports system 的发明。前者对于高负荷的系统优化，后者则是创建了一套简单且强大的机制维护第三方软件。有不少大型站台都使用 FreeBSD，像是 cdrom.com(一个巨大的软件收集站台)，Hotmail 以及 Yahoo。

FreeBSD 3.0 则引入了 ELF binary 格式，并开始支持多 CPU 系统(SMP，Symmetric multiprocessing)以及 64 位 Alpha 平台。3.x 对于系统做了非常多的改革，这些措施在当时并没有带来好处，但却是 4.X 成功的基石。

4.0-RELEASE 于 2000 年 3 月发布，最后一个版本 4.11-RELEASE 于 2005 年 1 月发布，并支持到 2007

[1] FreeBSD 是一个支持 Intel (x86 和 Itanium®)，AMD64，Sun UltraSPARC® 计算机的基于 4.4BSD-Lite 的操作系统。

年 1 月。FreeBSD 4 也是 FreeBSD 最长寿的主版本，在 FreeBSD 4 所发展出来的 kqueue 也被移植到各种不同 BSD 平台。

最后一个版本的 FreeBSD 5 是 5.5，是在 2006 年五月发布的。在 FreeBSD 4 的 SMP 架构下，在同一时间内只允许一个 CPU 进入核心（即 Giant Lock），FreeBSD 5 最大的改变在于改善底层核心 Locking 机制，审视并改写核心代码，使得不同的 CPU 可以同时进入系统核心，藉以增加效率。

另外一个重大的改变在于自 5.3 开始支持 m:n 线程的 KSE（Kernel Scheduled Entities），表示 m 个用户线程共用 n 个核心线程的模式。这个版本的许多贡献是由于商业化版本的 BSD OS 团队的支持。

FreeBSD 6 为一个 -STABLE 发展版本，FreeBSD 6.3 在 2008 年 1 月 18 日发布 [9]，这个版本主要针对软件的更新，并加入 lagg（可以对多张网卡操作）的支持，并引入重新改写的 unionfs。FreeBSD 6.4 在 2008 年 11 月 28 日发行。

FreeBSD 7 为目前第二个 -STABLE 发展版本，在 2007 年 6 月 19 日进入发布程序，2008 年 2 月 27 日 7.0-RELEASE 正式发布。2010 年 03 月 23 日 FreeBSD 7.3-RELEASE 正式发布。新增的功能包括了：

- SCTP
- 日志式 UFS 文件系统：gjournal
- 移植 DTrace
- 移植 ZFS 文件系统
- 使用 GCC4
- 对 ARM 与 MIPS 平台的支持
- 重写过的 USB stack
- Scalable concurrent malloc 实现
- ULE 调度表 2.0（SCHED_ULE），并修改加强为 SCHED_SMP，在交付至 CVS 时的正式名称为 ULE 3.0[2]。
- Linux 2.6 模拟层
- Camellia Block Cipher
- ZFS 的运行

FreeBSD 8 于 2009 年 11 月 27 日发布，2010 年 07 月 24 日发布 FreeBSD 8.1-RELEASE, [24] 增加如下新特性：

- 虚拟化方面：Xen DOM-U、VirtualBox guest 及 host 支持、层次式 jail。
- NFS：对 NFSv3 GSSAPI 的支持，以及试验性的 NFSv4 客户端和服务器。
- 802.11s D3.03 wireless mesh 网络，以及虚拟 Access Point 支持。
- ZFS 不再是试验性的了。
- 基于 Juniper Networks 提供 MIPS 处理器的实验性支持。
- SMP 扩展性的增强，显著改善在 16 核心处理器系统中的性能。
- VFS 加锁的重新实现，显著改善文件系统的可扩展性。
- 显著缓解缓冲区溢出和内核空指针问题。
- 可扩展的内核安全框架（MAC Framework）现已正式可用。
- 完全更新的 USB 堆栈改善了性能和设备兼容性，增加了 USB target 模式。

FreeBSD 9.0 于 2012 年 1 月发布，该版本是第一个 9.x 的 FreeBSD 稳定分支。该版本具有以下特性：

- 采用了新的安装程序 bsdinstall
- ZFS 和 NFS 文件系统得到改进
- 升级了 ATA/SATA 驱动并支持 AHCI
- 采用 LLVM/Clang 代替 GCC
- 高效的 SSH（HPN-SSH）
- PowerPC 版支持索尼的 PS3

[2] ULE 3.0 在 8 核心的机器上以 sysbench MySQL 测试的结果，速度上比 Linux 2.6 快大约 10%（无论是使用 Google 的 tcmalloc 或是 glibc+cfs）

目前,FreeBSD 有许多非凡的特性。其中一些包括:

- 抢占式多任务与动态优先级调整确保在应用程序和用户之间平滑公正的分享计算机资源,即使工作在最大的负载之下。
- 多用户设备使得许多用户能够同时使用同一 FreeBSD 系统做各种事情。比如,像打印机和磁带驱动器这样的系统外设,可以完全地在系统或者网络上的所有用户之间共享,可以对用户或者用户组进行个别的资源限制,以保护临界系统资源不被滥用。
- 符合业界标准的强大 TCP/IP 网络支持,例如 SCTP、DHCP、NFS、NIS、PPP,SLIP,IPsec 以及 IPv6。这意味着 FreeBSD 主机可以很容易地和其他系统互联,也可以作为企业的服务器,比如 NFS(远程文件访问) 以及 email 服务,或接入 Internet 并提供 WWW,FTP,路由和防火墙 (安全) 服务。
- 内存保护确保应用程序 (或者用户) 不会相互干扰。一个应用程序崩溃不会以任何方式影响其他程序。
- FreeBSD 是一个 32 位操作系统 (在 Itanium®,AMD64,和 UltraSPARC® 上是 64 位),并且从开始就是如此设计的。
- 业界标准的 X Window 系统[3] (X11R7) 为便宜的常见 VGA 显示卡和监视器提供了一个图形化的用户界面 (GUI),并且完全开放代码。
- 和许多 Linux、SCO、SVR4、BSDI 和 NetBSD 程序的二进制代码兼容性
- 数以千计的 ready-to-run 应用程序可以从 FreeBSD ports 和 packages 套件中找到。
- 可以在 Internet 上找到成千上万其它 easy-to-port 的应用程序。FreeBSD 和大多数流行的商业 UNIX® 代码级兼容,因此大多数应用程序不需要或者只要很少的改动就可以编译。
- 页式请求虚拟内存和“集成的 VM/buffer 缓存”设计有效地满足了应用程序巨大的内存需求并依然保持其他用户的交互式响应。
- SMP 提供对多处理器的支持。
- 内建了完整的 C、C++、Fortran 开发工具。许多附加的用于高级研究和开发的程序语言,也可以在通过 ports 和 packages 套件获得。
- 完整的系统源代码意味着用户对生产环境的最大程度的控制。

FreeBSD 在 BSD 许可证下发布,允许任何人在保留版权和许可协议信息的前提下随意使用和发行。BSD 许可协议并不限制将 FreeBSD 的源代码在另一个协议下发行,因此任何团体都可以自由地将 FreeBSD 代码融入它们的产品之中去。

FreeBSD 包含了 GNU 通用公共许可证、GNU 宽通用公共许可证、ISC、CDDL 和 Beerware 许可证的代码,也有使用三条款和四条款的 BSD 许可证的代码。另外有些驱动程序也包涵了 binary blob,像是 Atheros 公司的硬件抽象层。这使得所有人都可以自由地使用还有再散布 FreeBSD。

不过,FreeBSD 的核心和新开发的代码大多都使用两条款的 BSD 许可证发布,许多使用 GPL 的代码都必须经过净室工程,以其他授权方式重写,这主要是避免整个核心受到 GPL 影响。

在过去的几年中 FreeBSD 的中央源代码树是由 CVS (并行版本控制系统) 来维护的,CVS 是一个与 FreeBSD 捆绑的可自由获得的源代码控制工具。自 2008 年六月起,这个项目开始转为使用 SVN (Subversion)。这次转换被认为是非常必要的,因为 CVS 的对于快速扩展源代码树和历史记录的限制越趋明显。现在主源码库使用 SVN,客户端的工具像 CVSup 和 csup 这些依赖于旧的 CVS 基础结构依然可以使用——因为对于 SVN 源码库的修改会被导回进 CVS。目前只有中央原代码树是由 SVN 控制的。文档,万维网和 Ports 库还仍旧使用着 CVS。The primary repository resides on a machine in Santa Clara CA, USA 主 CVS 代码库放置在美国加利福尼亚州圣克拉拉的一台机器上,它被复制到全世界的大量镜像站上。包含 –CURRENT 和 –STABLE 的 SVN 树也同样能非常容易的复制到用户的机器上。

committer 是那些对 CVS 树有写权限的人,他们被授权修改 FreeBSD 的源代码 (术语“committer”来自于 cvs 的 commit 命令,这个命令用来把新的修改提交给 CVS 代码库)。提交修正的最好方法是使用 `send-pr` 命令。

[3]FreeBSD 是廉价 X 终端的一种绝佳解决方案,可以选择使用免费的 X11 服务器。与 X 终端不同,如果需要的话 FreeBSD 能够在本地直接运行程序,因而减少了中央服务器的负担。

5.3 OpenBSD

OpenBSD 是一个从 BSD 派生出的类 Unix 操作系统，是在 1995 年由项目发起人西奥·德·若特从 NetBSD 分支而出。OpenBSD 以对开放源代码的坚持、高质量的文件、坚定的软件授权条款和专注于系统安全及代码质量而闻名。OpenBSD 以河豚作为项目吉祥物。

OpenBSD 包含了一些在其他操作系统缺少或是列为选择性的安全特色，此外 OpenBSD 非常重视代码审阅，至今开发者仍然保有这样的传统，此外 OpenBSD 对软件授权条款相当坚持，所有对核心的修改都必须符合 BSD 许可证的条款。

目前 OpenBSD 可以在 17 种不同的硬件环境或平台下运作包含了 DEC Alpha、Intel i386、Hewlett-Packard PA-RISC，AMD AMD64 和 Motorola 68000 处理器、Apple's PowerPC、Sun 的 SPARC 架构、VAX 和 Sharp Zaurus。

当创立 OpenBSD 的时候，西奥·德·若特就决定任何人都可以在任何时间取得源代码，在 Chuck Cranor 的协助下他创建了一个公开且匿名的 CVS 服务器，是第一个以开放式 CVS 作为开发方式的软件。因为在当时 CVS 的应用上大多只让少数的开发者有访问权，外部的开发者没有办法知道目前的工作进度，贡献的修正档也常常是已经完成过的修正。这种开发方式让 OpenBSD 成为开放源代码的代表软件。

1994 年 12 月，NetBSD 的共同发起人西奥·德·若特被要求辞去 NetBSD 的开发工作，而他访问 NetBSD 代码的权利也被取消。他辞职的实际原因不明，虽然他声称是因为和 NetBSD 的开发团队发生冲突而辞去开发工作。许多人认为他的离开是因为个性上难以相处，但也有许多人认为他是个直率的人，而离开的原因是因为有些人不认同他极度注重操作系统安全的理念。

1995 年 10 月，西奥·德·若特从 NetBSD 1.0 派生出了 OpenBSD 计划，在 1996 年 7 月发布了最初的发布版 OpenBSD 1.2，同年 10 月发布了 OpenBSD 2.0。之后每隔 6 个月 OpenBSD 便会发布一个新版本，每个发布版本会维护 1 年。

2007 年 7 月 25 日，OpenBSD 决定成立一个 OpenBSD 基金会。这个非营利性质的基金会将提供 OpenBSD 用户或是组织对 OpenBSD 法律上的支持服务，组织的地点设在加拿大。

5.4 NetBSD

NetBSD 是一份免费，安全的具有高度可定制性的类 Unix 操作系统，适于多种平台，从 64 位 AMD Athlon 服务器和桌面系统到手持设备和嵌入式设备。它设计简洁，代码规范，拥有众多先进特性，使得它在业界和学术界广受好评，用户可以通过完整的源代码获得支持。许多程序都可以很容易地通过 NetBSD Packages Collection 获得。

NetBSD 如同他的姊妹 FreeBSD 都是从加州柏克莱大学的 4.3BSD via the Networking/2 及 386BSD 为基础发展。因 386BSD 开发社区在操作系统开发的节奏与方向上的失败，该计划得以开始。NetBSD 的四位发起人，Chris Demetriou、西奥·德·若特、Adam Glass 以及 Charles Hannum 觉得开放的发展模式会有助于 NetBSD 计划的进行。他们的目的在于发展一套跨平台、高质量、以柏克莱软件包为基础的操作系统。

由于网络对于共同发展的重要性，西奥·德·若特建议这个项目的名称叫做 NetBSD，取得其他三位发起人的认同。NetBSD 原始代码的版本库创建于 1993 年 3 月 21 日，并于 1993 年四月发布了第一个版本，NetBSD 0.8。

同年 9 月，NetBSD 发布 0.9 版，包含了许多修正与功能的加强，惟仅限于台式机上运行。1994 年 10 月，NetBSD 发布 1.0 版，这个版本是 NetBSD 一个提供多平台的版本。目前 NetBSD 的最新版本为 2013 年 5 月 18 日所发布的 6.1 版。

作为该项目的口号（"Of course it runs NetBSD"）表明，NetBSD 已移植到了大量的 32 -和 64 位体系结构。从 VAX 小型机 Pocket PC 掌上电脑。从 2009 年起，NetBSD 支持 57 硬件平台（横跨 15 个不同的处理器架构）。NetBSD 的发行版比任何单一的 GNU / Linux 发行版支持更多的平台。这些平台的内核

和用户空间都是由中央统一管理的 CVS 源代码树。目前，不像其他的内核，如 μCLinux，NetBSD 内核在任何给定的目标架构需要 MMU 的存在。

5.5 Darwin

Darwin 是由苹果公司于 2000 年所发布的一个开放源代码操作系统。Darwin 是 Mac OS X 和 iOS 操作环境的操作系统部份。苹果公司于 2000 年把 Darwin 发布给开放源代码社区。

Darwin 是一种类 Unix 操作系统，集成数种的技术，包含开放源代码的 XNU 核心，一种以微核心为基础的核心架构来实现 Mach kernel。操作系统的服务和 userland 工具是以 4.4BSD（柏克莱软件包的 UNIX），特别是 FreeBSD 和 NetBSD 为基础。类似其他 Unix-like 操作系统，Darwin 也有对称多处理器的优点，高性能的网络设施和支持多种集成的文件系统。

集成 Mach 到 XNU 核心的好处是可携性，或者是在不同形式的系统使用软件的能力。举例来说，一个操作系统核心集成了 Mach 微核心，能够提供多种不同 CPU 架构的二进制格式到一个单一的文件（例如 x86 和 PowerPC），这是因为它使用了 Mach-O 的二进制格式。Mach 的缺点则是增加了操作系统核心 - 核心 - 的复杂度。在过去的微核心实现上，这种复杂度有时候会导致很难分离核心性能的问题。因此，采用 Mach 微核心会伴随风险，但它有潜在的好处是广泛的可携性。以 Darwin 可携性的具体例子来说，在 2005 年 6 月，苹果计算机宣布它会于 2006 年在 Mac 计算机上开始采用 Intel 处理器。

Darwin 的开发者在 2000 年决定采用一个吉祥物，选择了鸭嘴兽 Hexley 而不是它的竞争对手，像是一只 AquaDarwin fish、Clarus、和一只海怪。苹果计算机也没有把 Hexley 认可为 Darwin 的一个标志。

在 2002 年 4 月，在 ISC（Internet Software Consortium，互联网软件论坛）上，Apple 成立 OpenDarwin.org，是一个协助合作 Darwin 发展的社区。OpenDarwin 创建它自己发布的 Darwin 操作系统。值得注意的是 OpenDarwin 子计划中包含了 DarwinPorts，目标是组合下一世代的 port 集合给 Darwin 使用（对于长期而言，也给其他的 BSD 所派生的操作系统）。

2003 年 7 月，苹果在 APSL 的 2.0 版本下发布了 Darwin，是由自由软件基金会（FSF）批准为自由软件的许可证。

OS X（前称 Mac OS X）是苹果公司为麦金塔计算机开发的专属操作系统的最新版本。Mac OS X 于 1998 年首次推出，并从 2002 年起随麦金塔计算机发售。它是一套 Unix 基础的操作系统，包含两个主要的部份：核心名就是 Darwin，是以 FreeBSD 源代码和 Mach 微核心为基础，由苹果公司和独立开发者社区协力开发；及一个由苹果计算机开发，名为 Aqua 之专有版权的图形用户界面。

OS X Server 亦同时于 2001 年发售，架构上来说与工作站（客户端）版本相同，只有在包含的工作组管理和管理软件工具上有所差异，提供对于关键网络服务的简化访问，像是邮件传输服务器，Samba 软件，LDAP 目录服务器，以及名称服务器（DNS）。同时它也有不同的授权型态。

简单来说，OS X 是 Mac OS“版本 10”的分支，然而它与早期发布的 Mac OS 相比，在 Mac OS 的历史上是倾向独立发展的。它以 Mach 内核为基础，加入 UNIX 的 BSD 实现，再集成到 NeXTSTEP[4] 当中。

同时，苹果计算机企图创造一个独家拥有的” 新世代” 操作系统，但只有少部份成功。最后 NeXT 的操作系统（在当时称作 OPENSTEP）被选为苹果下个操作系统的基础形式，然后苹果计算机将 NeXT 全部买下来，并重新聘雇乔布斯。

乔布斯重回苹果的领导层次结构后，带领着苹果公司把原本倾向便利程序员的 OPENSTEP，转换到苹果计算机主要销售的家用市场，以及受到专业人士欢迎的 Rhapsody 系统上。经历过打击 Mac OS 独立开发者忠诚度的失算策略、Mac OS 9 转换到新系统，减轻转变之后，Rhapsody 演化为 Mac OS X。

Mac OS X 是与先前麦金塔操作系统彻底地分离开来，它的底层代码完全地与先前版本不同，这个新的核心名为 Darwin，是一个开放源代码、符合 POSIX 标准的操作系统，伴随着标准的 Unix 命令行与其强大的应用工具。尽管最重要的架构改变是在表面之下，但是 Aqua GUI 是最突出和引人注目的特色。柔软边缘的使用，半透明颜色和细条纹（与第一台 iMac 的硬件相似）把更多的颜色和材质带入到桌面上的视窗和控件，比 OS9 所提供的” 白金” 外观更多，引发了用户间大量的争论。很多旧的麦金塔

[4]NeXTSTEP 为当时史蒂夫·乔布斯（Steve Jobs）于 1985 年被迫离开苹果后，到 NeXT 公司所发展的。

用户把这个接口描述得像是玩具一般,和缺乏专业的优美,而其他的人则为苹果革命的新 GUI 创新为所欢呼。这种外观非常立即地可以辨认出来,即使在第一个 Mac OS X 版本推出之前,第三方的开发者开始针对可以换外表的程序像是 Winamp 制作类似 Aqua 接口的外表。苹果计算机以法律行动,威胁那些声称是由他们有版权的设计下,所制造或散布且提供这种接口软件的人。

Mac OS X 包含了自家的软件开发程序,其重大的特色是名为 Xcode 的集成开发环境。Xcode 是一个能与数种编译器沟通的接口,包括 C、C++、Objective-C、以及 Java。可以编译出目前 Mac OS X 所运行的两种硬件平台之可执行文件,可以指定编译成 PowerPC 平台专用,x86 平台专用,或是跨越两种平台的通用二进制。

Mac OS X 通过提供一种称为 Classic 的模拟环境,保留了与较旧的 Mac OS 应用程序的兼容性,允许用户在 Mac OS X 中把 Mac OS 9 当作一个程序进程来运行,使大部分旧的应用程序就像在旧的操作系统下运行一样。另外,给 Mac OS 9 和 Mac OS X 的 Carbon API 可以创造出允许在两种系统运行的代码。OpenStep 的 API 也依然可以使用,但是苹果现在把它称为 Cocoa 技术。(这个遗留下来的传统可以在 Cocoa API 中看到,大部分的类型名称都是以 NeXTSTEP 的缩写"NS" 开头。)给开发者的第四个选项是可以在 Mac OS X 当作" 第一等公民" 一样的 Java 平台上写应用程序—事实上这就是说 Java 应用程序尽可能的与操作系统合适地搭配而仍然能够" 跨平台(cross-platform)",以及他的 GUI,是以 Swing 撰写的,看起来几乎完全地与天生的 Cocoa 接口类似。

只要他们能够在这个平台上被编译,Mac OS X 可以运行很多 BSD 或 Linux 软件包。编译过的代码通常是以 Mac OS X 封装的方式来发布,但有些可能需要命令行的组态设置或是编译。像是 Fink 和 DarwinPorts 这样的项目,提供很多标准包之预先编译或是预先格式好的封装。在 10.3 版开始,Mac OS X 已经包含 Apple X11,这是给 Unix 应用程序的 X11 图形接口的公司版本,当作是在安装阶段的选择性组件。苹果是以 XFree86 4.3 和 X11R6.6 为基础实现的,搭配一个模仿 Mac OS X 外观的窗口管理器,与 Mac OS X 有更密切的集成,延展扩充到使用天生的 Quartz 显像系统和加速 OpenGL。早期的 Mac OS X 版本可使用 XDarwin 来运行 X11 应用程序。

第 6 章

Linux

In 1991, while attending the University of Helsinki, Torvalds became curious about operating systems and frustrated by the licensing of MINIX, which limited it to educational use only. He began to work on his own operating system which eventually became the Linux kernel.

Torvalds began the development of the Linux kernel on MINIX, and applications written for MINIX were also used on Linux. Later, Linux matured and further Linux kernel development took place on Linux systems. GNU applications also replaced all MINIX components, because it was advantageous to use the freely available code from the GNU Project with the fledgling operating system; code licensed under the GNU GPL can be reused in other projects as long as they also are released under the same or a compatible license. Torvalds initiated a switch from his original license, which prohibited commercial redistribution, to the GNU GPL.

Linux[?] is a Unix-like and POSIX-compliant computer operating system assembled under the model of free and open source software development and distribution. The main form of distribution is through Linux distributions. The defining component of Linux is the Linux kernel, an operating system kernel first released on 5 October 1991 by Linus Torvalds. Because it considers Linux to be a variant of the GNU operating system, initiated in 1983 by Richard Stallman, the Free Software Foundation prefers the name GNU/Linux when referring to the operating system as a whole; see GNU/Linux naming controversy for more details.

Linux was originally developed as a free operating system for Intel x86-based personal computers. It has since been ported to more computer hardware platforms than any other operating system. It is a leading operating system on servers and other big iron systems such as mainframe computers and supercomputers: as of June 2013, more than 95% of the world's 500 fastest supercomputers run some variant of Linux, including all the 44 fastest. Linux also runs on embedded systems (devices where the operating system is typically built into the firmware and highly tailored to the system) such as mobile phones, tablet computers, network routers, building automation controls, televisions and video game consoles; the Android system in wide use on mobile devices is built on the Linux kernel.

The development of Linux is one of the most prominent examples of free and open source software collaboration: the underlying source code may be used, modified, and distributed —commercially or non-commercially — by anyone under licenses such as the GNU General Public License. Typically, Linux is packaged in a format known as a Linux distribution for desktop and server use. Some popular mainstream Linux distributions include Debian (and its derivatives such as Ubuntu and Linux Mint), Fedora (and its derivatives such as the commercial Red Hat Enterprise Linux and its open equivalent CentOS), Mandriva/Mageia, openSUSE (and its commercial derivative SUSE Linux Enterprise Server), and Arch Linux. Linux distributions include the Linux kernel, supporting utilities and libraries and usually a large amount of application software to fulfill the distribution's intended use.

Developers worked to integrate GNU components with the Linux kernel, making a fully functional and free operating system.

A distribution oriented toward desktop use will typically include the windowing systems X11 and Wayland and an accompanying desktop environment such as GNOME or the KDE Software Compilation. Some such distributions may include a less resource intensive desktop such as LXDE or Xfce for use on older or less powerful computers. A distribution intended to run as a server may omit all graphical environments from the standard install and instead include other software to set up and operate a LAMP or a LYCE solution stack. Because Linux is freely redistributable, anyone may create a distribution for any intended use.

Linux 是一种自由和开放源代码的类 UNIX 操作系统，该操作系统的内核由林纳斯·托瓦兹在 1991 年 10 月 5 日首次发布。

严格来讲，术语 Linux 只表示操作系统内核本身，但通常采用 Linux 内核来表达该意思。Linux 则常用来指基于 Linux 内核的完整操作系统，包括 GUI 组件和许多其他实用工具。由于这些支持用户空间的系统工具和库主要由理查德·斯托曼于 1983 年发起的 GNU 计划提供，自由软件基金会提议将该组合系统命名为“GNU/Linux”。

Linux 最初是作为支持英特尔 x86 架构的个人电脑的一个自由操作系统。目前 Linux 已经被移植到更多的计算机硬件平台，远远超出其他任何操作系统。Linux 是一个领先的操作系统，可以运行在服务器和其他大型平台之上，如大型主机和超级计算机。世界上 500 个最快的超级计算机 90% 以上运行 Linux 发行版或变种，包括最快的前 10 名超级电脑运行的都是基于 Linux 内核的操作系统。Linux 也广泛应用在嵌入式系统上，如手机，平板电脑，路由器，电视和电子游戏机等。在移动设备上广泛使用的 Android 操作系统就是创建在 Linux 内核之上。

Linux 也是自由软件和开放源代码软件发展中最著名的例子。只要遵循 GNU 通用公共许可证，任何个人和机构都可以自由地使用 Linux 的所有底层源代码，也可以自由地修改和再发布。通常情况下，Linux 被打包成供个人计算机和服务器使用的 Linux 发行版，一些流行的主流 Linux 发布版，包括 Debian（及其派生版本 Ubuntu，Linux Mint），Fedora（及其相关版本 Red Hat Enterprise Linux，CentOS）和 openSUSE 等。Linux 发行版包含 Linux 内核和支撑内核的实用程序和库，通常还带有大量可以满足各类需求的应用程序。个人计算机使用的 Linux 发行版通常包 X Window 和一个相应的桌面环境，如 GNOME 或 KDE。桌面 Linux 操作系统常用的应用程序，包括 Firefox 网页浏览器，LibreOffice 办公软件，GIMP 图像处理工具等。由于 Linux 是自由软件，任何人都可以创建一个符合自己需求的 Linux 发行版。

绝大多数 Linux 操作系统使用了大量的 GNU 软件。正因为如此，GNU 计划的开创者理查德·马修·斯托曼提议将 Linux 操作系统改名为 GNU/Linux，但多数人仍然习惯性地使用“Linux”。

大多数 Linux 系统还包括了像提供 GUI 界面的 X Window 之类的程序。除了一部分专家之外，大多数人都是直接使用 Linux 发布版，而不是自己选择每一样组件或自行设置。

最初，林纳斯·托瓦兹是在 MINIX 上开发 Linux 内核，为 MINIX 写的软件也可以在 Linux 内核上使用，直到后来 Linux 成熟之后就可以在自己上面开发自己了。使用 GNU 软件代替 MINIX 的软件，因为使用从 GNU 系统来的源代码可以自由使用，这对新操作系统是有益的。使用 GNU GPL 协议的源代码可以被其他项目所使用，只要这些项目使用同样的协议发布。为了让 Linux 可以在商业上使用，林纳斯·托瓦兹决定改变他原来的协议（这个协议会限制商业使用），使用 GNU GPL 协议来代替。开发者致力于融合 GNU 元素到 Linux 中，做出一个有完整功能的、自由的操作系统。

事实上，Linus Torvalds 对于个人计算机的 CPU 其实并不满意，因为他之前接触的计算机都是工作站型的计算机，这类计算机的 CPU 特色就是可以进行“多任务处理”的能力。

什么是多任务呢？理论上，一个 CPU 在一个时间内仅能进行一项工作，那如果有两个工作同时出现到系统中呢？

举例来说，可以在现今的计算机中同时开启两个以上的办公软件，例如电子制表与文字处理软件。这个同时开启的动作代表着这两个工作同时要交给 CPU 来处理。

CPU 一个时间点内仅能处理一个工作，那怎么办？

这个时候如果具有多任务的 CPU 就会自动在不同的工作间切换，也就是先运行 10% 的电子制表，再转到文字处理运行 10% ，再回去电子制表 … 一直到将两个工作结束为止（不一定同时结束，如果某个工作先结束了，CPU 就会全速去运行剩下的那个工作了）。

以 1GHz 的 CPU 为例，该 CPU 每一秒可以进行 10^9 次工作。假设 CPU 对每个程序都只进行 1000 次运行周期，然后就得切换到下个程序，那么 CPU 在 1 秒钟内就能够切换 10^6 次，这样在高速的 CPU 下，运行的程序在用户看来几乎是同步在运行的。

早期 Intel 的 x86 架构计算机不是很受重视的原因，就是因为 x86 的芯片对于多任务的处理不佳，CPU 在不同的工作之间切换不是很顺畅。但是这个情况在 386 计算机推出后有很大的改善。Linus

为什么有的时候同时打开两个文件(假设为 A,B 文件)所花的时间,要比打开完 A 再去打开 B 文件的时间还要多?

因为如果同时开启的话,CPU 就必须要在两个工作之间不停的切换,而切换的动作还是会耗去一些 CPU 时间的,所以同时运行两个

Torvalds 在得知新的 386 芯片的相关信息后,他认为以性价比的观点来看,Intel 的 386 便宜而且性能上也可以将就,所以他就贷款去买了一台 Intel 的 386。

为了彻底发挥 386 的效率,于是 Linus Torvalds 花了不少时间在测试 386 机器上,他的重要测试就是在测试 386 的多任务效率上。

首先,他写了三个小程序,一个程序会持续输出 A,另一个会持续输出 B,最后一个会将两个程序进行切换。他将三个程序同时执行,结果他看到显示器上很顺利的一直出现 ABABABAB···,他知道,他成功了。

6.1 Linux 0.02

探索完了 386 的硬件相关信息,并且也安装了类似 UNIX 的 Minix 操作系统,同时还取得 Minix 的源代码,接下来 Linus Torvalds 干嘛去了?因为 Minix 的开发控制在谭宁邦教授手上,他希望 Minix 能以教育的立场去开发,所以对于 Minix 的开发并不是十分的热衷,但是用户对于 Minix 的功能需求又很强烈,例如一些接口与周边的驱动程序与新的协议等。在无法快速的得到解决后,Linus Torvalds 就想,那我自己写一个更适合自己用的 Minix,于是他就开始进行核心程序的编写了。

对于 Linus Torvalds 来说 GNU 真的是一个不可多得的好帮手,因为他用来编写属于自己小核心的工具就是 GNU 的 bash 操作界面与 gcc 编译器等自由软件。他以 GNU 的软件针对 386 并参考 Minix 的设计理念(注意,仅是程序设计理念,并没有使用 Minix 的源代码)来写这个小核心,而这个小核心竟然可以在 386 上面顺利的运行起来,还可以读取 Minix 的文件系统。

不过还不够,他希望这个小核心可以获得大家的一些修改建议,于是他将这个核心放置在网络上提供大家下载,同时在 BBS 上面贴了一则消息:

```
Hello everybody out there using minix-
I'm doing a (free) operation system (just a hobby,
won't be big and professional like gnu) for 386(486) AT clones.
I've currently ported bash (1.08) and gcc (1.40),
and things seem to work. This implies that i'll get
something practical within a few months, and I'd like to know
what features most people want. Any suggestions are welcome,
but I won't promise I'll implement them :-)
```

他说,他完成了一个好玩的小核心操作系统,这个核心是运行在 386 机器上的,同时他真的仅是好玩,并不是想要做一个跟 GNU 一样大的计划。这则新闻引起很多人的注意,他们也去 Linus Torvalds 提供的网站上下载了这个核心来安装。有趣的是,因为 Linus Torvalds 放置核心的那个 FTP 网站的目录为:Linux,从此大家便称这个核心为 Linux 了[1]。

同时,为了让自己的 Linux 能够相容于 UNIX 系统,于是 Linus Torvalds 开始将一些能够在 UNIX 上面运行的软件拿来在 Linux 上面运行。不过他发现是有很多的软件无法在 Linux 这个核心上运行。这个时候他有两种作法,一种是修改软件,让该软件可以在 Linux 上运行,另一种则是修改 Linux,让 Linux 符合软件能够运行的规范。

由于 Linus Torvalds 希望 Linux 能够兼容于 UNIX,于是他选择了第二个作法“修改 Linux”。为了让所有的软件都可以在 Linux 上执行,Linus Torvalds 开始参考标准的 POSIX[2]规范。这个正确的决定让

[1]注意,此时的 Linux 就是那个 kernel。另外 Linus Torvalds 所放到该目录下的第一个核心版本为 0.02。

[2]POSIX 是可携式操作系统接口(Portable Operating System Interface)的缩写,重点在于规范内核与应用程序之间的接口,这是由美国电气与电子工程师学会(IEEE)发布的标准。

Linux 在起步的时候体质就比别人优良，因为 POSIX 标准主要是针对 UNIX 与一些软件运行时候的标准规范，只要依据这些标准规范来设计的核心与软件，理论上就可以搭配在一起执行了。而 Linux 的开发就是依据这个 POSIX 的标准规范，UNIX 上面的软件也是遵循这个规范来设计的，如此一来，让 Linux 很容易就与 UNIX 相容并共享互有的软件了。同时因为 Linux 直接放置在网络上供大家下载，所以在流通的速度上相当的快，导致 Linux 的使用率大增，这些都是造成 Linux 大受欢迎的几个重要因素。

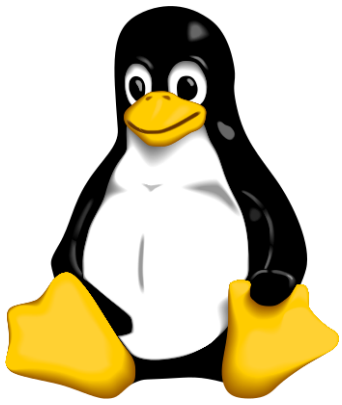
Linux 的第一个版本在 1991 年 9 月被大学 FTP server 管理员 Ari Lemmke 发布在 Internet 上，最初 Torvalds 称这个内核的名称为"Freax"，意思是自由("free")和奇异("freak")的结合字，并且附上了"X" 这个常用的字母，以配合所谓的类 Unix 的系统。但是 FTP 服务器管理员嫌原来的命名"Freax"的名称不好听，把内核的称呼改成"Linux"，当时仅有 10000 行程序码，仍必须运行于 Minix 操作系统之上，并且必须使用硬盘开机；随后在 10 月份第二个版本(0.02 版)就发布了，同时这位芬兰赫尔辛基的大学生在 comp.os.minix 上发布一则信息

```
Hello everybody out there using minix- I'm doing a (free) operation system (just a hobby,
```

1994 年 3 月，Linux1.0 版正式发布，Marc Ewing 成立了 Red Hat 软件公司，成为最著名的 Linux 经销商之一。

早期 Linux 的开机管理程序(boot loader)是使用 LILO(Linux Loader)，早期的 LILO 存在着一些难以容忍的缺陷，例如无法识别 1024 柱面以后的硬盘空间，后来新增 GRUB(GRand Unified Bootloader)克服了这些缺点，具有'动态搜索内核文件'的功能，可以让您在开机的时候，可以自行编辑您的开机设置系统文件，通过 ext2 或 ext3 文件系统中加载 Linux Kernel(GRUB 通过不同的文件系统驱动可以识别几乎所有 Linux 支持的文件系统，因此可以使用很多文件系统来格式化内核文件所在的扇区，并不局限于 ext 文件系统)。

Linux 的标志和吉祥物是一只名字叫做 Tux 的企鹅，标志的由来是因为 Linus 在澳洲时曾被一只动物园里的企鹅咬了一口，便选择了企鹅作为 Linux 的标志。更容易被接受的说法是：企鹅代表南极，而南极又是全世界所共有的一块陆地。这也就代表 Linux 是所有人的 Linux。



[!ht]
图 6.1: Tux - the penguin, mascot of Linux

6.2 XFree86

1988 年 — 图形界面 XFree86 计划

由于图形用户界面(Graphical User Interface，GUI)的需求日益增加，在 1984 年由 MIT 与其他厂商首次发布了 X Window System，并在 1988 年成立了非营利性质的 XFree86 组织。XFree86 其实是 X Window

System + Free + x86 的整合名称，而这个 XFree86 的 GUI 界面更是在 Linux 的核心 1.0 版于 1994 年发布时整合于 Linux 操作系统中。

X Window System 只是 Linux 上的一套软件，而不是核心。即使 X Window 出现问题，对 Linux 也不会造成直接的影响。当管理图形界面输出时，XFree86 管理着几乎所有与显示相关的控制，例如显卡、屏幕、键盘、鼠标等。或者，可以称 XFree86 为 X Window System 的服务器，简称为 X Server。

图形用户界面是利用 X Server 提供的相关显示硬件的功能来完成图形化显示的窗口管理器（Window Manager，WM）的功能，也就是说，Window Manager 是挂载在 X Server 上运行的一套显示图形用户界面的软件，包括 GNOME、KDE 等。

6.3 GNU/Linux

The Free Software Foundation views Linux distributions that use GNU software as GNU variants and they ask that such operating systems be referred to as GNU/Linux or a Linux-based GNU system. The media and common usage, however, refers to this family of operating systems simply as Linux, as do many large Linux distributions (e.g. SUSE Linux and Mandriva Linux). Some distributions, notably Debian, use GNU/Linux. The naming issue remains controversial.

Linux 虽然是 Linus Torvalds 发明的，而且内容还绝不会涉及专利软件的版权问题。不过如果单靠 Linus Torvalds 自己一个人的话，那么 Linux 要继续发展壮大实在是很困难。

因为一个人的力量是很有限的，好在 Linus Torvalds 选择 Linux 的开发方式相当的务实。首先他将发布的 Linux 核心放置在 FTP 上面，并请告知大家新的版本信息，等到用户下载了这个核心并且安装之后，如果发生问题，或者是由于特殊需求亟需某些硬件的驱动程序，那么这些用户就会主动汇报给 Linus Torvalds。在 Linus Torvalds 能够解决的问题范围内，他都能很快速的进行 Linux 核心的更新与除错。

不过，Linus Torvalds 总是有些硬件无法取得，那么他当然无法帮助进行驱动程序的编写与相关软件的改进，这个时候就会有些志愿者出来说：“这个硬件我有，我来帮忙写相关的驱动程序。”因为 Linux 的核心是 Open Source 的，志愿者们很容易就能够跟随 Linux 的设计架构，并且写出相容的驱动程序或者软件。

志愿者们写完的驱动程序与软件 Linus Torvalds 是如何看待的呢？首先，他将该驱动程序/软件加入核心中并且加以测试。只要测试可以运行，并且没有什么主要的大问题，那么他就会很乐意的将志愿者们写的程序代码加入核心中。总之，Linus Torvalds 是个很务实的人，对于 Linux 核心所欠缺的项目，他总是“先求有且能运行，再求进一步改进”的心态，这让 Linux 用户与志愿者得到相当大的鼓励。

另外，为顺应这种程序代码的加入，于是 Linux 便逐渐开发成具有模块（module）的功能，也就是将某些功能独立于核心外，在需要的时候才载入到核心中，这样如果有新的硬件驱动程序或者其他协议的程序代码进来时就可以模块化，而且模块化之后原先的核心不需要变动，大大的增加了 Linux 核心的可维护性。

后来因为 Linux 核心加入了太多的功能，光靠 Linus Torvalds 一个人进行核心的实际测试并加入核心原始程序实在太费力，结果就有很多的朋友出来帮忙这个前置操作，例如考克斯（Alan Cox）与崔迪（Stephen Tweedie）等，这些重要的副手会先将来自志愿者们的修补程序或者新功能的程序代码进行测试，并且将结果上传给 Linus Torvalds 看，让 Linus Torvalds 作最后核心加入的源代码的选择与整合。这个分层负责的结果让 Linux 的开发更加的容易。

特别值得注意的是，这些 Linus Torvalds 的 Linux 开发副手以及自愿传送修补程序的骇客志愿者，其实都没有见过面，而且彼此在地球的各个角落，大家群策群力的共同开发出现今的 Linux，我们称这群人为虚拟团队，而为了虚拟团队数据的传输，他们成立了核心网站：http://www.kernel.org/，而这群素未谋面的虚拟团队们，在 1994 年终于完成了 Linux 的核心正式版—Version 1.0，这一版同时还加入了 X Window System 的支持，接着于 1996 年完成了 2.0 版，同时顺应商业版本的需求开始将核心版本以测试版及稳定版同时开发，次版本偶数为稳定版，奇数为开发中的测试版。例如 2.6 与 2.5 版为

相同的版本,不过 2.6 为稳定版,2.5 则为测试版。测试版含有较多的功能,不过稳定性还不确定,并且 Linus Torvalds 指明了企鹅为 Linux 的吉祥物。

可以使用如下的命令查看 Linux 的内核版本编号:

```
[root@linux ~]# uname -r
3.6.3-1
主版本号. 次版本号. 释出版本-修改版本
```

Linus Torvalds 将内核的开发趋势分为两股,并根据这两个内核的开发分别给予不同的内核编号,那就是:

1. 主、次版本号为奇数:开发中版本(development)

这种内核版本主要用在测试与开发新功能,通常这种版本仅有内核开发工程师会使用。如果有新增的内核程序代码,会加到这种版本中,等到测试通过后才加入下一版的稳定内核中。

2. 主、次版本号为偶数:稳定版本(stable)

等到内核功能开发成熟后会加到这类的版本中,主要用在家用计算机与企业版本中,重点在于提供给用户一个相对稳定的 Linux 作业环境平台。

3. 发布版本是在主、次版本架构不变的情况下,新增的功能累积到一定的程度后所发布的内核版本。
4. Linux 内核版本与 distribution 版本并不相同,Linux 版本指的是内核版本,不同的 Linux 的 distribution 使用的都是 Linux 的内核,但是版本号并不相同,而且不同的 distribution 所选用的软件以及它们自己开发的工具并不相同,多少还是有一定差异。

由于最初 Linux 是 Linus Torvalds 针对 386 编写的,跟 386 硬件的相关性很强,所以早期的 Linux 确实是不具有移植性的。不过大家知道 Open Source 的好处就是可以修改程序代码去适合操作的环境,因此在 1994 年以后 Linux 便被开发到很多的硬件上面去了,目前除了 x86 之外,IBM、HP、Sun 等等公司的硬件也都被 Linux 所支持。

6.4 Copyright, trademark, and naming

Linux kernel is licensed under the GNU General Public License (GPL), version 2. The GPL requires that anyone who distributes a software product based on GPL-licensed source code, must make the originating source code (and any modifications) available to the recipient under the same terms. Other key components of a typical Linux distribution are also mainly licensed under the GPL, but they may use other licenses; many libraries use the GNU Lesser General Public License (LGPL), a more permissive variant of the GPL, and the X.org implementation of the X Window System uses the MIT License.

Torvalds states that the Linux kernel will not move from version 2 of the GPL to version 3. He specifically dislikes some provisions in the new license which prohibit the use of the software in digital rights management. It would also be impractical to obtain permission from all the copyright holders, who number in the thousands.

A 2001 study of Red Hat Linux 7.1 found that this distribution contained 30 million source lines of code. Using the Constructive Cost Model, the study estimated that this distribution required about eight thousand man-years of development time. According to the study, if all this software had been developed by conventional proprietary means, it would have cost about $1.46 billion (2013 US dollars) to develop in the United States. Most of the source code (71%) was written in the C programming language, but many other languages were used, including C++, Lisp, assembly language, Perl, Python, Fortran, and various shell scripting languages. Slightly over half of all lines of code were licensed under the GPL. The Linux kernel itself was 2.4 million lines of code, or 8% of the total.

In a later study, the same analysis was performed for Debian GNU/Linux version 4.0 (etch, which was released in 2007). This distribution contained close to 283 million source lines of code, and the study estimated that it would have required about seventy three thousand man-years and cost US$8.07 billion (in 2013 dollars) to develop by conventional means.

In the United States, the name Linux is a trademark registered to Linus Torvalds. Initially, nobody registered it, but on 15 August 1994, William R. Della Croce, Jr. filed for the trademark Linux, and then demanded royalties from Linux distributors. In 1996, Torvalds and some affected organizations sued him to have the trademark assigned to Torvalds, and in 1997 the case was settled. The licensing of the trademark has since been handled by the Linux Mark Institute. Torvalds has stated that he trademarked the name only to prevent someone else from using it. LMI originally charged a nominal sublicensing fee for use of the Linux name as part of trademarks, but later changed this in favor of offering a free, perpetual worldwide sublicense.

Linux 的注册商标是 Linus Torvalds 所有的。这是由于在 1996 年，一个名字叫做 William R. Della Croce 的律师开始向各个 Linux 发布商发信，声明他拥有 Linux 商标的所有权，并且要求各个发布商支付版税，这些发布商集体进行上诉，要求将该注册商标重新分配给 Linus Torvalds。Linus Torvalds 一再声明 Linux 是自由且免费的，他本人可以卖掉，但 Linux 绝不能卖。

根据托瓦兹的说法，Linux 的发音和“Minix”是押韵的。

“Li”中“i”的发音类似于“Minix”中“i”的发音，而“nux”中“u”的发音类似于英文单词“profess”中“o”的发音。依照国际音标应该是 ['linəks][□l□n□ks]。

“GNU/Linux”是 GNU 计划的支持者与开发者，特别是其创立者理查德·斯托曼对于以 Linux 内核为内核的操作系统的称呼。由于 Linux 使用了许多 GNU 程序，理查德·斯托曼认为应该将该操作系统称为“GNU/Linux”比较恰当。

一些人拒绝使用“GNU/Linux”作为操作系统名称的人认为 Linux 朗朗上口，短而好记，并且斯托曼直到 1990 年代中期 Linux 开始流行后才要求更名。

有部分 Linux 发行版，包括了 Debian，采用了“GNU/Linux”的称呼。但大多数商业 Linux 发行版依然将操作系统称为 Linux。有些人也认为“操作系统”一词指的应该只是系统的内核，其他程序都只能算是应用软件，这么一来，该操作系统的内核应叫 Linux，而 Linux 发行版是在 Linux 内核的基础上加入各种 GNU 工具。

第 7 章

Debian

Debian is an operating system composed of free software mostly carrying the GNU General Public License. The operating system is developed by an internet collaboration of volunteers aligned with The Debian Project.

Debian systems can use either the Linux kernel (known as the Debian GNU/Linux distribution), the FreeBSD kernel (known as the Debian GNU/kFreeBSD distribution) or, more recently, the GNU Hurd kernel (more precisely, the GNU Mach microkernel and its servers; known as the Debian GNU/Hurd distribution).

Debian GNU/Linux is one of the most popular Linux distributions for personal and Internet server machines. Debian is seen as a solid Linux, and as a consequence has been used as a base for other Linux distributions; DistroWatch lists 144 active Debian derivatives. Debian has been forked many times, but is not affiliated with its derivatives.

The vital role the Debian project plays in free software is demonstrated by its advancement of development and security patches relating to its strong participation in CVE compatibility efforts.

The Debian project released a new kernel as of Wheezy's release date: the Debian GNU/kFreeBSD kernel. Debian now supports two kernels, Linux and kFreeBSD, and offers other kernels as development works (GNU Hurd and NetBSD). This project's new kernel has recently come out of preview but still lacks the amount of software available as on Debian's Linux. The kernel is offered for Intel/AMD 32-bit and 64-bit architecture machines.[15] Wheezy is officially supported on ten machine architectures and also brought support for two new architectures: s390x and armhf.

Debian is still primarily known as a Linux distribution with access to online repositories hosting over 37,500 software packages.[16] Debian officially hosts free software on its repositories but also allows non-free software to be installed. Debian includes popular programs such as LibreOffice,[17] Iceweasel (a rebranding of Firefox), Evolution mail, CD/DVD writing programs, music and video players, image viewers and editors, and PDF viewers. The cost of developing all the packages included in Debian 5.0 lenny (323 million lines of code) using the COCOMO model, has been estimated to be about US$ 8 billion.[18] Ohloh estimates that the codebase (68 million lines of code) using the same model, would cost about US$ 1.2 billion to develop.

Debian offers 10 DVD and 69 CD images for download and installation, but using just the first iso image of any of its downloadable sets is sufficient; the installer can retrieve software not contained in the first iso image from online repositories. The Wheezy release offers to install a variety of default Desktops from its DVD boot menu (GNOME, KDE, Xfce, and LXDE) and allows visually-impaired people to use its installer. The new feature in Debian's latest installer of Wheezy for the visually-impaired supports a mode which is textual but performs audio output for each stage of installation. Debian offers different network installation methods for expert users. A minimal install of Debian is available via the 'netinstall' CD, whereby Debian is installed with just a base and later additional software can be downloaded from the internet.

Debian's new form of installation-from-USB has been supported since its sixth edition. Debian supports this capability inside its first iso-file of any of its install media sets (whether CD or DVD) and does not require the help of 'extraction tools' such as unetbootin. This new feature is called Hybrid iso in which an .iso file is dumped to USB. Debian is one of the few Linux distributions offering this feature with its install iso media, and other distributions are starting to adopt alike.

Other notable new features in Debian's latest release include: Multiarch, which allows 32-bit Linux software to

run on 64-bit operating system installs;[21] improved multimedia support, reducing reliance on third-party repositories; compiled packages with hardened security flags; AppArmor, which can protect a system against unknown vulnerabilities; and systemd, which shipped as a technology preview.

Debian's install-media is distributed via downloadable CD/DVD images from its mirrored sites,[22] BitTorrent, jigdo and from optional retailers.

Debian offers stable and testing CD images specifically built for GNOME (the default), KDE Plasma Workspaces, Xfce and LXDE.[24] Less common window managers such as Enlightenment, Openbox, Fluxbox, GNUstep, IceWM, Window Maker and others can also be installed.

It was previously suggested that the default desktop environment of version 7.0 "Wheezy" may be switched to Xfce, because GNOME 3 might not fit on the first CD of the set.[25] The Debian Installer team announced that the first CD includes GNOME thanks to their efforts to minimize the amount of disc space GNOME takes up.[26] [27] In November 2013 it was announced that the desktop environment of version 8.0 "Jessie" would change to Xfce. Demand for Xfce will be evaluated and the default environment may switch back to GNOME again just before Jessie is frozen.

MATE and Cinnamon are not available in the official Debian package repositories, but are available in third-party repositories.

7.1 Packages

Most software packages in the official Debian repositories are compiled for an abundance of available and older instruction sets.

Debian is known for its serious manifesto social contract and policies.[56] Debian's policies and team efforts focus on collaborative software development and testing processes and dedicates lengthy development time between unstable and stable release cycles. As a result of its strictly guarded policies, a new distribution release for Debian tends to occur every one to two years.[57] The strategy policies used by the Debian project for minimizing software bugs, albeit with longer release cycles, has allowed it to remain one of the most stable and secure Linux distributions.

Debian's official standard for administering packages on its system is the apt toolset. Though for many years apt-get has been the de facto tool for administering packages on Debian, suggestions point to aptitude as better for interactive use as aptitude supports better search on package metadata.[39]

dpkg is the storage information center of installed packages and provides no configuration for accessing online repositories. The dpkg database is located at /var/lib/dpkg/available and contains the list of "installed" software on the current system.

The dpkg command tool is used for the dpkg database without capability of accessing online repositories.[40] The command can work with local .deb package files as well as information from the dpkg database.

An APT tool allows administration of an installed Debian system for retrieving and resolving package dependencies from online repositories. APT tools share dependency information(/etc/apt configuration files) and downloaded cache .deb files(/var/cache/apt/archives). APT tools depend on verifying what is installed in the dpkg database in order to determine missing packages for requested installs.

- aptitude is a command tool and offers a TUI interface. This command's support is self-contained for full features of APT administration
- apt-get and apt-cache are command tools of the standard APT-class toolset apt package. The package is called "apt" and supplies the command "apt-cache" as well as "apt-get". "apt-get" installs and removes packages. "apt-cache" is used for searching packages and displaying package information.
- gdebi is an APT which can be used in command-line and on the GUI. gdebi combines the functionality of the dpkg tool and APT package resolving with online repositories. The local .deb file in the GUI environment's file manager can be associated to be opened with gdebi providing a graphical experience of installing packages

via double clicking. gdebi can also install a local .deb file via the command line alike the dpkg command, but with access to online repositories. If gdebi is requested to install a local .deb file that requires to install a dependency, it then searches the repositories as defined in the common APT configuration folder (/etc/apt) and performs to resolving and downloading missing packages.

There are various front-ends for APT, both graphical and command-line based. Graphical applications include Software Center, Synaptic and Apper.

7.2 Repositories

The Debian Project offers three distributions, each with different characteristics. The distributions include packages which comply with the Debian Free Software Guidelines (DFSG), which are included inside the main repositories.[42] Debian also officially supports the optional backports repository for the stable distribution.[43]

When in need of updated versions of software, it is possible to use Debian testing instead of stable as it usually contains more modern, though slightly less stable packages. Another alternative is to use Debian backports, which are ”recompiled packages from testing (mostly) and unstable (in a few cases only, e.g. security updates), so they will run without new libraries (wherever it is possible) on a stable Debian distribution”.[44]

Official repositories are the following:

- stable, currently aliased wheezy, is the current release that has stable and well-tested software. Stable is made by freezing testing for a few months where bugs are fixed to make the distribution as stable as possible; then the resulting system is released as stable. It is updated only if major security or usability fixes are incorporated. After Debian 6.0, new releases will be made every two years.[45] Stable's CDs and DVDs can be found in the Debian website.[42] backports: This repository provides more recent versions than stable for some software. It is mainly intended for users of stable who need a newer version of a particular package.
- testing, currently aliased jessie, contains software being tested for inclusion in the next major release. The packages included in this distribution have had some testing in unstable but they may not be completely fit for release yet. It contains more modern packages than stable but older than unstable. This distribution is updated continually until it enters the ”frozen” state. Security updates for testing distribution are provided by Debian testing security team. Testing's CDs and DVDs can be found on the Debian website.[42]
- unstable, permanently aliased sid, repository contains packages currently under development; it is updated continually. This repository is designed for Debian developers who participate in a project and need the latest libraries available, or for those who like to ”live on the edge”, so it will not be as stable as the other distributions. Debian does not support official sid installation CDs/DVDs because the repository is rapidly changing, although CD and DVD images of sid are built quarterly by aptosid. Additionally, the other distributions can be set to draw software from unstable the next time the system is upgraded.

Depreciating repositories in Debian:

- oldstable, presently aliased squeeze, is the prior stable release. It is supported until 1 year after a new stable is released. Debian recommends to update to the new stable once it has been released.[46]
- snapshot: The snapshot repositories provide older versions of other repositories. They may be used to install a specific older version of some software.

Other repository in Debian:

- experimental: is meant to be a temporary staging area of highly experimental software for developers. Even though it's documented as a 'distribution' branch from Debian's documentation, many dependency packages that fulfill experimental software are commonly resolved with the unstable branch. This branch of experimental software is not as documented as the other branches as it shouldn't even be referenced by advanced users.

The Debian Free Software Guidelines (DFSG) defines its distinctive meaning of the word ”free” as in ”free and

open source software" (FOSS), although it is not endorsed by the Free Software Foundation or GNU foundation; reason being Debian's servers includes and supports a proprietary repository and documentation that recommends non-free software.[48][49] In accordance with its guidelines, a relatively small number of packages are excluded from the distributions' main repositories and included inside the non-free and contrib repositories. These two repositories are not officially part of Debian GNU/Linux. The Debian project offers its distribution without non-free repositories but can be adopted manually after initial setup.

- non-free: repositories include packages which do not comply with the DFSG (this does not usually include legally questionable packages, like libdvdcss).[42]
- contrib: repositories include packages which do comply with the DFSG, but may fail other requirements. For instance, they may depend on packages which are in non-free or requires such for building them.

These repositories are not part of the Debian Project, they are maintained by third party organizations. They contain packages that are either more modern than the ones found in stable or include packages that are not included in the Debian Project for a variety of reasons such as: e.g. possible patent infringement, binary-only/no sources, or special licenses that are too restrictive. Their use requires precise configuration of the priority of the repositories to be merged; otherwise these packages may not integrate correctly into the system, and may cause problems upgrading or conflicts between packages from different sources. The Debian Project discourages the use of these repositories as they are not part of the project.

Debian（国际音标：/ˈdɛ.bi.ən/）是由 GPL 和其他自由软件许可协议授权的自由软件组成的操作系统，由 Debian 计划（Debian Project）组织维护。Debian 计划是一个独立的、分散的组织，由 3000 人志愿者组成，接受世界多个非盈利组织的资金支持，Software in the Public Interest 提供支持并持有商标作为保护机构。

Debian 以其坚守 Unix 和自由软件的精神，以及其给予用户的众多选择而闻名。现时 Debian 包括了超过 37,500 个软件包并支持 12 个计算机系统结构。

Debian 是一个大的系统组织框架，在这个框架下有多种不同操作系统核心的分支计划，主要为采用 Linux 核心的 Debian GNU/Linux 系统，其他还有采用 GNU Hurd 核心的 Debian GNU/Hurd 系统、采用 FreeBSD 核心的 Debian GNU/kFreeBSD 系统，以及采用 NetBSD 核心的 Debian GNU/NetBSD 系统。甚至还有应用 Debian 的系统架构和工具，采用 OpenSolaris 核心构建而成的 Nexenta OS 系统。在这些 Debian 系统中，以采用 Linux 核心的 Debian GNU/Linux 最为著名。众多的 Linux 发布版，例如 Ubuntu、Knoppix 和 Linspire 及 Xandros 等，都建基于 Debian GNU/Linux。

Debian 于 1993 年 8 月 16 日由美国普渡大学学生伊恩·默多克首次发表。伊恩·默多克最初把他的系统称为“Debian Linux Release”。[3] 在定义文件 Debian Manifesto 中，Ian Murdock 宣布将以开源的方式，本着 Linux 及 GNU 的精神发布一套 GNU/Linux 发布版。Debian 的名称是由他当时的女友（现在为其前妻 [4]）Debra 和 Ian Murdock 自己的名字合并而成的，所以 Debian 一词是根据这两个名字在美国英语的发音而读作/dɛbiən/。

Debian 计划最初发展缓慢，在 1994 年和 1995 年分别发布了 0.9x 版本；1.x 版本则在 1996 年发布。1996 年，布鲁斯·佩伦斯接替了伊恩·默多克成为了 Debian 计划的领导者。同年，一名开发者 Ean Schuessler 提议 Debian 应在其计划与用户之间创建一份社区契约。经过讨论，布鲁斯·佩伦斯发表了 Debian 社区契约及 Debian 自由软件指导方针，定义了开发 Debian 的基本承诺。

1998 年在建基于 GNU C 运行期库的 Debian 2.0 发布之前，布鲁斯·佩伦斯离开了 Debian 的开发工作。Debian 开始选出新的领导者，并发布了另外两个 2.x 版本，包含了更多接口和软件包。APT 和第一个非 Linux 接口—Debian GNU/Hurd 的开发也展开。第一个建基于 Debian 的 Linux 发布版 Corel Linux 和 Stormix 的 Storm Linux 在 1999 年开始开发。尽管未能成功开发，这两个发布版成为了建基于 Debian 的 Linux 发布版的先驱。

在 2000 年后半年，Debian 对数据库和发布的管理作出了重大的改变，它重组了收集软件的过程，并创造了“测试”（testing）版本作为较稳定的对下一个发布的演示。同年，Debian 的开发者开始举办名为 Debconf 的年会，为其开发者和技术用家提供讲座和工作坊。

正在开发中的软件会被上载到名为“不稳定”（unstable，代号 sid）和“实验性”（experimental）的计划分支上。上载至“不稳定”分支上的软件通常是由软件的原开发者发布的稳定版本，但包含了一些未经测试的 Debian 内部的修改（例如软件的打包）。而未达到“不稳定”分支要求的软件会被置于“实验性”分支。

一套软件在置于“不稳定”分支一段时间后（关乎软件修改的紧急性），该软件会自动被移至“测试”分支。但如果软件有严重错误被报告，或其所依存的软件未合乎“测试”分支的要求，该软件则不会被移至“测试”分支。

因为 Debian 官方发布的正式版本并不包含新的特色，一些桌面用家会选择安装“测试”甚至“不稳定”分支。但是这两个分支所进行的测试比稳定版本少些，可能较不稳定；而且这两个分支并没有定时的安全更新。[7] 更甚者，软件不当地升级至不稳定的版本可能严重影响其运用。

在“测试”分支中的软件三年没有回报一个 bug 后，“测试”分支会成为下一个稳定版本。

当然，人们真正需要的是应用软件，Debian 上的软件管理系统为 APT，亦有图形界面的 synaptic 和 aptitude 可供使用。为了方便用户使用，这些软件包都已经被编译包装为一种方便的格式，开发人员把它叫做 deb 包。

7.3 Branches

Debian 主要分三个版本：稳定版本（stable）、测试版本（testing）、不稳定版本（unstable）。目前的稳定版本为 Debian Wheezy，目前的测试版本为 Debian Jessie，不稳定版本永远为 Debian sid[1]。到目前为止所有开发代号均出自 Pixar 的电影玩具总动员。

Debian 以稳定性闻名，所以很多服务器都使用 Debian 作为其操作系统；而很多 Linux 的 LiveCD 亦以 Debian 为基础改写，最为著名的例子为 Knoppix。而在桌面领域，Debian 的一个修改版 Ubuntu Linux 就获得了很多 Linux 用户的支持。

对比 Ubuntu、Fedora 等 Linux 发布版，较少桌面用家会选择使用 Debian。主要原因是其基于较新功能的考量。包版本一般需要长时间的测试，甚至因为测试时间过长造成与最新的软件包有些落差，以稳定的系统要求为优先。

把 Debian 移植至其他内核的工作正在进行，最主要的就是 Hurd。Hurd 是一组在微内核（例如 Mach）上运行的服务器，它们可以提供各种不同的功能。Hurd 是由 GNU 计划所设计的自由软件。

这份操作系统中的大部分的基本工具来自于 GNU 计划；因此把它们命名为 GNU/Linux 和 GNU/Hurd。这些工具同样都是自由的。

该计划至少已公开测试了 12 个 α 版本，最新版本为 K16。

Debian 现在还有基于 FreeBSD 内核的版本，它现在已经完全可用了，很多人用它来完成日常的工作，这个是 Debian 的一个子计划，叫做 Debian GNU/kFreeBSD。

Debian 另外还有基于 NetBSD 内核的计划，名字叫做 Debian GNU/NetBSD，不过这个计划还处于 α 阶段。

很多 Debian 的支持者认为，因为 Debian Project 独立运作，不带有任何商业性质，不依附任何商业公司或者机构，使得它能够有效地坚守其信奉的自由理念和风格。因为 Debian 不受任何商业公司或者机构控制，所以它不会发生为了某些商业上的利益而牺牲用户的权益，也不会因为公司经营不善或者商业模式转换等变化而导致开发作业终止。而这些特色使得 Debian 在众多的 GNU/Linux 的发布包中独树一帜。

Debian 对 GNU 和 UNIX 精神的坚持，也获得开源社区和自由软件或开源软件信奉者的支持。支持者的其他评价如下：

- Debian 是精简的 Linux 发布版，有着干净的作业环境。
- 安装步骤简易有效，大部分情况下只要 <Enter>、<Enter> 一直按下去便可以顺利安装。
- 拥有方便高效的软体包管理程序和 deb 软体包，可以让用户容易的查找、安装、移除、更新程序，或系统升级。
- 健全的软件管理制度，包括了 Bug 汇报、包维护人等制度，让 Debian 所收集的软件质量在其它的 Linux 发布包之上。
- 拥有庞大的包库，令用户只需通过其自身所带的软件管理系统便可下载并安装包，不必再在网络上查找。
- 包库分类清楚，用户可以明确地选择安装自由软件、半自由软件或闭源软件。

对 Debian 的技术性批评是，因为 Debian 的发布周期较长，稳定版本的包可能已经过时。由于 Debian 很大程度上是为“不动的”平台（例如服务器和用于开发的机器）设计，而这些平台只需要安全性的更新。

[1] Debian sid 也称为 Debian unstable，即不稳定版本，凡是 Debian 要收录的软件都必须首先放在这个版本里面进行测试，等到足够稳定以后会放到 testing 版本里面。

第 8 章

Ubuntu

Ubuntu[?] is a Debian-based Linux operating system, with Unity as its default desktop environment (GNOME was the previous desktop environment). It is based on free software and named after the Southern African philosophy of ubuntu (literally, “human-ness”), which often is translated as “humanity towards others” or ”the belief in a universal bond of sharing that connects all humanity”.

According to some metrics, Ubuntu is the most popular desktop Linux distribution. See Installed base section.

Development of Ubuntu is led by Canonical Ltd., a company based on the Isle of Man and owned by South African entrepreneur Mark Shuttleworth. Canonical generates revenue through the sale of technical support and other services related to Ubuntu. The Ubuntu project is publicly committed to the principles of open source development; people are encouraged to use free software, study how it works, improve upon it, and distribute it.

8.1 Features

Ubuntu is composed of many software packages, the majority of which are free software. Free software gives users the freedom to study, adapt/ modify, and distribute it. Ubuntu can also run proprietary software. Ubuntu Desktop is built around Unity, a graphical desktop environment.

Ubuntu comes installed with a wide range of software that includes LibreOffice, Firefox, Empathy, Transmission, and several lightweight games (such as Sudoku and chess). Additional software that is not installed by default (including software that used to be in the default installation such as Evolution, GIMP, Pidgin, and Synaptic) can be downloaded and installed using the Ubuntu Software Center[15] or other APT-based package management tools. Programs in the Software Center are mostly free, but there are also priced products, including applications and magazines. Ubuntu can also run many programs designed for Microsoft Windows (such as Microsoft Office), through Wine or using a Virtual Machine (such as VirtualBox or VMware Workstation).

The Ubiquity installer allows Ubuntu to be installed to the hard disk from within the Live CD environment.

GNOME (the former default desktop) supports more than 46 languages.

For increased security, the sudo tool is used to assign temporary privileges for performing administrative tasks, allowing the root account to remain locked, and preventing inexperienced users from inadvertently making catastrophic system changes or opening security holes. PolicyKit is also being widely implemented into the desktop to further harden the system through the principle of least privilege.

Ubuntu can close its own network ports using its own firewall software. End-users can install Gufw (GUI for Uncomplicated Firewall) and keep it enabled.

Ubuntu compiles its packages using GCC features such as PIE and Buffer overflow protection to harden its software. These extra features greatly increase security at the performance expense of 1% in 32 bit and 0.01% in 64 bit.

Beginning with Ubuntu 5.04, UTF-8 became the default character encoding, which allows for support of a variety of non-Roman scripts.

8.2 History

Ubuntu is built on Debian's architecture and infrastructure, to provide Linux server, desktop, phone, tablet and TV operating systems. Ubuntu releases updated versions predictably - every six months[8] - and that each release would receive free support for nine months (eighteen months prior to 13.04) with security fixes, other high-impact bug fixes and very conservative, substantially beneficial low-risk bug fixes. The first release was on October 2004.

It was decided that every fourth release, issued on a two-year basis, would receive long-term support (LTS). Long term support includes updates for new hardware, security patches and updates to the 'Ubuntu stack' (cloud computing infrastructure). The first LTS releases were supported for three years on the desktop and five years on the server; since Ubuntu 12.04 LTS, desktop support for LTS releases was increased to five years as well. LTS releases get regular point releases with support for new hardware and integration of all the updates published in that series to date.

Ubuntu packages are based on packages from Debian's unstable branch: both distributions use Debian's deb package format and package management tools (APT and Ubuntu Software Center). Debian and Ubuntu packages are not necessarily binary compatible with each other, however, and sometimes .deb packages may need to be rebuilt from source to be used in Ubuntu. Many Ubuntu developers are also maintainers of key packages within Debian. Ubuntu cooperates with Debian by pushing changes back to Debian, although there has been criticism that this does not happen often enough. In the past, Ian Murdock, the founder of Debian, has expressed concern about Ubuntu packages potentially diverging too far from Debian to remain compatible. Before release, packages are imported from Debian Unstable continuously and merged with Ubuntu-specific modifications. A month before release, imports are frozen, and packagers then work to ensure that the frozen features interoperate well together.

Ubuntu is currently funded by Canonical Ltd. On 8 July 2005, Mark Shuttleworth and Canonical Ltd. announced the creation of the Ubuntu Foundation and provided an initial funding of US$10 million. The purpose of the foundation is to ensure the support and development for all future versions of Ubuntu. Mark Shuttleworth describes the foundation as an "emergency fund" (in case Canonical's involvement ends).

On 12 March 2009, Ubuntu announced developer support for 3rd party cloud management platforms, such as for those used at Amazon EC2.

Beginning with version 10.10, Ubuntu Netbook Edition used the Unity desktop as its desktop interface.[34] Starting with Ubuntu 11.04, the netbook edition has been merged into the desktop edition and Unity became the default GUI for Ubuntu Desktop.

Mark Shuttleworth announced on 31 October 2011 that Ubuntu's support for smartphones, tablets, TVs and smart screens is scheduled to be added by Ubuntu 14.04. On 9 January 2012, Canonical announced Ubuntu TV at the Consumer Electronics Show.

As of version 12.04, Ubuntu supports the ARM and x86 (32 bit and 64 bit) architectures. There is unofficial support for PowerPC.

8.3 Installation

Installation of Ubuntu is generally performed with the Live CD or a Live USB drive. The Ubuntu OS can run directly from the CD (although this is usually slower than running Ubuntu from an HDD), allowing a user to "test-drive" the OS for hardware compatibility and driver support. The CD also contains the Ubiquity installer, which can then guide the user through the permanent installation process. CD images of all current and past versions are available for download at the Ubuntu web site.

Users can download a disk image (.iso) of the CD, which can then either be written to a physical medium (CD or DVD), or optionally run directly from a hard drive (via UNetbootin or GRUB). Ubuntu is also available on PowerPC, SPARC, and IA-64 platforms, although none are officially supported.

Canonical offered Ubuntu and Kubuntu Live installation CDs at no cost including paid postage for most destinations around the world via a service called ShipIt. This service closed in April 2011. The Canonical Store offers five CDs for £5.00. Various third-party programs such as remastersys and Reconstructor are available to create customized copies of the Ubuntu Live CDs.

Ubuntu and Kubuntu can be booted and run from a USB Flash drive (as long as the BIOS supports booting from USB), with the option of saving settings to the flashdrive. This allows a portable installation that can be run on any PC which is capable of booting from a USB drive. In newer versions of Ubuntu, the USB creator program is available to install Ubuntu on a USB drive (with or without a LiveCD disc).

The desktop edition can be also installed using the Netboot image which uses the debian-installer and allows certain specialist installations of Ubuntu: setting up automated deployments, upgrading from older installations without network access, LVM and/or RAID partitioning.

8.4 Package Management

Ubuntu divides most software into four domains to reflect differences in licensing and the degree of support available. Some unsupported applications receive updates from community members, but not from Canonical Ltd.

	Free software	Non-free software
Supported	Main	Restricted
Unsupported	Universe	Multiverse

Free software includes software that has met the Ubuntu licensing requirements, which roughly correspond to the Debian Free Software Guidelines. Exceptions, however, include firmware and fonts, in the Main category, because although they are not allowed to be modified, their distribution is otherwise unencumbered.

Non-free software is usually unsupported (Multiverse), but some exceptions (Restricted) are made for important non-free software. Supported non-free software includes device drivers that can be used to run Ubuntu on some current hardware, such as binary-only graphics card drivers. The level of support in the Restricted category is more limited than that of Main, because the developers may not have access to the source code. It is intended that Main and Restricted should contain all software needed for a complete desktop environment. Alternative programs for the same tasks and programs for specialized applications are placed in the Universe and Multiverse categories.

In addition to the above, in which the software does not receive new features after an initial release, Ubuntu Backports is an officially recognized repository for backporting newer software from later versions of Ubuntu.[56] The repository is not comprehensive; it consists primarily of user-requested packages, which are approved if they meet quality guidelines. Backports receives no support at all from Canonical, and is entirely community-maintained.

The -updates repository provides stable release updates (SRU) of Ubuntu and are generally installed through update-manager. Each release is given its own -updates repository (e.g. intrepid-updates). The repository is supported by Canonical Ltd. for packages in main and restricted, and by the community for packages in universe and multiverse. All updates to the repository must meet certain requirements and go through the -proposed repository before being made available to the public. Updates are scheduled to be available until the end of life for the release.

In addition to the -updates repository, the unstable -proposed repository contains uploads which must be confirmed before being copied into -updates. All updates must go through this process to ensure that the patch does truly fix the bug and there is no risk of regression. Updates in -proposed are confirmed by either Canonical or members of the community.

Canonical's partner repository lets vendors of proprietary software deliver their products to Ubuntu users at no cost through the same familiar tools for installing and upgrading software. The software in the partner repository is officially supported with security and other important updates by its respective vendors. Canonical supports the

packaging of the software for Ubuntu and provides guidance to vendors. The partner repository is disabled by default and can be enabled by the user. Some popular products distributed via the partner repository as of 28 April 2013 are Adobe Flash Player, Adobe Reader and Skype.

Ubuntu has a certification system for third party software. Some third-party software that does not limit distribution is included in Ubuntu's multiverse component. The package ubuntu-restricted-extras additionally contains software that may be legally restricted, including support for MP3 and DVD playback, Microsoft TrueType core fonts, Sun's Java runtime environment, Adobe's Flash Player plugin, many common audio/video codecs, and unrar, an unarchiver for files compressed in the RAR file format.

Additionally, third party application suites are available for purchase through Ubuntu Software Center, including many high-quality games such as Braid and Oil Rush, software for DVD playback and media codecs.

There is also Steam available for Ubuntu with a wide range of indie games such as Amnesia: The Dark Descent, as well as some AAA titles, such as Counter-Strike: Source and Half Life 2.

8.5 Releases

Each Ubuntu release has a version number that consists of the year and month number of the release. For example, the first release was Ubuntu 4.10 as it was released on 20 October 2004. Version numbers for future versions are provisional; if the release is delayed the version number changes accordingly.

Ubuntu releases are also given alliterative code names, using an adjective and an animal (e.g., "Dapper Drake" and "Intrepid Ibex"). With the exception of the first two releases, code names are in alphabetical order, allowing a quick determination of which release is newer. "We might skip a few letters, and we'll have to wrap eventually." says Mark Shuttleworth while describing the naming scheme. Commonly, Ubuntu releases are referred to using only the adjective portion of the code name; for example, the 12.04 LTS release is commonly known as "Precise".

Releases are timed to be approximately one month after GNOME releases (which in turn are about one month after releases of X.org). As a result, every Ubuntu release was introduced with an updated version of both GNOME and X. Upgrades between releases have to be done from one release to the next release (e.g. Ubuntu 10.04 to Ubuntu 10.10) or from one LTS release to the next LTS release (e.g. Ubuntu 8.04 LTS to Ubuntu 10.04 LTS).

Ubuntu 10.10 (Maverick Meerkat), was released on 10 October 2010 (10-10-10). This departed from the traditional schedule of releasing at the end of October in order to get "the perfect 10", and makes a playful reference to The Hitchhiker's Guide to the Galaxy books, since, in binary, 101010 equals decimal 42, the "Answer to the Ultimate Question of Life, the Universe and Everything" within the series.

12.10 becomes unsupported in April 2014 while 13.04 becomes unsupported in January 2014. This is because the support duration for non-LTS versions was reduced from 18 months to 9 months beginning in 13.04.

In 2013, Canonical reached an agreement with the Ministry of Industry and Information Technology of the People's Republic of China to make Ubuntu the new basis of the Kylin(麒麟) operating system starting with Raring Ringtail (version 13.04). The first version of UbuntuKylin was released on 25 April 2013.

Version	Code name	Release date	Supported until	
			Desktop	Server
4.10	Warty Warthog	2004-10-20	2006-04-30	
5.04	Hoary Hedgehog	2005-04-08	2006-10-31	
5.10	Breezy Badger	2005-10-13	2007-04-13	
6.06 LTS	Dapper Drake	2006-06-01	2009-07-14	2011-06-01
6.10	Edgy Eft	2006-10-26	2008-04-25	
7.04	Feisty Fawn	2007-04-19	2008-10-19	

Version	Code name	Release date	Supported until	
			Desktop	Server
7.10	Gutsy Gibbon	2007-10-18	2009-04-18	
8.04 LTS	Hardy Heron	2008-04-24	2011-05-12	2013-05-09
8.10	Intrepid Ibex	2008-10-30	2010-04-30	
9.04	Jaunty Jackalope	2009-04-23	2010-10-23	
9.10	Karmic Koala	2009-10-29	2011-04-30	
10.04 LTS	Lucid Lynx	2010-04-29	2013-05-09	2015-04
10.10	Maverick Meerkat	2010-10-10	2012-04-10	
11.04	Natty Narwhal	2011-04-28	2012-10-28	
11.10	Oneiric Ocelot	2011-10-13	2013-05-09	
12.04 LTS	Precise Pangolin	2012-04-26	2017-04	
12.10	Quantal Quetzal	2012-10-18	2014-04	
13.04	Raring Ringtail	2013-04-25	2014-01	
13.10	Saucy Salamander	2013-10-17	2014-07	
14.04 LTS	Trusty Tahr	2014-04-17	2019-04	

Ubuntu has a server edition that uses the same APT repositories as the Ubuntu Desktop Edition. The differences between them are the absence of an X Window environment in a default installation of the server edition (although one can easily be installed including Unity, GNOME, KDE or XFCE) and the installation process. The server edition uses a screen mode character-based interface for the installation, instead of a graphical installation process.

Ubuntu 12.04 LTS Server supports three major architectures: IA-32, X86-64 and ARM.

Ubuntu 10.04 Server Edition can also run on VMware ESX Server, Oracle's VirtualBox and VM, Citrix Systems XenServer hypervisors, Microsoft Hyper-V, QEMU, Kernel-based Virtual Machine, or any other IBM PC compatible emulator or virtualizer. Ubuntu 10.04 turns on AppArmor (security module for the Linux kernel) by default on key software packages, and the firewall is extended to common services used by the operating system. The home and Private directories can also be encrypted. It includes MySQL 5.1, Tomcat 6, OpenJDK 6, Samba 3.4, Nagios 3, PHP 5.3, Python 2.6

Ubuntu offers Ubuntu Cloud Images which are pre-installed disk images that have been customized by Ubuntu engineering to run on cloud-platforms such as Amazon EC2, Openstack, Windows and LXC. Ubuntu is also prevalent in the VPS provider Digital Ocean.

Ubuntu 11.04 added support for OpenStack, with Eucalyptus to OpenStack migration tools added by Canonical in Ubuntu Server 11.10. Ubuntu 11.10 added focus on OpenStack as the Ubuntu's preferred IaaS offering though Eucalyptus is also supported. Another major focus is Canonical Juju for provisioning, deploying, hosting, managing, and orchestrating enterprise data center infrastructure services, by, with, and for the Ubuntu Server.

The Ubuntu Developer Summit (UDS) is a gathering of software developers which occurs prior to the release of a new public version of Ubuntu.

At the beginning of a new development cycle, Ubuntu developers from around the world gather to help shape and scope the next release of Ubuntu. The summit is open to the public, but it is not a conference, exhibition or other audience-oriented event. Rather, it is an opportunity for Ubuntu developers, who usually collaborate online, to work together in person on specific tasks. From 2013 February, Ubuntu Developer Summit (UDS) is organized online through Google+ Hangouts, any number of participants and viewers can participate. Online UDS is held on two different days instead of two consecutive days. The Online UDS video is archived and is available on the website.

简介

Ubuntu（乌班图）[?] 是一个以桌面应用为主的 GNU/Linux 操作系统，其名称来自非洲南部祖鲁语或科萨语的“ubuntu”一词，意思是“人性”、“我的存在是因为大家的存在”，是非洲传统的一种价值观。

Ubuntu 由马克·舍特尔沃斯创立，其首个版本—4.10 发布于 2004 年 10 月 20 日。Ubuntu 基于 Debian 发布版和 GNOME 桌面环境，Ubuntu 的每个新版本均会包含当时最新的 GNOME 桌面环境，通常在 GNOME 发布新版本后一个月内发布。

Ubuntu 与 Debian 的不同在于它每 6 个月会发布一个新版本，每 2 年发布一个 LTS 长期支持版本[1]。Ubuntu 建构于 Debian 的不稳定分支：不论其软件格式（deb）还是软件管理与安装系统（Debian Apt）。Ubuntu 的开发者会把对软件的修改实时反馈给 Debian 社区，而不是在发布新版时才宣布这些修改。事实上，很多 Ubuntu 的开发者同时也是 Debian 主要软件的维护者。不过，Debian 与 Ubuntu 的软件并不一定完全兼容，也就是说，将 Debian 的包安装在 Ubuntu 上可能会出现兼容性问题，反之亦然。

普通的桌面版可以获得发布后 18 个月内的支持，标为 LTS（长期支持）的桌面版可以获得更长时间的支持。例如，Ubuntu 8.04 LTS（代号 Hardy Heron），其桌面应用系列可以获得为期 3 年的技术支持，服务器版可以获得为期 5 年的技术支持。而自 Ubuntu 12.04 LTS 开始，桌面版和服务器版均可获得为期 5 年的技术支持[2]。目前 Ubuntu 共有四个长期支持版本（Long Term Support，LTS）：Ubuntu 6.06、8.04、10.04 与 12.04。Ubuntu 12.04 桌面版与服务器版都有 5 年支持周期，而之前的长期支持版本为桌面版 3 年，服务器版 5 年。

Ubuntu 在 Ubuntu 12.04 的发布页面上使用了“友帮拓”一词作为其官方译名。之前曾一些中文用户使用班图、乌班图、乌斑兔、乌帮图、笨兔 [16] 等作为非官方译名。Ubuntu 在 2013 年推出了新产品 Ubuntu Phone OS 和 Ubuntu Tablet，意图统一桌面设备和移动设备的屏幕。

特色

系统管理

Ubuntu 所有系统相关的任务均需使用 sudo[3] 指令是其一大特色，这种方式比传统的以系统管理员账号进行管理工作的方式更为安全。

Ubuntu 的开发者与 Debian 和 GNOME 开源社区合作密切，其各个正式版本的桌面环境均采用 GNOME 的最新版本，通常会紧随 GNOME 项目的进展而及时更新（同时，也提供基于 KDE、XFCE 等桌面环境的派生版本）。Ubuntu 与 Debian 使用相同的 deb 软件包格式，可以安装绝大多数为 Debian 编译的软件包，虽不能保证完全兼容，但大多数情况是通用的。

开发理念

Ubuntu 计划强调易用性和国际化，以便能为尽可能多的人所用。在发布 5.04 版时，Ubuntu 就已经把万国码（UTF-8 Unicode）作为系统默认编码，用以应对各国各地区不同的语言文字，试图给用户提供一个无乱码的交流平台。它在语言支持方面，算是 Linux 发布版中相当好的。

与其它大型 Linux 厂商不同，Ubuntu 不对所谓“企业版”收取升级订购费（意即没有所谓的企业版本，人人所使用的版本皆一样，用户只有在购买官方技术支持服务时才要付费）。

此外，Ubuntu 计划强调要尽量使用自由软件，以便为各个版本的用户提供便捷的升级途径。

---

[1] 与 Debian 稳健的升级策略不同，Ubuntu 每六个月便会发布一个新版，以便人们实时地获取和使用新软件。

[2] Ubuntu 计划在 4 月 25 日 Ubuntu 13.04 发布后，将非 LTS 版本的支持时间自 18 个月缩短至 9 个月，并采用滚动发布模式，允许开发者在不升级整个发布版的情况下升级单个核心包。

[3] sudo 为 substitute user do 的简写，即超级用户的工作，在 Ubuntu 的默认环境里，root（即管理员）账号是停用的，所有与系统相关的工作指令均需在进行时于终端机接口输入 sudo 在指令前方，并输入密码确认，这样做是为了防止因一时失误对系统造成破坏。sudo 工具的默认密码是目前账户的密码。

	自由软件	非自由软件
官方支持	Main	Restricted
非官方支持	Universe	Multiverse

安装设置

一直以来，Ubuntu 均支持主流的 i386、AMD64 与 PowerPC 平台，自 2006 年 6 月，Ubuntu 新增了对升阳的 UltraSPARC 与 UltraSPARC T1 平台的支持。

许多软件（比如 remastersys 和 Reconstructor）可以将 Ubuntu 进行修改后重新打包成 Ubuntu Live CD。通过 Live CD 可以直接启动并允许用户测试硬件兼容性和驱动程序支持。

Live CD 中带有一个安装器，使用户可以将系统永久地装在计算机上。可以将 Live CD 镜像烧录到 CD 中，也可以使用一些工具（USB 启动盘创建器、UNetBootin 等）将其制作成 USB 启动盘进行测试或安装。

Live CD 中还带有一个 Wubi 工具，可以在不改变分区的情况下安装 Ubuntu（不过性能稍有一些损失）。新版 ubuntu 支持在 Windows 中进行在线安装。

由 Ubuntu 母公司 Canonical 有限公司所架设的 Launchpad 网站提供了在线翻译服务，任何人都可以通过这个网站协助翻译 Ubuntu。但是经由此方式对非 Ubuntu 独有组件的翻译成果将不会自动反馈到上游，故不被推荐。

软件包管理

Ubuntu 的包管理系统与 Debian 的类似，所有软件分为 main、restricted、universe 和 multiverse 等 4 类，每一类为一个“组件（component）”，代表着不同的使用许可和可用的支持级别。一般来说，官方支持的 main 组件主要用来满足大多数个人计算机用户的基本要求，restricted（“版权限制”）组件主要用来提高系统的可用性，因此通常需要安装这两类组件中的软件。

- main 即“基本”组件，其中只包含符合 Ubuntu 的许可证要求并可以从 Ubuntu 团队中获得支持的软件，致力于满足日常使用，位于这个组件中的软件可以确保得到技术支持和及时的安全更新。此组件内的软件是必须符合 Ubuntu 版权要求（Ubuntu license requirements）的自由软件，而 Ubuntu 版权要求大致上与 Debian 自由软件指导纲要（Debian Free Software Guidelines）相同。
- restricted 即“受限”组件，其中包含了非常重要的，但并不具有合适的自由许可证的软件，例如只能以二进制形式获得的显卡驱动程序。由于 Ubuntu 开发者无法获得相应的源代码，restricted 组件能够获得的支持与 main 组件相比是非常有限的。
- universe 即“社区维护”组件，其中包含的软件种类繁多，均为自由软件，但都不为 Ubuntu 团队所支持。
- multiverse 即“非自由”组件，其中包括了不符合自由软件要求而且不被 Ubuntu 团队支持的软件包，通常为商业公司编写的软件。

Ubuntu 有一个代号为 Grumpy Groundhog 的分支，这个分支直接从 Ubuntu 的软件版本控制系统里获取软件的源代码，主要用于测试和开发。由于这个分支不稳定，因此不对公众开放。

软件维护

Ubuntu 的新版一旦发布，该版本的包库就会被冻结，此后只对该包库提供安全性更新。为此，官方推出了一个名为 Ubuntu Backports 的后续支持计划，让用户可以在不更新包库的情况下，获得和使用各类新版的应用软件。

Ubuntu 系统通常不必安装杀毒软件。管理员们如有需要，可自行安装 ClamAV，以便扫描和清除服务器中的 Windows 病毒。Ubuntu 系统中默认带有 ufw 防火墙软件，但不提供相应的图形设置界面，用户可自行安装 firestarter，以便通过图形界面设置防火墙。

安装软件时可以通过运行 apt-get 命令，或使用图形接口的 Synaptic 工具或“软件中心”来完成。Ubuntu 能够使用的软件大多存放在被称为“软件源”的服务器中，用户只要运行相应的 apt-get 指令（或使用 Synaptic 工具进行相关操作），系统就会自动查找、下载和安装软件了。

虽然 Ubuntu 主要采用自由软件，但也接纳部分可以自由散发的私有软件，并将它们放在 multiverse 组件中。Ubuntu 还为第三方软件设立了认证程序。

由 Ubuntu 母公司 Canonical 有限公司所架设的 Launchpad 网站提供了一套在线回报软件程序错误的机制，任何人都可以把自己所发现的软件程序错误、功能缺陷和安全漏洞通过这套机制回报给开发小组。

长期支持

Ubuntu 长期支持版本更新和维护的时间比较长，约 2 年会推出一个 LTS 版本。LTS 针对企业用户，有别于一般版本的 6 个月支持。代号“Dapper Drake”的 Ubuntu 6.06 LTS 是第一个获得长期支持的版本，Canonical 公司计划对 6.06 的桌面系列版本提供 3 年的更新及付费技术支持服务，对服务器版则提供 5 年的支持。Ubuntu 6.06 LTS 包括 GNOME 2.14、Mozilla Firefox 1.5.0.3、OpenOffice.org 2.0.2、Xorg7.0、GCC 4.0.3 以及 2.6.15 版的 Linux 核心，2006 年 8 月 10 日发布的首个维护更新版本 6.06.1。

最新的长期支持版本为 2012 年 4 月 26 日推出的 12.04 LTS。

分支版本

Linux 各种发布版是使用 Linux 核心一类开放型的操作系统。由发布版定制其应用软件、桌面环境的组合和配置，因此同一发布版也可分支。所谓的 Ubuntu 系统，指的是默认的 Ubuntu 版本，采用母公司研发的 Unity 界面。11.04 版以及之前支持 Gnome 桌面环境。

Ubuntu 官方认可的分支系统众多，其主要差异在于使用的桌面系统不同，而内部的默认软件也会有所歧异。此外尚有许多基于 Ubuntu 的非官方派生版本，还有基于 Ubuntu 开发的发布版。

第 9 章

Development

The primary difference between Linux and many other popular contemporary operating systems is that the Linux kernel and other components are free and open source software. Linux is not the only such operating system, although it is by far the most widely used. Some free and open source software licenses are based on the principle of copyleft, a kind of reciprocity: any work derived from a copyleft piece of software must also be copyleft itself. The most common free software license, the GNU General Public License (GPL), is a form of copyleft, and is used for the Linux kernel and many of the components from the GNU Project.

Linux based distributions are intended by developers for interoperability with other operating systems and established computing standards. Linux systems adhere to POSIX,[68] SUS,[69] LSB, ISO, and ANSI standards where possible, although to date only one Linux distribution has been POSIX.1 certified, Linux-FT.

Free software projects, although developed in a collaborative fashion, are often produced independently of each other. The fact that the software licenses explicitly permit redistribution, however, provides a basis for larger scale projects that collect the software produced by stand-alone projects and make it available all at once in the form of a Linux distribution.

Many Linux distributions, or "distros", manage a remote collection of system software and application software packages available for download and installation through a network connection. This allows users to adapt the operating system to their specific needs.

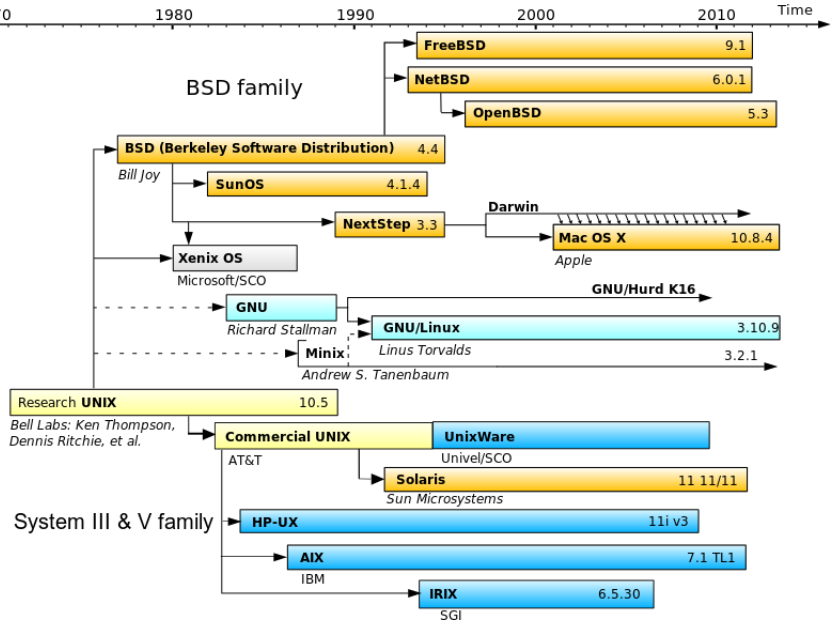
Distributions are maintained by individuals, loose-knit teams, volunteer organizations, and commercial entities. A distribution is responsible for the default configuration of the installed Linux kernel, general system security, and more generally integration of the different software packages into a coherent whole. Distributions typically use a package manager such as dpkg, Synaptic, YAST, yum, or Portage to install, remove and update all of a system's software from one central location.

Today, Linux systems are used in every domain, from embedded systems to supercomputers, and have secured a place in server installations often using the popular LAMP or LYME/LYCE application stacks. Use of Linux distributions in home and enterprise desktops has been growing.

Linux distributions have also gained popularity with various local and national governments. The federal government of Brazil is well known for its support for Linux. News of the Russian military creating its own Linux distribution has also surfaced, and has come to fruition as the G.H.ost Project. The Indian state of Kerala has gone to the extent of mandating that all state high schools run Linux on their computers. China uses Linux exclusively as the operating system for its Loongson processor family to achieve technology independence. In Spain some regions have developed their own Linux distributions, which are widely used in education and official institutions, like gnuLinEx in Extremadura and Guadalinex in Andalusia. Portugal is also using its own Linux distribution Caixa Mágica, used in the Magalhães netbook and the e-escola government program. France and Germany have also taken steps toward the adoption of Linux.

Linux distributions have also become popular in the netbook market, with many devices such as the ASUS Eee PC and Acer Aspire One shipping with customized Linux distributions installed.

Torvalds continues to direct the development of the kernel.[53] Stallman heads the Free Software Foundation, which in turn supports the GNU components.[55] Finally, individuals and corporations develop third-party



[!ht]

图 9.1: Simplified history of Unix-like operating systems. Linux shares similar architecture and concepts (as part of the POSIX standard) but does not share non-free source code with the original Unix or MINIX.

non-GNU components. These third-party components comprise a vast body of work and may include both kernel modules and user applications and libraries.

Linux vendors and communities combine and distribute the kernel, GNU components, and non-GNU components, with additional package management software in the form of Linux distributions.

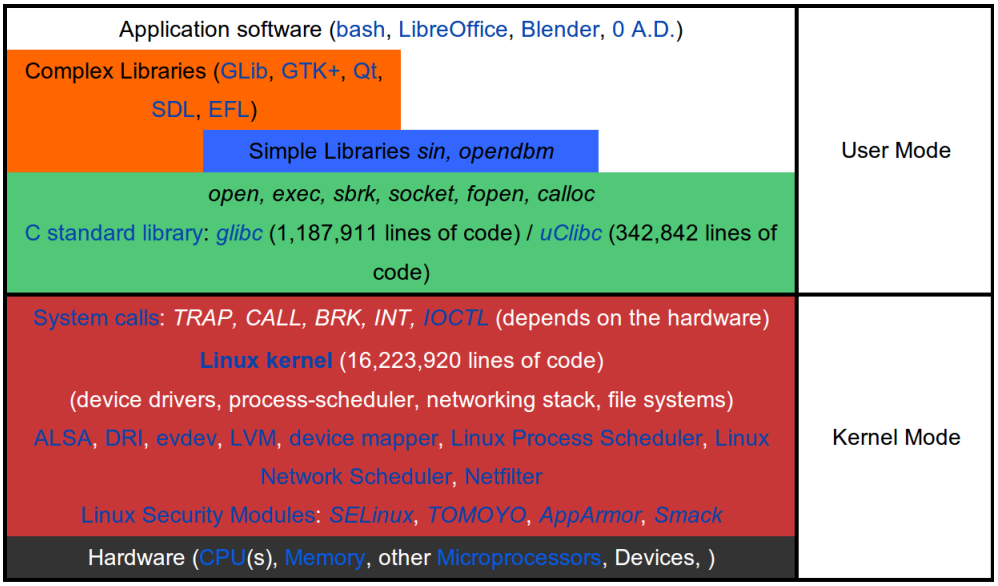
A Linux-based system is a modular Unix-like operating system. It derives much of its basic design from principles established in Unix during the 1970s and 1980s. Such a system uses a monolithic kernel, the Linux kernel, which handles process control, networking, and peripheral and file system access. Device drivers are either integrated directly with the kernel or added as modules loaded while the system is running.

Separate projects that interface with the kernel provide much of the system's higher-level functionality. The GNU userland is an important part of most Linux-based systems, providing the most common implementation of the C library, a popular shell, and many of the common Unix tools which carry out many basic operating system tasks. The graphical user interface (or GUI) used by most Linux systems is built on top of an implementation of the X Window System.

Some components of an installed Linux system are:

- A bootloader, for example GNU GRUB or LILO. This is a program which is executed by the computer when it is first turned on, and loads the Linux kernel into memory.
- An init program. This is the first process launched by the Linux kernel, and is at the root of the process tree: in other terms, all processes are launched through init. It starts processes such as system services and login prompts (whether graphical or in terminal mode).
- Software libraries which contain code which can be used by running processes. On Linux systems using ELF-format executable files, the dynamic linker which manages use of dynamic libraries is "ld-linux.so". The most commonly used software library on Linux systems is the GNU C Library. If the system is set up for the user to compile software themselves, header files will also be included to describe the interface of installed libraries.
- User interface programs such as command shells or windowing environments.

今天由 Linus Torvalds 带领下，众多开发共同参与开发和维护 Linux 内核。理查德·斯托曼领导的自由软件基金会，继续提供大量支持 Linux 内核的 GNU 组件。一些个人和企业开发的第三方的非 GNU



[!ht] Various layers within Linux, also showing separation between the userland and kernel space.

组件也提供对 Linux 内核的支持，这些第三方组件包括大量的作品，有内核模块和用户应用程序和库等内容。Linux 社区或企业都推出一些重要的 Linux 发行版，包括 Linux 内核、GNU 组件、非 GNU 组件，以及其他形式的的软件包管理系统软件。

基于 Linux 的系统是一个模块化的类 Unix 操作系统。Linux 操作系统的大部分设计思想来源于 20 世纪 70 年代到 80 年代的 Unix 操作系统所创建的基本设计思想。Linux 系统使用单内核，由 Linux 内核负责处理进程控制、网络，以及外围设备和文件系统的访问。在系统运行的时候，设备驱动程序要么与内核直接集成，要么以加载模块形式添加。

Linux 具有设备独立性，它内核具有高度适应能力，从而给系统提供了更高级的功能。GNU 用户界面组件是大多数 Linux 操作系统的重要组成部分，提供常用的 C 函数库，shell，还有许多常见的 Unix 实用工具，可以完成许多基本的操作系统任务。大多数 Linux 系统使用的图形用户界面创建在 X 窗口系统之上，由 X 窗口系统通过软件工具及架构协议来创建操作系统所用的图形用户界面。

已安装 Linux 操作系统包含的一些组件：

- 启动程序——例如 GRUB 或 LILO。该程序在计算机开机启动的时候运行，并将 Linux 内核加载到内存中。
- init 程序。init 是由 Linux 内核创建的第一个进程，称为根进程，所有的系统进程都是它的子进程，即所有的进程都是通过 init 启动。init 启动的进程如系统服务和登录提示（图形或终端模式的选择）。
- 软件库包含代码，可以通过运行的进程。在 Linux 系统上使用 ELF 格式来执行文件，负责管理库使用的动态链接器是“ld-linux.so”。Linux 系统上最常用的软件库是 GNU C 库。
- 用户界面程序，如命令行 shell 或窗口环境。

9.1 User Interface

The user interface, also known as the shell, is either a command-line interface (CLI), a graphical user interface (GUI), or through controls attached to the associated hardware, which is common for embedded systems. For desktop systems, the default mode is usually a graphical user interface, although the CLI is available through terminal emulator windows or on a separate virtual console. Most low-level Linux components, including the GNU userland, use the CLI exclusively. The CLI is particularly suited for automation of repetitive or delayed tasks, and provides very simple inter-process communication.

On desktop systems, the most popular user interfaces are the extensive desktop environments KDE Plasma Desktop, GNOME, Cinnamon, Unity, LXDE, Pantheon and Xfce,[59] though a variety of additional user interfaces exist. Most popular user interfaces are based on the X Window System, often simply called "X". It provides network transparency and permits a graphical application running on one system to be displayed on another where a user may interact with the application.

Other GUIs may be classified as simple X window managers, such as FVWM, Enlightenment, and Window Maker, which provide a minimalist functionality with respect to the desktop environments. A window manager provides a means to control the placement and appearance of individual application windows, and interacts with the X Window System. The desktop environments include window managers as part of their standard installations (Mutter for GNOME, KWin for KDE, Xfwm for Xfce as of January 2012) although users may choose to use a different window manager if preferred.

Linux currently has two modern kernel-userspace APIs for handing video input devices: V4L2 API for video streams and radio, and DVB API for digital TV reception.

Due to the complexity and diversity of different devices, and due to the large amount of formats and standards handled by those APIs, this infrastructure needs to evolve to better fit other devices. Also, a good userspace device library is the key of the success for having userspace applications to be able to work with all formats supported by those devices.

9.2 Programming

Most Linux distributions support dozens of programming languages. The original development tools used for building both Linux applications and operating system programs are found within the GNU toolchain, which includes the GNU Compiler Collection (GCC) and the GNU build system. Amongst others, GCC provides compilers for Ada, C, C++, Java, and Fortran. First released in 2003, the Low Level Virtual Machine project provides an alternative open-source compiler for many languages. Proprietary compilers for Linux include the Intel C++ Compiler, Sun Studio, and IBM XL C/C++ Compiler. BASIC in the form of Visual Basic is supported in such forms as Gambas, FreeBASIC, and XBasic, and in terms of terminal programming or QuickBASIC or Turbo BASIC programming in the form of QB64.

A common feature of Unix-like systems, Linux includes traditional specific-purpose programming languages targeted at scripting, text processing and system configuration and management in general. Linux distributions support shell scripts, awk, sed and make. Many programs also have an embedded programming language to support configuring or programming themselves. For example, regular expressions are supported in programs like grep, or locate, while advanced text editors, like GNU Emacs have a complete Lisp interpreter built-in.

Most distributions also include support for PHP, Perl, Ruby, Python and other dynamic languages. While not as common, Linux also supports C# (via Mono), Vala, and Scheme. A number of Java Virtual Machines and development kits run on Linux, including the original Sun Microsystems JVM (HotSpot), and IBM's J2SE RE, as well as many open-source projects like Kaffe and JikesRVM.

GNOME and KDE are popular desktop environments and provide a framework for developing applications. These projects are based on the GTK+ and Qt widget toolkits, respectively, which can also be used independently of the larger framework. Both support a wide variety of languages. There are a number of Integrated development environments available including Anjuta, Code::Blocks, CodeLite, Eclipse, Geany, ActiveState Komodo, KDevelop, Lazarus, MonoDevelop, NetBeans, and Qt Creator, while the long-established editors Vim, nano and Emacs remain popular.

9.3 Features

众所皆知，Linux 的内核原型是由 Linus Torvalds 在 1991 年发布的，但是 Linus Torvalds 为什么可以开发 Linux 操作系统，为什么要选择 386 的计算机来开发，为什么 Linux 的发展这么迅速，又为什么 Linux 是免费的以及目前为什么有这么多的 Linux 发行版（distributions）。弄清楚这些之后，才能知道为什么 Linux 可以免于专利软件之争，并且理解 Linux 为什么可以同时在个人计算机与服务器上大放异彩。所以在进入 Linux 的世界前，需要先来谈一谈这些历史。

Linux 是在计算机上运行的，所以首先要了解计算机，到底有哪些种类的计算机，Linux 可以在哪些种类的计算机上面运作以及 Linux 源自哪里。

在目前的生活中，应该很难不接触到计算机。当初在开发计算机的时候是希望可以辅助与简化人们进行大量的运算工作，后来才发展成为一些特殊用途。无论如何，计算机基本的功能就是：接受用户输入命令，经由 CPU 的算术与逻辑单元运算处理后，产生或存储成有用的信息。

为了实现这些功能，计算机就必须要有：

1. 输入单元：例如鼠标、键盘、读卡器等；
2. 中央处理器（CPU）：含有算术逻辑、控制、存储等单元；
3. 输出单元：例如显示器、打印机等

上面这些其实就是组成计算机的主要元件，而为了连接各个元件才有了主板，所以主机里面就包含了主板以及 CPU，还有各种需要的适配卡，再加上屏幕、键盘、鼠标等则通过与主机的连接，构成了一台可以运行的计算机。

整台主机的重点在于中央处理器（Central Processing Unit，CPU），CPU 是一个具有特定功能的芯片（Chip），里面还有微指令集。CPU 的工作主要在于管理和运算，因此 CPU 内又分为两个主要的单元，

另外，由于计算机仅认识 0 和 1，因此计算机主要是以二进制的方式来计算的，因此通常计算机的运算/存储单位都是以 Byte 或 bits 为基本单位，换算关系如下：

```
1 Bytes = 8 bits
1 KB = 1024 Bytes
1 MB = 1024 KB
1 GB = 1024 MB
```

而计算机也因为它的复杂度细分为不同等级，例如：

- 超级计算机（Supercomputer）
  超级计算机是运行速度最快的电脑，但是维护、操作费用也最高，主要是用于需要有高速计算的环境中。例如，国防军事、气象预测、太空科技等需要模拟的领域。
- 大型机（Mainframe Computer）
  大型计算机通常也具有数个高速的 CPU，功能上虽不及超级计算机，但也可用来处理大量数据与复杂的运算。例如大型企业的主机、全国性的证券交易所等每天需要处理数百万笔信息的企业机构，或者是大型企业的资料库服务器等。
- 小型机（Minicomputer）
  小型机仍具有大型电脑同时支持多用户的特性，但是主机可以放在一般场所，不必像前两个大型计算机需要特殊的空调场所。通常用来作为科学研究、工程分析与工厂的流程管理等。
- 微机（Microcomputer）
  微机又可以称为个人电脑，具有体积最小、价格最低的特点，但功能还是都具备的。个人电脑大致又可分为桌上型、笔记本型等等。

虽然在目前个人电脑的使用广泛，但是在 1990 年以前，个人电脑是不被重视的，因为其运算速度在当时实在很慢，而且当时比较有名的操作系统也没有对个人电脑提供支持，所以不太流行。

9.3.1 Free and Open Source

Linux 基于 GPL，因此任何人皆可以自由取得 Linux，不同于 UNIX 需要负担庞大的版权费用，也不同于微软需要一而再、再而三的更新系统并且要缴纳大量正版授权费用。

由于 Linux 是基于 GPL（General Public License）许可证下，因此它是自由软件，也就是任何人都可以自由的使用或者是修改其中的源代码，这就是所谓的“开放性架构”。很多的工程师由于特殊的需求常常需要修改系统的源代码从而使该系统可以符合自己的需求，而这个开放性的架构将可以满足各不同需求的工程师，因此就有可能越来越流行。

Linux 可以支持个人计算机的 x86 架构，从而不必像 UNIX 系统那样仅适合于某一公司（例如 Sun、IBM 等）的设备。

目前，自由开源意识形态与商业用途的冲突、缺乏强而有力的推广厂商、缺乏对特殊的硬件和应用程序的支持、电脑技术人员不愿再花费时间重复学习、对已有平台的依赖，是制约 Linux 被采纳的主要因素。

虽然早在 1994 年 Linux 1.0 版发布时，就已经含有 XFree86 的 X Window 架构了。不过 X Window 毕竟是 Linux 上的一个软件，并不是 Linux 最核心的部分，有没有它对 Linux 的服务器运行都没有影响。

现在已有 KDE[1] 及 GNOME[2] 等优秀的视窗管理程序，不过毕竟整合度还是需要加强，未来有可能看到整合度超高的 Linux Desktop 型计算机。

虽然能在 Windows 或 Mac OS 上运行的应用软件大部分都没有 Linux 的版本，不过在 Linux 平台上通常可以找到类似功能的应用软件。

大多数在 Windows 平台上广泛使用的自由软件都有相应的 Linux 版本，例如 Mozilla Firefox、Apache OpenOffice、Pidgin、VLC、GIMP。部分流行的专有软件也有相应的 Linux 版本，如 Adobe Flash Player、Adobe Reader、Google Earth、Nero Burning ROM、Opera、Maple、MATLAB、Skype、Maya、SPSS、Google Chrome。

另外，相当多的 Windows 应用程序可以通过 Wine 和一些基于 Wine 的项目如 CrossOver 正常运行和工作。如 Microsoft Office、Adobe Photoshop、暴雪娱乐的游戏、Picasa 其中对于 Photoshop 的 Crossover（Wine）兼容性工作有 Disney、DreamWorks、Pixar 投资支持，等。Google 大力参与 Wine 项目改进，Picasa 的 GNU/Linux 版本也是经 Wine 测试的 Windows 平台编译版本。

但是，Linux 下也有相当多不能在 Windows 平台下运行的软件，主要是依靠 X 窗口系统和其他 Windows 无法利用的资源，或者是因为稳定性等其他方面的考虑并不准备支持 Windows。不过近年来，也不断向其移植。有如 KDE SC、Cinepaint 正在进行向 Windows 的移植。Linux 使用的增多也使得 Windows 开源软件向 Linux 移植，比如 Filezilla。

9.3.2 Interfaces

Linux 是基于 UNIX 概念而开发出来的操作系统，因此 Linux 具有与 UNIX 系统相似的的程序接口跟操作方式，也继承了 UNIX 稳定并且有效率的特点。由于 Linux 功能并不会输给一些大型的 UNIX 工作站，因此近年来越来越多的公司或者是团体、个人投入这一操作系统的开发与整合工作。

Linux 支持比较公认而正式的标准，例如开发时就使用的 POSIX 规范。为了让软件开发商、与硬件开发者有一个依循的方向，因此发布了 Linux Standard Base（LSB）及 File system Hierarchy Standard（FHS）等。

各个 distribution 也都要遵循 LSB 上面的规范，软硬件开发者也都会依循 LSB，所以我们才会常说各大 distribution 虽然在提供的工具与创意上面有所不同，但是基本上它们的架构都是很类似的。

- FHS: http://www.pathname.com/fhs/
- LSB: http://www.linuxbase.org/

另外，由于很多的软件套件逐渐被 Linux 操作系统来使用，而且很多套件软件也都在 Linux 这个操

[1] http://www.kde.org/
[2] http://www.gnome.org/

作系统上面进行开发与测试，因此 Linux 已经可以独力完成几乎所有的工作站或服务器的服务了，例如 Web、Mail、Proxy、FTP 等。目前 Linux 已经是相当成熟的操作系统而且消耗资源低又可以自由获得。

Linux 发行版一般被用来作为服务器的操作系统，并且已经在该领域中占据重要地位。Linux 发行版是构成 LAMP[3]（Linux 操作系统，Apache，MySQL，Perl/PHP/Python）的重要部分。

FreeBSD 具备完整的 NIS 客户端和服务器端支持，事务式 TCP 协议支持，按需拨号的 PPP，集成的 DHCP 支持，改进的 SCSI 子系统，ISDN 的支持，ATM，FDDI，快速 Gigabit 以太网 (1000 Mbit) 支持，并且提升了最新的 Adaptec 控制器的支持和修补了很多的错误。

9.3.3 Multi-user & Multi-tasking

Linux 主机上可以同时允许多用户同时在线工作，并且资源的分配较为公平。可以在 Linux 主机上面规划出不同等级的用户，而且每个用户登录系统时的工作环境都可以不相同。此外还可以允许不同的用户在同一个时间登入主机以同时使用主机的资源。

要达到多任务(multitasking)的环境，除了硬件(主要是 CPU)需要能够具有多任务的特性外，操作系统也需要支持这个功能。

一些不具有多任务特性的操作系统，想要同时执行两个程序是不可能的。除非先被执行的程序执行完毕，否则后面的程序不可能被主动

这有点像在开记者发布会，主持人(CPU)会问“谁要发问”？一群记者(工作程序) 就会举手(看谁的工作重要)，先举手的自然就被允许

在 Linux 系统中，文件的属性可以分为“可读、可写、可执行”等参数来定义一个文件的适用性，而且这些属性还可以分为三个种类——分别是“文件拥有者、文件所属群组、其他非拥有者与用户组者”，这对于项目计划或者开发者具有良好的安全性。

9.3.4 Embedded Devices

由于 Linux 只要几百 K 不到的程序代码就可以完整地驱动整个计算机的硬件并成为一个完整的操作系统，因此相当适合作为家电或者电子设备的操作系统，比如手机、数码相机、PDA、家电用品等嵌入式系统，通过裁剪 Linux kernel，可以使其运行在省电以及较低硬件资源的环境下。

Linux 的低成本、强大的定制功能以及良好的移植性能，使得 Linux 在嵌入式系统方面也得到广泛应用。流行的 TiVo 数字视频录像机还采用了定制的 Linux，思科在网络防火墙和路由器也使用了定制的 Linux。Korg OASYS、Korg 的 KRONOS、雅马哈的 YAMAHA MOTIF XS/Motif XF 音乐工作站、雅马哈的 S90XS/S70XS、雅马哈 MOX6/MOX8 次合成器、雅马哈 MOTIF-RACK XS 音源模块，以及 Roland RD-700GX 数码钢琴均运行 Linux。Linux 也用于舞台灯光控制系统，如 WholeHogIII 控制台。

在手机、平板电脑等移动设备方面，Linux 也得到重要发展，基于 Linux 内核的操作系统也成为最广泛的操作系统。基于 Linux 内核的 Android 操作系统已经超越诺基亚的 Symbian 操作系统，成为当今全球最流行的智能手机操作系统。

在 2010 年第三季度，销售全球的全部智能手机中使用 Android 的占据 25.5%（所有的基于 Linux 的手机操作系统在这段时间为 27.6%）。从 2007 年起，手机和掌上电脑上运行基于 Linux 的操作系统变得更加普遍，例如诺基亚 N810，Openmoko 的“的 Neo1973，摩托罗拉的 ROKR E8。

Palm（后来被 HP 公司收购）推出了一个新的基于 Linux 的 webOS 操作系统，并使用在新生产的 Palm Pre 智能手机上。

MeeGo 是诺基亚和英特尔于 2010 年 2 月联合推出的基于 Linux 的操作系统，诺基亚也推出了使用 MeeGo 操作系统的 N9 手机。

2011 年 9 月 28 日，继诺基亚宣布放弃开发 MeeGo 之后，英特尔也正式宣布将 MeeGo 与 LiMo 合并成为新的系统 Tizen。Jolla Mobile 公司成立并推出了由 MeeGo 发展而来的 Sailfish 操作系统。

[3]LAMP 是一个常见的网站托管平台，在开发者中已经得到普及。

另外，FreeBSD 通过整合虚拟内存/文件系统中的高速缓存改进了虚拟内存系统，它不仅提升了性能，而且减少了 FreeBSD 对内存的需要，使得 5 MB 内存成为可接受的最小配置。

9.3.5 Security

在网络上，“没有绝对安全的主机”。不过 Linux 由于支持者众多，有相当多的团体和个人参与开发，可以随时获得最新的安全信息并给予及时的更新，从而也达到了具有相对的安全性。

9.4 Appraisal

人们对 Linux 的正面评价如下：

- 开放源代码的 Linux 可以让知识延续下去，新兴的软件公司可以从开放源代码上快速、低价的
- 创建专业能力，丰富市场的竞争，防止独霸软件巨兽的存在。
- 个人使用很少有版权问题，绝大多数都是免费使用，几乎无所谓盗版问题。
- 新的 Linux 发布版大多数软件都有服务器的服务，只要点击就可以自动下载、安装经过认证的软件，不需要到市面购买、安装。
- Linux 学习的投资有效时间较长。旧版软件、系统都还是存在，有源代码可以派生、分支，维护周期普遍比 Windows 长很多。就算被放弃，还是可以凭借源代码派生。新的软件更新发展多样化，容易养成用户习惯掌握原理，而不是养成操作习惯。
- 强大的 Shell 及脚本支持，容易组合出符合需求的环境或创造自动程序。
- 默认安全设置相对于目前主流的 Windows 操作系统安全很多。Windows 操作系统为了非专业用户降低了默认安全性的设置，导致系统容易受到木马、病毒的侵害。盗版的 Windows 更糟糕，可能随盗版操作系统捆绑木马、恶意程序，部分默认超级用户（Administrator）登录、关闭安全更新等问题导致安全性更差。

当然，也有一些对 Linux 的负面评价：

- BSD 的开发人员曾经批评过 Linux 内核开发人员过于重视新功能的添加而不是踏踏实实的把代码写好、整理好。
- Solaris 系统管理员则认为 SMF、ZFS、DTrace 等 Solaris 特有工具使得 Solaris 比 Linux 更加优秀。
- Minix 爱好者认为微内核是将来技术发展的方向，Linux 在技术上是落伍陈旧的。(参见塔能鲍姆-林纳斯辩论[?])
- 软硬件支持性较差。大部份的软、硬件厂商没有或者不会优先开发 Linux 平台的版本，或者 Linux 平台的版本功能较少，致使可用的应用程序、硬件周边支持性相较于 Windows、Mac 平台差。
- 相当多的发布版（超过 200 个以上），使程序开发者无法针对所有发布版做测试，使用 Linux 平台的应用软件安装在非主流发布版可能遭遇预料之外的问题或甚至于无法使用。
- Linux 系统及相关应用软件主要是由黑客等程序员及其它 Linux 爱好者共同合作开发出来的，所以缺少了商业软件基于商业利益而调整操作界面使之更适合不同用户的行为。对 Linux 使用方式的不习惯，以及不同软件操作方式缺乏一致性使得用户产生 Linux 系统难以使用的感受。

第 10 章

Community

A distribution is largely driven by its developer and user communities. Some vendors develop and fund their distributions on a volunteer basis, Debian being a well-known example. Others maintain a community version of their commercial distributions, as Red Hat does with Fedora and SUSE does with openSUSE.

In many cities and regions, local associations known as Linux User Groups (LUGs) seek to promote their preferred distribution and by extension free software. They hold meetings and provide free demonstrations, training, technical support, and operating system installation to new users. Many Internet communities also provide support to Linux users and developers. Most distributions and free software / open source projects have IRC chatrooms or newsgroups. Online forums are another means for support, with notable examples being LinuxQuestions.org and the various distribution specific support and community forums, such as ones for Ubuntu, Fedora, and Gentoo. Linux distributions host mailing lists; commonly there will be a specific topic such as usage or development for a given list.

There are several technology websites with a Linux focus. Print magazines on Linux often include cover disks including software or even complete Linux distributions.

Although Linux distributions are generally available without charge, several large corporations sell, support, and contribute to the development of the components of the system and of free software. An analysis of the Linux kernel showed 75 percent of the code from December 2008 to January 2010 was developed by programmers working for corporations, leaving about 18 percent to volunteers and 7% unclassified. Some of the major corporations that contribute include Dell, IBM, HP, Oracle, Sun Microsystems (now part of Oracle), SUSE, and Nokia. A number of corporations, notably Red Hat, Canonical, and SUSE, have built a significant business around Linux distributions.

The free software licenses, on which the various software packages of a distribution built on the Linux kernel are based, explicitly accommodate and encourage commercialization; the relationship between a Linux distribution as a whole and individual vendors may be seen as symbiotic. One common business model of commercial suppliers is charging for support, especially for business users. A number of companies also offer a specialized business version of their distribution, which adds proprietary support packages and tools to administer higher numbers of installations or to simplify administrative tasks.

Another business model is to give away the software in order to sell hardware. This used to be the norm in the computer industry, with operating systems such as CP/M, Apple DOS and versions of Mac OS prior to 7.6 freely copyable (but not modifiable). As computer hardware standardized throughout the 1980s, it became more difficult for hardware manufacturers to profit from this tactic, as the OS would run on any manufacturer's computer that shared the same architecture.

第 11 章

Distributions

Linux 其实就是一个操作系统最底层的核心及其提供的核心工具。Linux 遵循 GNU 授权模式，任何人均可取得源代码并能够执行这个核心程序，而且还可以修改它。此外 Linux 遵循 POSIX 设计规范，因此相容于 UNIX 操作系统，故亦可称之为 UNIX-like 的一种。[1]

Linux 的出现，使得 GNU 软件大多以 Linux 为主要操作系统来进行开发，此外很多其他的自由软件团队，例如 sendmail、wu-ftp、apache 等也都有以 Linux 为开发测试平台的计划出现。如此一来，除了主要的核心程序外，可以在 Linux 上面运行的软件也越来越多，如果有心就能够组成完整的 Linux 操作系统。

虽然由 Torvalds 负责开发的 Linux 仅具有 kernel 与 kernel 提供的工具，不过如上所述，很多的软件已经可以在 Linux 上面运行了，因此 Linux + 软件就可以完成一个相当完整的操作系统了。不过要完成这样的操作系统，还是有一定难度的。

因为 Linux 早期都是由骇客工程师所开发维护的，他们并没有考虑到一般用户的能力。为了让用户能够接触到 Linux，很多的商业公司或非营利团体就将 Linux kernel（含 tools）与可运行的软件整合起来，加上自己具有创意的工具程序，这个工具程序可以让用户以光盘或者通过网络直接安装/管理 Linux 系统。这个 kernel + Softwares + Tools 的可完整安装的组合，称之为 Linux distribution，一般译成可完整安装套件或者发行版等。

现在，Linux 发布版指的就是通常所说的“Linux 操作系统”，它可能是由一个组织，公司或者个人发布的。

> 由于 Linux 核心是由骇客工程师写的，要由源代码安装到 x86 计算机上面成为可以执行的二进制文件，这个过程提高了 Linu
>
> 一直到一些社群与商业公司将 Linux 核心配合自由软件并提供完整的安装程序，并制成光盘后，对于一般用户来说，Linux 才越

Linux 主要作为 Linux 发布版（通常被称为“distro”）的一部分而使用。通常来讲，一个 Linux 发布版包括 Linux 内核，将整个软件安装到电脑上的一套安装工具，各种 GNU 软件以及其他的一些自由软件。在一些特定的 Linux 发布版中也有一些专有软件。发布版为许多不同的目的而制作，包括对不同电脑硬件结构的支持，对一个具体区域或语言的本地化，实时应用和嵌入式系统等。

GNU 的 GPL 授权并非不能从事商业行为，于是便有很多商业公司销售 Linux distribution。由于 Linux 的 GPL 版权协议，因此商业公司所销售的 Linux distributions 通常也都可以从 Internet 上面下载的。此外，如果您想要其他商业公司的服务，那么直接向该公司购买光盘来安装，也是一个很不错的方式。

不过，由于开发 Linux distributions 的公司实在太多了，例如有名的 Red Hat、Mandriva、Debian、SuSE 等，所以很多人都很担心，如此一来每个 distribution 是否都不相同呢？这就不需要担心了，因为每个 Linux distributions 使用的 kernel 都是http://www.kernel.org/发布的，而他们所选择的软件，几乎都是目前很知名的软件，重复程度相当的高，例如 WWW 服务器的 Apache，Mail 服务器的 Postfix/sendmail，File 服务器的 Samba 等。

此外，为了让所有的 Linux distributions 开发不致于差异太大，还有 Linux Standard Base（LSB）以及目

[1] 在 Linux 发布之前，GNU 一直以来就是缺乏核心程序，导致 GNU 自由软件只能在其他的 UNIX 上面运行。既然有了 Linux，并且 Linux 也用了很多的 GNU 相关软件，所以 Stallman 认为 Linux 的全名应该称之为 GNU/Linux。

录架构的 File system Hierarchy Standard（FHS）来规范开发者。唯一有差别的可能就是该开发者自家所开发出来的管理工具以及套件管理的模式。基本上每个 Linux distributions 除了架构的严谨度与选择的套件内容外，其实差异并不太大，大家可以选择自己喜好的 distribution 来安装即可。

一个典型的 Linux 发布版包括：Linux 内核，一些 GNU 库和工具，命令行 shell，图形界面的 X 窗口系统和相应的桌面环境，如 KDE 或 GNOME，并包含数千种从办公包，编译器，文本编辑器，小游戏，儿童教育软件，到科学工具的应用软件。

下面列出几个主要的 Linux distributions 网址：

- Red Hat：http://www.redhat.com/
- Fedora：http://fedora.redhat.com/
- Mandriva：http://www.mandriva.com/
- Novell SuSE：http://www.novell.com/linux/suse/
- Debian：http://www.debian.org/
- Slackware：http://www.slackware.com/
- Linpus：http://www.linpus.com.tw/
- Gentoo：http://www.gentoo.org/
- Ubuntu：http://www.ubuntulinux.org/
- CentOS：http://www.centos.org/

很多版本 Linux 发布版使用 LiveCD 技术，也就是不需要安装，放入光盘驱动器里面进行开机，就能使用。如果只是想看看 Linux，就可以选择可启动光盘进入 Linux 的 Live CD 版本，比较著名的有 Damn Small Linux，Knoppix 等。LiveCD 的相关技术进步到现在，有些发布版本身的安装光盘也有 LiveCD 的功能，如 Fedora，Ubuntu 等。

此外，如果还想要知道更多的 Linux distributions 的下载与使用信息，可以参考：

```
http://www.linuxiso.org/
http://distrowatch.com/
```

如同上面提到的，其实每个 distributions 差异性并不大。不过由于套件管理的方式主要分为 Debian 的 `dpkg` 及 Red Hat 系统的 `RPM` 方式，目前的建议是，先学习以 `RPM` 套件管理为主的 `Fedora/SuSE/Mandriva` 等用户较多的版本，这样一来，发生问题时可以提供解决的管道比较多。如果已经接触过 Linux 了，还想要更严谨的 Linux 版本，那可以考虑使用 Debian，如果以效率至上来考量，那么或许 Gentoo 是不错的建议。总之版本很多，但是各版本差异其实不大，建议一定要先选定一个版本，先彻头彻尾的了解它，那再继续使用其他的版本时就可以很快的进入状况。

由于开发的相关理念与相容性，因此也可以称 Linux 为 UNIX-like 操作系统的一种。

> 其实 `UNIX-like` 可以说是目前服务器类型的操作系统的统称。因为不论是 `FreeBSD、BSD、Sun UNIX、HP UNIX、Red Hat Li`

今天各种场合都有使用各种 Linux 发行版，从嵌入式设备到超级电脑，并且在服务器领域确定了地位，通常服务器使用 LAMP 组合。在家庭与企业中使用 Linux 发行版的情况越来越多。并且在政府中也很受欢迎，巴西联邦政府由于支持 Linux 而世界闻名。有新闻报道俄罗斯军队自己制造的 Linux 发布版的，做为 G.H.ost 项目已经取得成果。印度的 Kerala 联邦计划在向全联邦的高中推广使用 Linux。中华人民共和国为取得技术独立，在龙芯过程中排他性地使用 Linux。在西班牙的一些地区开发了自己的 Linux 发布版，并且在政府与教育领域广泛使用，如 Extremadura 地区的 gnuLinEx 和 Andalusia 地区的 Guadalinex。葡萄牙同样使用自己的 Linux 发布版 Caixa Mágica，用于 Magalh?es 笔记本电脑和 e-escola 政府软件。法国和德国同样开始逐步采用 Linux。

Linux 发布版同样在笔记本电脑市场很受欢迎，像 ASUS Eee PC 和 Acer Aspire One，销售时安装有订制的 Linux 发布版。

传统的 Linux 用户一般都是专业人士。他们愿意安装并设置自己的操作系统，往往比其他操作系统的用户花更多的时间在安装并设置自己的操作系统。这些用户有时被称作“黑客”或是“极客”。

使用 Linux 主要的成本为移植、培训和学习的费用，早期由于会使用 Linux 的人较少，并且在软件设计时并未考虑非专业者的使用，导致这方面费用极高。但这方面的费用已经随着 Linux 的日益普及和 Linux 上的软件越来越多、越来越方便而降低，但专业仍是使用 Linux 的主要成本。

然而随着 Linux 慢慢开始流行，有些原始设备制造商（OEM）开始在其销售的电脑上预装上 Linux，Linux 的用户中也有了普通电脑用户，Linux 系统也开始慢慢出现在个人电脑操作系统市场。Linux 在欧洲、美国和日本的流进程度较高，欧美地区还发布 Linux 平台的游戏和其他家用软件。Linux 开源社区方面也是以欧洲、美国、日本等发达国家的人士居多。

每个孩子一台笔记本电脑这一项目正在催生新的更为庞大的 Linux 用户群，计划将包括发展中国家的几亿学童、他们的家庭和社区。在 2007 年，已经有六个国家订购了至少每个国家一百万台以上免费发放给学生。Google、Red Hat 和 eBay 是该项目的主要支持者。

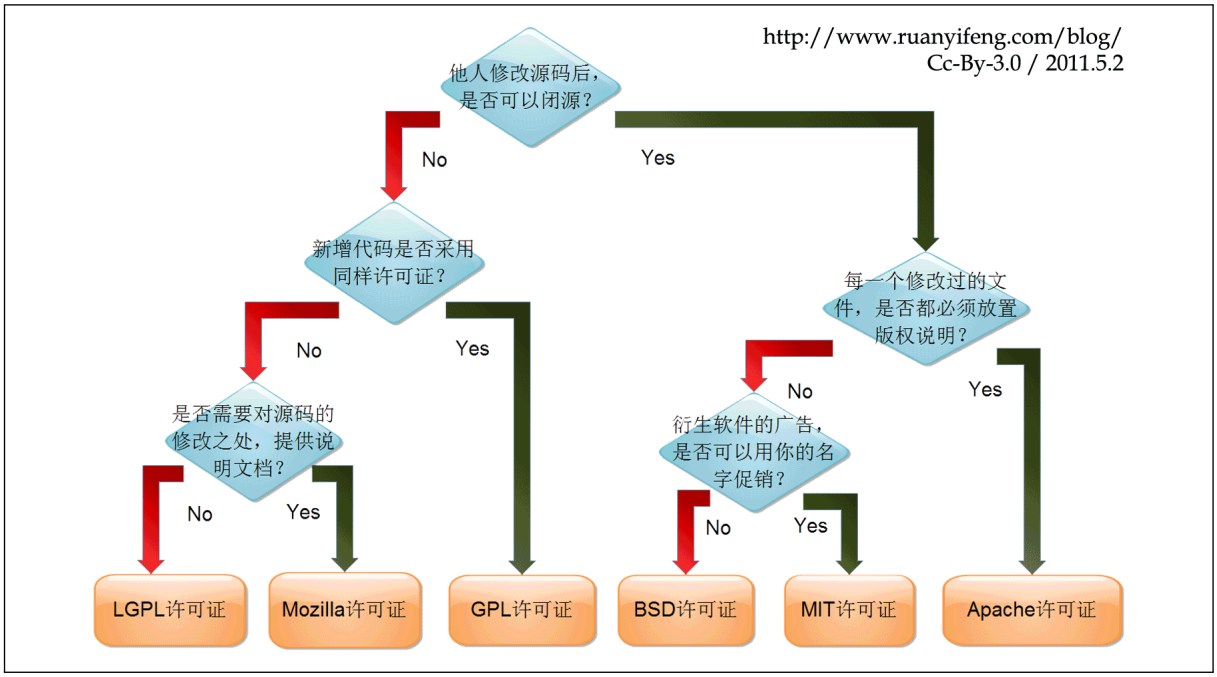
基于其低廉成本与高度可设置性，Linux 常常被应用于嵌入式系统，例如机上盒、移动电话及移动设备等。在移动电话上，Linux 已经成为 IOS 的主要竞争者；而在移动设备上，则成为 Windows CE 与 Palm OS 外之另一个选择。目前流行的 TiVo 数字摄影机使用了经过定制化后的 Linux。此外，有不少硬件式的网络防火墙及路由器，例如部份 LinkSys 的产品，其内部都是使用 Linux 来驱动、并采用了操作系统提供的防火墙及路由功能。

11.1 Licensing

现今存在的开源协议很多，而经过 Open Source Initiative 组织通过批准的开源协议目前有 58 种（http://www.opensource.org/licenses/alphabetical）。开源软件或开源组件都会基于某种协议[?]来提供源码和授权，其代表授权为 GNU 的 GPL 及 BSD 等，而且这些开源协议确实具有相应的约束。

常见的开源协议如 BSD, GPL, LGPL,MIT 等都是 OSI 批准的协议。如果要开源自己的代码，最好也是选择这些被批准的开源协议。

乌克兰程序员Paul Bagwell画了一张分析图，说明应该怎么选择开源协议。阮一峰[?]将其翻译成中文如下：



11.1.1 GNU General Public License

在自由软件所使用的各种许可证之中，最为人们注意的也许是通用性公开许可证 (General Public License，简称 GPL)[?]。目前有 version 2，version 3 两种版本，主要定义在“自由软件”上面，任何遵循 GPL 授权的软件，需要公布其源代码(Open Source)，Linux 使用的是 version 2 这一版。

GPL 同其它的自由软件许可证一样，许可社会公众享有：运行、复制软件的自由，发行传播软件的自由，获得软件源码的自由，改进软件并将自己作出的改进版本向社会发行传播的自由。

GPL 还规定：只要这种修改文本在整体上或者其某个部分来源于遵循 GPL 的程序，该修改文本的整体就必须按照 GPL 流通，不仅该修改文本的源码必须向社会公开，而且对于这种修改文本的流通不准许附加修改者自己作出的限制。因此，一项遵循 GPL 发布的程序不能同非自由的软件合并。GPL 所表达的这种流通规则称为 copyleft，表示与 copyright(版权) 的概念“相左”。

GPL 有几个主要的大方向：

1. 任何个人或公司均可发布自由软件(free software)；
2. 任何发布自由软件的个人或公司，均可由自己的服务来收取适当的费用；
3. 该软件的源代码(Source Code)需要随软件附上，并且是可公开发表的；
4. 任何人均可通过任何正常管道取得此自由软件，且均可取得此一授权模式。

GPL 协议最主要的几个原则：

- 确保软件自始至终都以开放源代码形式发布，保护开发成果不被窃取用作商业发售。任何一套软件，只要其中使用了受 GPL 协议保护的第三方软件的源程序，并向非开发人员发布时，软件本身也就自动成为受 GPL 保护并且约束的实体。也就是说，此时它必须开放源代码。
- GPL 大致就是一个左侧版权(Copyleft，或译为“反版权”、“版权属左”、“版权所无”、“版责”等) 的体现。你可以去掉所有原作的版权信息，只要你保持开源，并且随源代码、二进制版附上 GPL 的许可证就行，让后人可以很明确地得知此软件的授权信息。GPL 精髓就是，只要使软件在完整开源的情况下，尽可能使使用者得到自由发挥的空间，使软件得到更快更好的发展。
- 无论软件以何种形式发布，都必须同时附上源代码。例如在 Web 上提供下载，就必须在二进制版本(如果有的话)下载的同一个页面，清楚地提供源代码下载的链接。如果以光盘形式发布，就必须同时附上源文件的光盘。
- 开发或维护遵循 GPL 协议开发的软件的公司或个人，可以对使用者收取一定的服务费用。但还是一句老话——必须无偿提供软件的完整源代码，不得将源代码与服务做捆绑或任何变相捆绑销售。

11.1.2 Berkeley Software distribution

BSD 授权模式授权模式其实与 GPL 很类似，而其精神也与 Open Source 相呼应。

BSD 开源协议[?] 是一个给于使用者很大自由的协议。可以自由的使用，修改源代码，也可以将修改后的代码作为开源或者专有软件再发布。当你发布使用了 BSD 协议的代码，或者以 BSD 协议代码为基础做二次开发自己的产品时，需要满足三个条件：

1. 如果再发布的产品中包含源代码，则在源代码中必须带有原来代码中的 BSD 协议。
2. 如果再发布的只是二进制类库/软件，则需要在类库/软件的文档和版权声明中包含原来代码中的 BSD 协议。
3. 不可以用开源代码的作者/机构名字和原来产品的名字做市场推广。

BSD 代码鼓励代码共享，但需要尊重代码作者的著作权。BSD 由于允许使用者修改和重新发布代码，也允许使用或在 BSD 代码上开发商业软件发布和销售，因此是对商业集成很友好的协议。很多的公司企业在选用开源产品的时候都首选 BSD 协议，因为可以完全控制这些第三方的代码，在必要的时候可以修改或者二次开发。

11.1.3 Apache License, Version 2.0

Apache 是服务器软件, 在 Apache 的授权中规定, 如果想要重新发布此软件时 (如果修改过该软件), 软件的名称依旧需要命名为 Apache 才可以。

Apache Licence[?] 是著名的非盈利开源组织 Apache 采用的协议。该协议和 BSD 类似, 同样鼓励代码共享和尊重原作者的著作权, 同样允许代码修改, 再发布 (作为开源或商业软件)。需要满足的条件也和 BSD 类似:

- 需要给代码的用户一份 Apache Licence
- 如果你修改了代码, 需要在被修改的文件中说明。
- 在延伸的代码中 (修改和有源代码衍生的代码中) 需要带有原来代码中的协议, 商标, 专利声明和其他原来作者规定需要包含的说明。
- 如果再发布的产品中包含一个 Notice 文件, 则在 Notice 文件中需要带有 Apache Licence。你可以在 Notice 中增加自己的许可, 但不可以表现为对 Apache Licence 构成更改。

Apache Licence 也是对商业应用友好的许可。使用者也可以在需要的时候修改代码来满足需要并作为开源或商业产品发布/销售。

11.1.4 MIT

MIT 许可证[?] 之名源自麻省理工学院 (Massachusetts Institute of Technology, MIT), 又称「X 条款」(X License) 或「X11 条款」(X11 License)

MIT 是和 BSD 一样宽范的许可协议, 作者只想保留版权, 而无任何其他了限制。也就是说, 你必须在你的发行版里包含原许可协议的声明, 无论你是以二进制发布的还是以源代码发布的。

其中, MIT 内容与三条款 BSD 许可证 (3-clause BSD license) 内容颇为近似, 但是赋予软体被授权人更大的权利与更少的限制。

- 被授权人有权利使用、复制、修改、合并、出版发行、散布、再授权及贩售软体及软体的副本。
- 被授权人可根据程式的需要修改授权条款为适当的内容。
- 在软件和软件的所有副本中都必须包含版权声明和许可声明。

此授权条款并非属 copyleft 的自由软体授权条款, 允许在自由/开放源码软体或非自由软体 (proprietary software) 所使用。

此亦为 MIT 与 BSD (The BSD license, 3-clause BSD license) 本质上不同处。

MIT 条款可与其他授权条款并存。另外, MIT 条款也是自由软体基金会 (FSF) 所认可的自由软体授权条款, 与 GPL 相容。

11.1.5 MPL

MPL[?] 是 The Mozilla Public License 的简写, 是 1998 年初 Netscape 的 Mozilla 小组为其开源软件项目设计的软件许可证。MPL 许可证出现的最重要原因就是, Netscape 公司认为 GPL 许可证没有很好地平衡开发者对源代码的需求和他们利用源代码获得的利益。同著名的 GPL 许可证和 BSD 许可证相比, MPL 在许多权利与义务的约定方面与它们相同 (因为都是符合 OSIA 认定的开源软件许可证)。但是, 相比而言 MPL 还有以下几个显著的不同之处:

1. MPL 虽然要求对于经 MPL 许可证发布的源代码的修改也要以 MPL 许可证的方式再许可出来, 以保证其他人可以在 MPL 的条款下共享源代码。但是, 在 MPL 许可证中对“发布”的定义是“以源代码方式发布的文件”, 这就意味着 MPL 允许一个企业在自己已有的源代码库上加一个接口, 除了接口程序的源代码以 MPL 许可证的形式对外许可外, 源代码库中的源代码就可以不用 MPL 许可证的方式强制对外许可。这些, 就为借鉴别人的源代码用做自己商业软件开发的行为留了一个豁口。

2. MPL 许可证第三条第 7 款中允许被许可人将经过 MPL 许可证获得的源代码同自己其他类型的代码混合得到自己的软件程序。
3. 对软件专利的态度，MPL 许可证不像 GPL 许可证那样明确表示反对软件专利，但是却明确要求源代码的提供者不能提供已经受专利保护的源代码（除非他本人是专利权人，并书面向公众免费许可这些源代码），也不能在将这些源代码以开放源代码许可证形式许可后再去申请与这些源代码有关的专利。
4. 对源代码的定义
   而在 MPL（1.1 版本）许可证中，对源代码的定义是："源代码指的是对作品进行修改最优先择取的形式，它包括所有模块的所有源程序，加上有关的接口的定义，加上控制可执行作品的安装和编译的'原本'（原文为'Script'），或者不是与初始源代码显著不同的源代码就是被源代码贡献者选择的从公共领域可以得到的程序代码。"
5. MPL 许可证第 3 条有专门的一款是关于对源代码修改进行描述的规定，就是要求所有再发布者都得有一个专门的文件就对源代码程序修改的时间和修改的方式有描述。

11.1.6 LGPL

LGPL[?] 是 GPL 的一个为主要为类库使用设计的开源协议。和 GPL 要求任何使用/修改/衍生之 GPL 类库的的软件必须采用 GPL 协议不同。LGPL 允许商业软件通过类库引用 (link) 方式使用 LGPL 类库而不需要开源商业软件的代码。这使得采用 LGPL 协议的开源代码可以被商业软件作为类库引用并发布和销售。

但是如果修改 LGPL 协议的代码或者衍生，则所有修改的代码，涉及修改部分的额外代码和衍生的代码都必须采用 LGPL 协议。因此 LGPL 协议的开源代码很适合作为第三方类库被商业软件引用，但不适合希望以 LGPL 协议代码为基础，通过修改和衍生的方式做二次开发的商业软件采用。

GPL/LGPL 都保障原作者的知识产权，避免有人利用开源代码复制并开发类似的产品。

11.1.7 Close Source

相对于 Open Soure，Close Source 仅推出可执行的二进制程序，程序的核心是封闭的，优点是有专人维护，不需要去更改它。缺点则是灵活度大打折扣，用户无法变更该程序成为自己想要的样式，此外若有木马程序或者安全漏洞，将会花上相当长的一段时间来除错，这也是所谓专利软件（copyright）常见的软件销售方式。

代表的授权模式有：

（1）Freeware：不同于 Free software，Freeware 为"免费软件"而非"自由软件"。虽然它是免费的软件，但是不见得要公布其源代码，要看发布者的意见。此外目前很多"免费软件"的程序很多都有小问题，例如假借免费软件的名义，窃取用户资料，所以"来路不明的软件请勿安装"。

（2）Shareware：共享软件与免费软件有点类似的是，Shareware 在使用初期也是免费的，但是到了所谓的"试用期限"之后，就必须要选择"付费后继续使用"或者"将它移除"。通常，这些共享软件都会自行编写失效程序，在试用期限之后就无法使用该软件。

11.2 Usage

11.2.1 Desktop

The popularity of Linux on standard desktop computers and laptops has been increasing over the years. Currently most distributions include a graphical user environment, with the two most popular environments being GNOME (which can utilize additional shells such as the default GNOME Shell and Ubuntu Unity), and the KDE Plasma Desktop.

There is no “one” Linux desktop, but rather there is a pool of free and open-source software from which desktop environments and Linux distributions select components with which they construct a GUI implementing some more or less strict design guide. GNOME, for example, has its Human interface guidelines as a design guide, which gives the Human-Machine Interface an important role, not just when doing the graphical design, but also when looking at people with disabilities, and even when looking at security.

The collaborative nature of free software development allows distributed teams to perform language localization of some Linux distributions for use in locales where localizing proprietary systems would not be cost-effective. For example the Sinhalese language version of the Knoppix distribution was available significantly before Microsoft Windows XP was translated to Sinhalese. In this case the Lanka Linux User Group played a major part in developing the localized system by combining the knowledge of university professors, linguists, and local developers.

[!h]

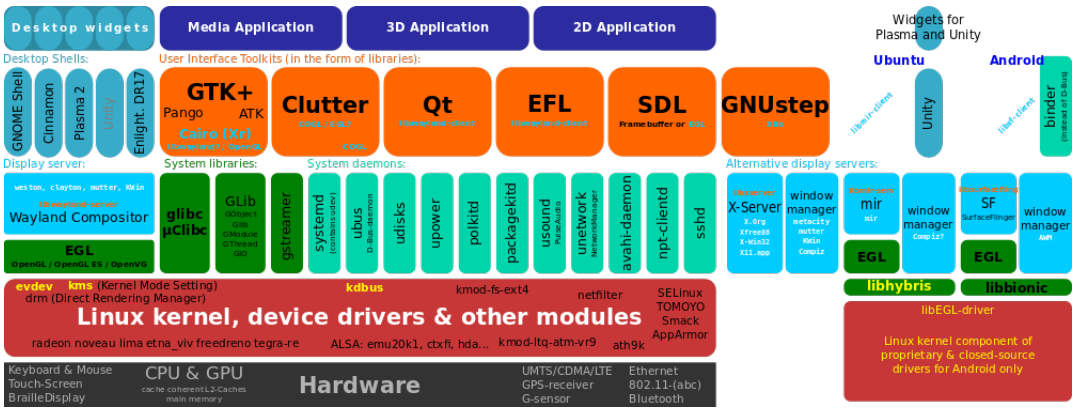


图 11.1: Visible software components of the Linux desktop stack are including the display server, widget engines, and some of the more widespread widget toolkits. There are also invisible components, including D-Bus and PulseAudio.

The performance of Linux on the desktop has been a controversial topic; for example in 2007 Con Kolivas accused the Linux community of favoring performance on servers. He quit Linux kernel development because he was frustrated with this lack of focus on the desktop, and then gave a ”tell all” interview on the topic. Since then a significant amount of development has been undertaken in an effort to improve the desktop experience. Projects such as Upstart and systemd aim for a faster boot time.

Many popular applications are available for a wide variety of operating systems. For example Mozilla Firefox, OpenOffice.org/LibreOffice and Blender have downloadable versions for all major operating systems. Furthermore, some applications were initially developed for Linux, such as Pidgin, and GIMP, and were ported to other operating systems including Windows and Mac OS X due to their popularity. In addition, a growing number of proprietary desktop applications are also supported on Linux, such as Autodesk Maya, Softimage XSI and Apple Shake in the high-end field of animation and visual effects; see the List of proprietary software for Linux for more details. There are also several companies that have ported their own or other companies’ games to Linux, with Linux also being a supported platform on both the popular Steam and Desura digital distribution services.

Many other types of applications available for Microsoft Windows and Mac OS X are also available for Linux. Commonly, either a free software application will exist which does the functions of an application found on another operating system, or that application will have a version that works on Linux, such as with Skype and some video games.

Furthermore, the Wine project provides a Windows compatibility layer to run unmodified Windows applications on Linux. CrossOver is a proprietary solution based on the open source Wine project that supports running Windows versions of Microsoft Office, Intuit applications such as Quicken and QuickBooks, Adobe Photoshop versions through CS2, and many popular games such as World of Warcraft and Team Fortress 2. In other cases,

where there is no Linux port of some software in areas such as desktop publishing and professional audio, there is equivalent software available on Linux.

Besides externally visible components, such as X window managers, a non-obvious but quite central role have the programs hosted by freedesktop.org, such as D-Bus or PulseAudio; both big desktop enironments (GNOME and KDE) include them, each offering graphical front-ends written using the corresponding toolkit (GTK+ or Qt). A display server is another component, which for the longest time has been communicating in the X11 display server protocol with its clients; prominent software talking X11 includes the X.Org Server and Xlib. Frustration over the cumbersome X11 core protocol, and especially over its numerous extensions, has led to the creation of a new display server protocol, Wayland.

Installing, updating and removing software in Linux is typically done through the use of package managers such as the Synaptic Package Manager, PackageKit, and Yum Extender. While most major Linux distributions have extensive repositories, often containing tens of thousands of packages, not all the software that can run on Linux is available from the official repositories. Alternatively, users can install packages from unofficial repositories, download pre-compiled packages directly from websites, or compile the source code by themselves. All these methods come with different degrees of difficulty; compiling the source code is in general considered a challenging process for new Linux users, but it is hardly needed in modern distributions and is not a method specific to Linux.

11.2.2 Servers

Linux distributions have long been used as server operating systems, and have risen to prominence in that area; Netcraft reported in September 2006 that eight of the ten most reliable internet hosting companies ran Linux distributions on their web servers. Since June 2008, Linux distributions represented five of the top ten, FreeBSD three of ten, and Microsoft two of ten;[88] since February 2010, Linux distributions represented six of the top ten, FreeBSD two of ten, and Microsoft one of ten.

Linux distributions are the cornerstone of the LAMP server-software combination (Linux, Apache, MariaDB/MySQL, Perl/PHP/Python) which has achieved popularity among developers, and which is one of the more common platforms for website hosting.

Linux distributions have become increasingly popular on mainframes in the last decade partly due to pricing and the open-source model. In December 2009, computer giant IBM reported that it would predominantly market and sell mainframe-based Enterprise Linux Server.

Linux distributions are also commonly used as operating systems for supercomputers: as of June 2013, more than 95% of the world's 500 fastest supercomputers run some variant of Linux, including all the 44 fastest. Linux was also selected as the operating system for the world's most powerful supercomputer, IBM's Sequoia, which became operational in 2011.

11.2.3 Embedded devices

Due to its low cost and ease of customization, Linux is often used in embedded systems.

In the mobile sub-sector of the Telecommunications equipment sector there are three major platforms based on a more or less modified version of the Linux kernel: mer, Tizen and Android. Android has become the dominant mobile operating system for smartphones, during the second quarter of 2013, 79.3% of smartphones sold worldwide used Android. Cell phones and PDAs running Linux on open-source platforms became more common from 2007; examples include the Nokia N810, Openmoko's Neo1973, and the Motorola ROKR E8. Continuing the trend, Palm (later acquired by HP) produced a new Linux-derived operating system, webOS, which is built into its new line of Palm Pre smartphones.

In the non-mobile telecommunications equipment sector, the majority of Customer-premises equipment-

[!ht]

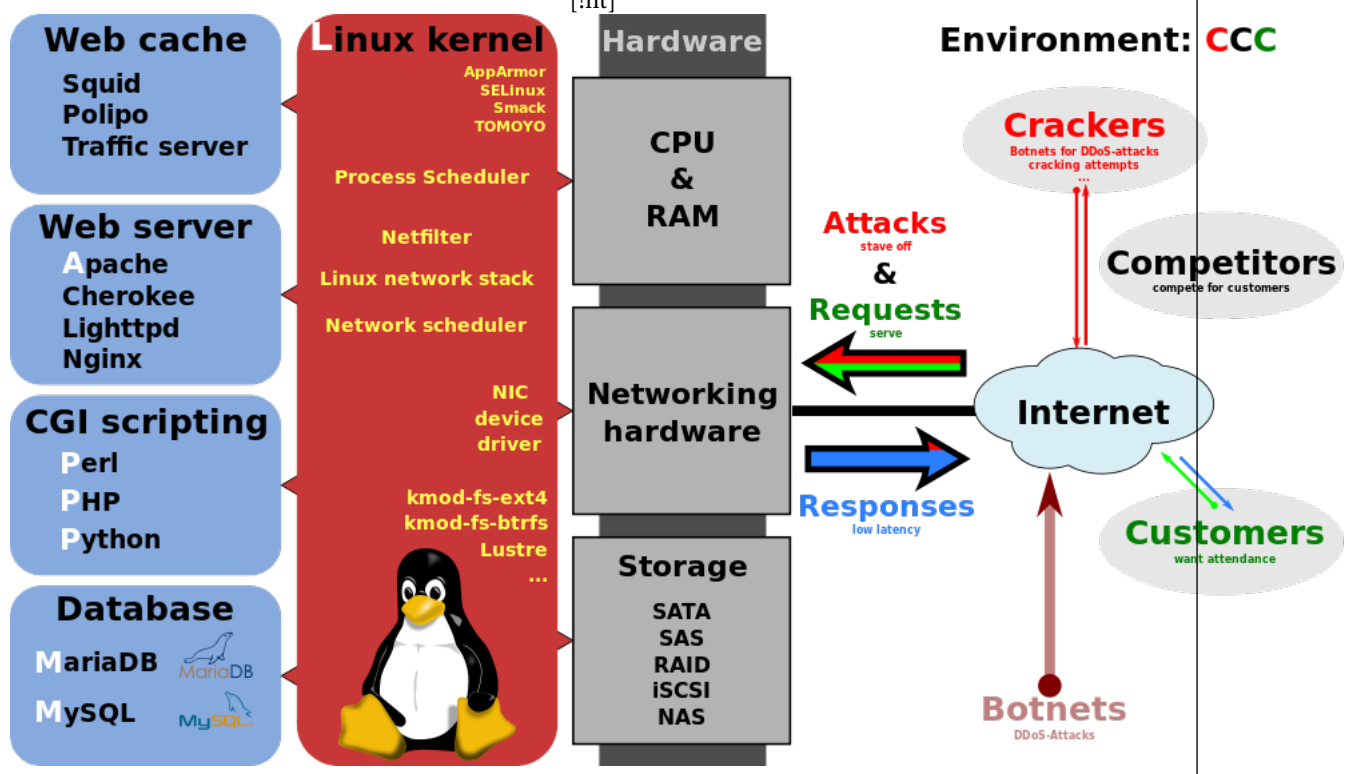


图 11.2: Broad overview of the LAMP software bundle, displayed here together with Squid. A high-performance and high-availability web server solution providing security in a hostile environment.

hardware runs some Linux-based operating system. OpenWrt is a community driven example upon which many of the OEM firmwares are based.

The popular TiVo digital video recorder also uses a customized Linux, as do several network firewalls and routers from such makers as Cisco/Linksys. The Korg OASYS, the Korg KRONOS, the Yamaha Motif XS/Motif XF music workstations, Yamaha S90XS/S70XS, Yamaha MOX6/MOX8 synthesizers, Yamaha Motif-Rack XS tone generator module, and Roland RD-700GX digital piano also run Linux. Linux is also used in stage lighting control systems, such as the WholeHogIII console.

Embedded and real time segments are very vast categories with different subcategories like Automotive (e.g. IVI, Engine control unit, GENIVI Alliance), Avionics, Health, Medical Equipment, Consumer Electronics, Intelligent Homes, Telecommunications and a couple more. The aggregated information above could be very different for each subcategory taken separately. The embedded segment is the largest segment in term of numbers of units.

Part II

Foundation

第 12 章

Overview

从打开电源到开始操作，计算机的启动是一个非常复杂的过程。[8]

启动计算机又称为 boot，boot 原来的意思是靴子，这里的 boot 是 bootstrap（鞋带）的缩写，它来自一句谚语：

```
"pull oneself up by one's bootstraps"
```

字面意思是” 拽着鞋带把自己拉起来”，这当然是不可能的事情。最早的时候，工程师们用它来比喻，计算机启动是一个很矛盾的过程：

```
必须先运行程序，然后计算机才能启动，但是计算机不启动就无法运行程序！
```

早期启动计算机时必须想尽各种办法，把一小段程序装进内存，然后计算机才能正常运行。

12.1 BIOS

上个世纪 70 年代初，“只读内存”（read-only memory，缩写为 ROM）发明，开机程序被刷入 ROM 芯片。

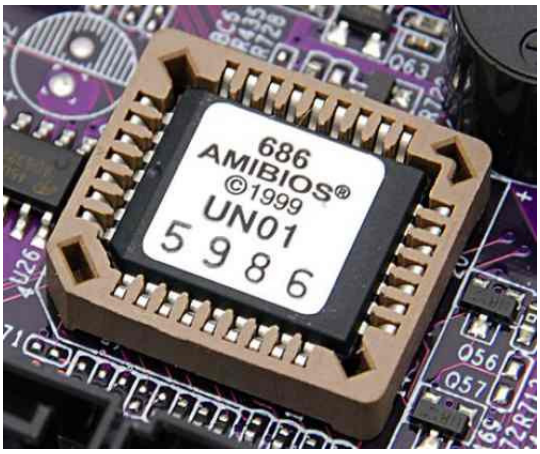


图 12.1: BIOS

这块芯片里的程序叫做 “基本输入/输出系统”（Basic Input/Output System），简称为 BIOS。BIOS 是一种用于 IBM PC 兼容机上的固件接口，是操作系统输入输出管理系统的一部分。

可以将 BIOS 视为是一个永久地记录在 ROM 中的一个软件，也是个人电脑启动时加载的第一个软件。计算机通电或被重置(reset)时，第一件事就是读取 BIOS，处理器第一条指令的地址会被定位到 BIOS 的存储器中，让初始化程序开始运行。

早期的 BIOS 芯片确实是”只读”的，里面的内容是用一种烧录器写入的，一旦写入就不能更改，除非更换芯片。现在的主机板都使用 Flash EPROM 芯片来存储系统 BIOS，里面的内容可通过使用主板厂商提供的擦写程序擦除后重新写入，这样就给用户升级 BIOS 提供了极大的方便。

在 PC 启动的过程中，BIOS 担负著初始化硬件，执行系统各部分的的自检，以及启动引导程序或装载在内存的操作系统的责任。此外，BIOS 还向操作系统提供一些系统参数。系统硬件的变化是由 BIOS 隐藏，程序使用 BIOS 服务而不是直接访问硬件。现代操作系统会忽略 BIOS 提供的抽象层并直接访问硬件组件。

当计算机的电源打开，BIOS 就会由主板上的闪存(flash memory)运行，并将芯片组和存储器子系统初始化。BIOS 会把自己从闪存中解压缩到系统的主存，然后从主存开始运行。PC 的 BIOS 代码也包含诊断功能，以保证某些重要硬件组件(比如 CPU、内存、磁盘设备、键盘、输出输入端口等)可以正常运作且正确地初始化。几乎所有的 BIOS 都可以选择性地运行 CMOS 存储器的设置程序，也就是保存 BIOS 会访问的用户自定义设置数据(时间、日期、硬盘细节等)。

现代的 BIOS 可以配置 PnP(Plug and Play，即插即用)设备并定义出可启动的设备顺序，从而可以让用户选择由哪个设备启动计算机，例如光盘驱动器、硬盘、软盘和 USB 闪存盘等。有些 BIOS 系统允许用户可以选择要加载哪个操作系统(例如从其他硬盘加载操作系统)，虽然这项功能通常是由第二阶段的开机管理程序(Bootloader)来处理。

12.1.1 POST

BIOS 的功能由两部分组成，分别是 POST 码和 Runtime 服务。POST 阶段完成后它将从存储器中被清除，而 Runtime 服务会被一直保留，用于目标操作系统的启动。

开机后，BIOS 程序执行系统完整性检查。首先检查计算机硬件能否满足运行的基本条件，这叫做“开机自我检测”(Power-On Self-Test，缩写为 POST)，主要负责检测系统外围关键设备(如 CPU、内存、显卡、I/O、键盘鼠标等)是否正常。例如，最常见的是内存松动的情况，BIOS 自检阶段会报错，系统就无法启动起来。

1. 开机系统重置 REST 启动 CPU。
2. CPU 指向 BIOS 自我测试的地址 FFFFOH 并打开 CPU 运行第一个指令。
3. CPU 内部暂存器的测试。
4. CMOS 146818 SRAM 检查。
5. ROM BIOS 检查码测试。
6. 8254 计时/计数器测试。
7. 8237 DMA 控制器测试。
8. 74612 页暂存器测试。
9. REFRESH 刷新电路测试。
10. 8042 键盘控制器测试。
11. DRAM 64KB 基本存储器测试。
12. CPU 保护模式的测试。
13. 8259 中断控制器的测试。
14. CMOS 146818 电力及检查码检查。
15. DRAM IMB 以上存储器检查。
16. 显卡测试。
17. NMI 强制中断测试。
18. 8254 计时/计数器声音电路测试。

19. 8254 计时/计数器计时测试。
20. CPU 保护模式 SHUT DOWN 测试。
21. CPU 回至实模式 (REAL MODE)。
22. 键盘鼠标测试。
23. 8042 键盘控制器测试。
24. 8259 中断控制器 IRQ0 至 IRQ18 创建。
25. 磁盘驱动器及界面测试。
26. 设置并行打印机及串行 RS232 的界面。
27. 检查 CMOS IC 时间、日期。
28. 检查完成

开机自检结果会显示在固件可以控制的输出接口，像显示器[1]、LED、打印机等等设备上。如果硬件出现问题，主板会发出不同含义的蜂鸣，启动中止。

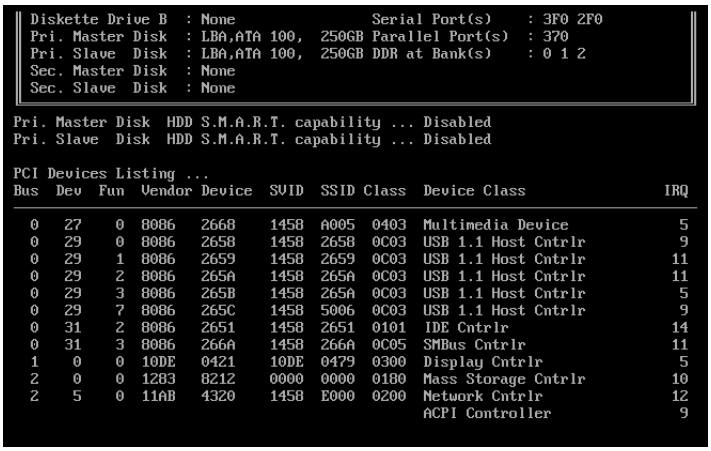


图 12.2: 硬件自检

计算机系统中可以包含多个 BIOS 固件芯片，其中开机 BIOS 主要是包含访问基本硬件组件（例如键盘或软盘驱动器）的代码。额外的适配器（例如 SCSI/SATA 硬盘适配器、网络适配器、显卡等）也会包含他们自己的 BIOS，补充或取代系统 BIOS 代码中有关这些硬件的部份。

为了在开机时找到这些存储器映射的扩充只读存储器，PC BIOS 会扫描物理内存，从 0xC0000 到 0xF0000 的 2KB 边界中查找 0x55 0xaa 记号，接在其后的是一个比特，表示有多少个扩充只读存储器的 512 位区块占据真实存储器空间。接着 BIOS 马上跳跃到指向由扩充只读存储器所接管的地址，以及利用 BIOS 服务来提供用户设置接口，注册中断矢量服务供开机后的应用程序使用，或者显示诊断的信息。

确切地说扩展卡上的 ROM 不能称之为 BIOS。它只是一个程序片段，用来初始化自身所在的扩展卡。

12.1.2 Boot Sequence

硬件自检完成后，BIOS 便会执行一段小程序用来枚举本地设备并对其初始化，这一步主要是根据我们在 BIOS 中设置的系统启动顺序来搜索用于启动系统的驱动器，例如硬盘、光盘、U 盘、软盘和网络等。

BIOS 中有一个外部储存设备的排序，排在前面的设备就是优先转交控制权的设备，这种排序叫做“启动顺序”（Boot Sequence），可以在 BIOS 中“设定启动顺序”。

[1]如果没有显示器，我们可以通过 POST CARD 来完成上面的测试工作。

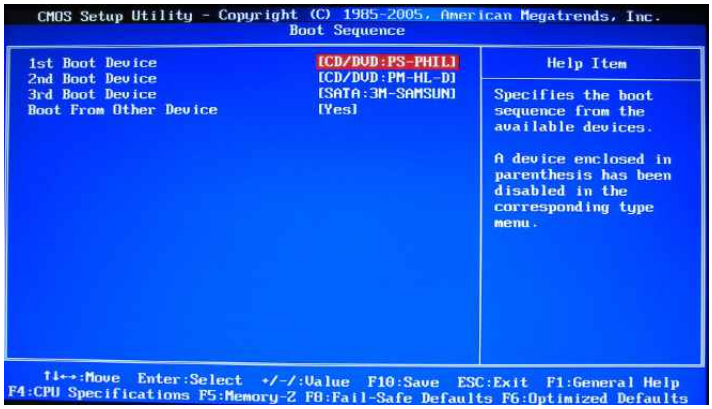


图 12.3: 设定启动顺序

这时，BIOS 需要知道，“下一阶段的启动程序” 具体存放在哪一个设备，然后 BIOS 就把控制权转交给下一阶段的启动程序（Bootloader）。

接下来，计算机读取启动设备的第一个扇区，也就是读取最前面的 512 个字节。如果这 512 个字节的最后两个字节是 0x55 和 0xAA，表明这个设备可以用于启动；如果不是，表明设备不能用于启动，控制权于是被转交给 “启动顺序” 中的下一个设备。

以硬盘启动为例，BIOS 此时去读取硬盘驱动器的第一个扇区，然后执行里面的代码。硬盘的 0 柱面、0 磁头、1 扇区称为主引导扇区，“主引导记录”（Master Boot Record，缩写为 MBR）就位于其第一块扇区内。实际上，这里 BIOS 并不关心启动设备第一个扇区中是什么内容，它只是负责读取该扇区内容并执行。

至此，BIOS 的任务就完成了，此后将系统启动的控制权移交到 MBR 部分的代码。

12.2 UEFI

统一可扩展固件接口（Unified Extensible Firmware Interface，UEFI[6]）是一种替代 BIOS 的个人计算机系统规格，用来定义操作系统与系统固件之间的软件界面。

UEFI 和 BIOS 这两个技术都在计算机启动的时候发出第一个命令指示，并使得操作系统能够被顺利加载。UEFI 负责加电自检（POST）、连系操作系统以及提供连接操作系统与硬件的接口。

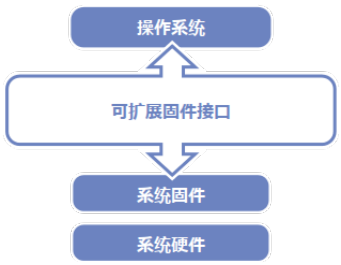


图 12.4: 可扩展固件接口在软件层次中的位置

UEFI 原名本为可扩展固件接口（Extensible Firmware Interface），最初是由英特尔开发并于 2002 年 12 月发布其订定的版本 -1.1 版。英特尔于 2005 年将此规范格式交由 UEFI 论坛来推广与发展，UEFI 论坛于 2007 年 1 月 7 日发布 2.1 版本的规格，其中较 1.1 版本增加与改进了加密编码（cryptography）、网络认证（network authentication）与用户接口架构（User Interface Architecture）。

2009 年 5 月 9 日，UEFI 论坛发布 2.3 版本，最新的公开的版本是 2.3.1。

12.2.1 EFI

EFI 和 BIOS 二者显著的区别就是 EFI 是用模块化，C 语言风格的参数堆栈传递方式，动态链接的形式构建的系统，较 BIOS 而言更易于实现，容错和纠错特性更强，缩短了系统研发的时间。EFI 运行于 32 位或 64 位模式，乃至未来增强的处理器模式下，突破传统 16 位代码的寻址能力，达到处理器的最大寻址。

EFI 利用加载 EFI 驱动的形式来识别和操作硬件，不同于 BIOS 利用挂载真实模式中断的方式增加硬件功能。后者必须将一段类似于驱动的 16 位代码，放置在固定的 0x000C0000 至 0x000DFFFF 之间存储区中，运行这段代码的初始化部分，它将挂载实模式下约定的中断矢量向其他程序提供服务。例如，VGA 图形及文本输出中断（INT 10h），磁盘访问中断服务（INT 13h）等等。由于这段存储空间有限（128KB），BIOS 对于所需放置的驱动代码大小超过空间大小的情况无能为力。

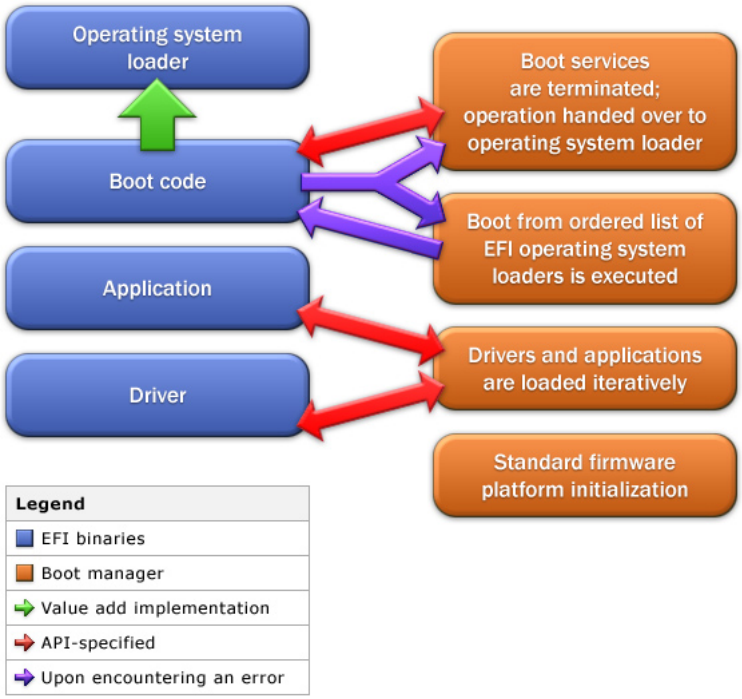


图 12.5: EFI boot manager and EFI drivers

另外，BIOS 的硬件服务程序都以 16 位代码的形式存在，这就给运行于增强模式的操作系统访问其服务造成了困难。因此 BIOS 提供的服务在现实中只能提供给操作系统引导程序或 MS-DOS 类操作系统使用。而 EFI 系统下的驱动并不是由可以直接运行在 CPU 上的代码组成的，而是用 EFI Byte Code 编写而成的。这是一组专用于 EFI 驱动的虚拟机器语言，必须在 EFI 驱动运行环境（Driver Execution Environment，或 DXE）下被解释运行。这就保证了充分的向下兼容性，打个比方说，一个带有 EFI 驱动的扩展设备，既可以将其安装在安腾处理器的系统中，也可以安装于支持 EFI 的新 PC 系统中，而它的 EFI 驱动不需要重新编写。这样就无需对系统升级带来的兼容性因素作任何考虑。

由于 EFI 驱动开发简单，所有的 PC 部件提供商都可以参与，情形非常类似于现代操作系统的开发模式。基于 EFI 的驱动模型可以使 EFI 系统接触到所有的硬件功能，在操作系统运行以前浏览万维网站不再是天方夜谭，甚至实现起来也非常简单。这对基于传统 BIOS 的系统来说是件不可能的任务，在 BIOS 中添加几个简单的 USB 设备支持都曾使很多 BIOS 设计师痛苦万分，更何况除了添加对无数网络硬件的支持外，还得凭空构建一个 16 位模式下的 TCP/IP 协议栈。

一般认为，EFI 由以下几个部分组成：

1. Pre-EFI 初始化模块
2. EFI 驱动执行环境

3. EFI 驱动程序
4. 兼容性支持模块 (CSM)
5. EFI 高层应用
6. GUID 磁盘分区表 (GPT)

EFI 在概念上非常类似于一个低阶的操作系统，并且具有操控所有硬件资源的能力，因此 EFI 将有可能代替现代的操作系统。事实上，EFI 的缔造者们在第一版规范出台时就将 EFI 的能力限制于不足以威胁操作系统的统治地位。

- 首先，它只是硬件和预启动软件间的接口规范；
- 其次，EFI 环境下不提供中断的机制，也就是说每个 EFI 驱动程序必须用轮询 (polling) 的方式来检查硬件状态，并且需要以解释的方式运行，较操作系统下的机械码驱动效率更低；
- 再则，EFI 系统不提供复杂的缓存器保护功能，它只具备简单的缓存器管理机制，具体来说就是指运行在 x86 处理器的段保护模式下，以最大寻址能力为限把缓存器分为一个平坦的段 (Segment)，所有的程序都有权限访问任何一段位置，并不提供真实的保护服务。

当 EFI 所有组件加载完毕时，系统可以开启一个类似于操作系统 Shell 的命令解释环境，在这里，用户可以调入执行任何 EFI 应用程序，这些程序可以是硬件检测及除错软件，引导管理，设置软件，操作系统引导软件等等。

理论上来说，对于 EFI 应用程序的功能并没有任何限制，任何人都可以编写这类软件，并且效果较以前 MS-DOS 下的软件更华丽，功能更强大。一旦引导软件将控制权交给操作系统，所有用于引导的服务代码将全部停止工作。

在实现中，EFI 初始化模块和驱动执行环境通常被集成在一个只读存储器中。Pre-EFI 初始化程序在系统开机的时候最先得到执行，它负责最初的 CPU，主桥及存储器的初始化工作，紧接着载入 EFI 驱动执行环境 (DXE)。当 DXE 被载入运行时，系统便具有了枚举并加载其他 EFI 驱动的能力。在基于 PCI 架构的系统中，各 PCI 桥及 PCI 适配器的 EFI 驱动会被相继加载及初始化；这时，系统进而枚举并加载各桥接器及适配器后面的各种总线及设备驱动程序，周而复始，直到最后一个设备的驱动程序被成功加载。正因如此，EFI 驱动程序可以放置于系统的任何位置，只要能保证它可以按顺序被正确枚举。例如一个具 PCI 总线接口的 ATAPI 大容量存储适配器，其 EFI 驱动程序一般会放置在这个设备的符合 PCI 规范的扩展只读存储器 (PCI Expansion ROM) 中，当 PCI 总线驱动被加载完毕，并开始枚举其子设备时，这个存储适配器旋即被正确识别并加载它的驱动程序。部分 EFI 驱动程序还可以放置在某个磁盘的 EFI 专用分区中，只要这些驱动不是用于加载这个磁盘的驱动的必要部件。

在 EFI 规范中，一种突破传统 MBR 磁盘分区结构限制的 GUID 磁盘分区系统 (GPT) 被引入，磁盘的分区数不再受限制 (在 MBR 结构下，只能存在 4 个主分区)，另外 EFI/UEFI+GUID 结合还可以支持 2.1 TB 以上硬盘[2]，并且分区类型将由 GUID 来表示。在众多的分区类型中，EFI 系统分区可以被 EFI 系统访问，用于存放部分驱动和应用程序。很多人担心这将会导致新的安全性因素，因为 EFI 系统比传统的 BIOS 更易于受到计算机病毒的攻击，当一部分 EFI 驱动程序被破坏时，系统有可能面临无法引导的情况。实际上，系统引导所依赖的 EFI 驱动部分通常都不会存放在 EFI 的 GUID 分区中，即使分区中的驱动程序遭到破坏，也可以用简单的方法得到恢复，这与操作系统下的驱动程序的存储习惯是一致的。CSM 是在 x86 平台 EFI 系统中的一个特殊的模块，它将为不具备 EFI 引导能力的操作系统提供类似于传统 BIOS 的系统服务。

12.3 MBR

BIOS 搜索并加载执行 MBR Bootloader 程序后，接下来 BIOS 把控制权转交给排在第一位的存储设备。

在 ROM BIOS 检查结束时，接下来 BIOS 会按照 “启动顺序” 来搜索并执行系统中第一个启动设备

[2]有测试显示，3TB 硬盘使用 MBR，并且安装 Windows 6.x 64 位系统，只能识别到 2.1TB。

的第一个物理扇区(sector)中的启动程序(Bootloader)[3]。Bootloader 程序使用分区信息来确定哪个分区是可引导的(通常是第一个主分区)并尝试从该分区引导。

简单的 Bootloader 的虚拟汇编码,如其后的八个指令:

0. 将 P 暂存器的值设为 8;
1. 检查纸带 (paper tape) 读取器,是否已经可以进行读取;
2. 如果还不能进行读取,跳至 1;
3. 从纸带读取器,读取一 byte 至累加器;
4. 如为带子结尾,跳至 8;
5. 将暂存器的值,存储至 P 暂存器中的数值所指定的地址;
6. 增加 P 暂存器的值;
7. 跳至 1。

硬盘的第一个物理扇区称为主引导记录(Master Boot Record, MBR[5]),又叫做主引导扇区或主引导块,它是计算机开机后访问硬盘时所必须要读取的首个扇区。

主引导扇区的读取流程如下:

1. BIOS 加电自检 (POST)。BIOS 执行内存地址为 FFFF:0000H 处的跳转指令,跳转到固化在 ROM 中的自检程序处,对系统硬件 (包括内存) 进行检查。
2. 读取主引导记录 (MBR)。当 BIOS 检查到硬件正常并与 CMOS 中的设置相符后,按照 CMOS 中对启动设备的设置顺序检测可用的启动设备。BIOS 将相应启动设备的第一个扇区 (也就是 MBR 扇区) 读入内存地址为 0000:7C00H 处并跳转执行。
3. 检查 0000:7CFEH-0000:7CFFH(MBR 的结束标志位) 是否等于 55AAH,若不等于则转去尝试其他启动设备,如果没有启动设备满足要求则显示 "NO ROM BASIC" 然后死机。
4. 当检测到有启动设备满足要求后,BIOS 将控制权交给相应启动设备。启动设备的 MBR 将自己复制到 0000:0600H 处,然后继续执行。
5. 根据 MBR 中的引导代码启动引导程序。

事实上,BIOS 不仅检查 0000:7CFEH-0000:7CFFH(MBR 的结束标志位) 是否等于 55AAH,往往还对磁盘是否有写保护、主引导扇区中是否存在活动分区等进行检查。如果发现磁盘有写保护,则显示磁盘写保护出错信息;如果发现磁盘中不存在活动分区,则显示类似如下的信息"Remove disk or other media Press any key to restart"。

对于硬盘而言,MBR 在硬盘上的三维地址为:

$$(\text{柱面},\text{磁头},\text{扇区})=(0,0,1)$$

而一个扇区可能的字节数为 128×2^n (n=0,1,2,3),大多数情况下取 n=2,因此一个扇区(sector)的大小为 512 字节。

MBR 记录着硬盘本身的相关信息以及硬盘各个分区的大小及位置信息。具体而言,MBR 由三个部分组成,分别是主引导程序 (Bootloader)、硬盘分区表 DPT(Disk Partition table) 和硬盘有效标志 (55AA)。MBR 一共占用 512bytes,其结构图如下所示:

其中:

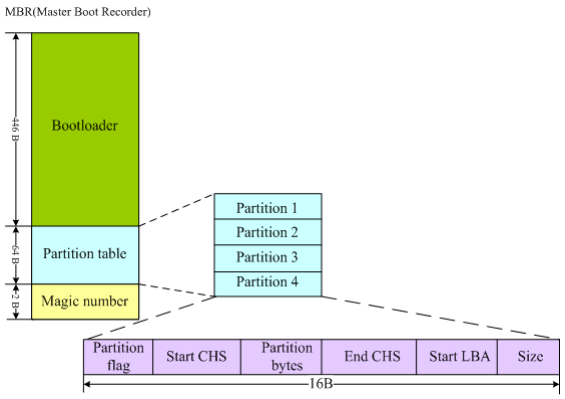
1. Bootloader(第 1 ～ 446 字节):调用操作系统的机器码。
2. DPT(第 447 ～ 510 字节):硬盘分区表(Disk Partition table)。
3. MBR 验证信息(第 511 ～ 512 字节):主引导记录签名(0x55 和 0xAA)[4]。

在硬盘的主引导记录最开头是第一阶段引导代码,其中的硬盘主引导程序的主要作用是检查分区表是否正确并且在系统硬件完成自检以后将控制权交给硬盘上的引导程序(如 GNU GRUB[5]),该程序使用分区信息来确定哪个分区是可引导的并尝试从该分区引导。

[3]BIOS 可以通过硬件的 INT 13 中断功能来读取 MBR。也就是说,只要 BIOS 可以检测到硬盘,就可以通过 INT 13 硬件中断来读取该磁盘的第一个扇区内的 MBR,这样 Bootloader 就能够被执行。

[4]结束标志字 55 和 AA(偏移 1FEH －偏移 1FFH)是主引导扇区的最后两个字节,是检验主引导记录是否有效的标志。

[5]通常情况下,诸如 LILO、GUN GRUB 这些常见的引导程序都直接安装在 MBR 中。



在深入讨论主引导扇区内部结构的时候，有时也将其开头的 446 字节内容特指为“主引导记录”(MBR)，其后是 4 个 16 字节的“磁盘分区表”(DPT)以及 2 字节的结束标志(55AA)，因此在使用“主引导记录”(MBR)这个术语的时候，需要根据具体情况判断其到底是指整个主引导扇区，还是主引导扇区的前 446 字节。

标准 MBR 结构

地址			描述		长度 (字节)
Hex	Oct	Dec			
0000	0000	0	代码区		**440** (最大 446)
01B8	0670	440	选用磁盘标志		4
01BC	0674	444	一般为空值; 0x0000		2
01BE	0676	446	**标准 MBR 分区表规划** (四个16 byte的主分区表入口)		**64**
01FE	0776	510	55h	MBR 有效标志: 0x55AA	**2**
01FF	0777	511	AAh		
MBR, 总大小: 446 + 64 + 2 =					**512**

图 12.6: 标准 MBR 结构

MBR 的主要作用是告诉计算机到硬盘的哪一个位置去找操作系统，分区信息(或分区表)存储在该扇区的末尾处。如果 MBR 受到破坏，硬盘上的基本数据结构信息将会丢失，需要用繁琐的方式试探性的重建数据结构信息后才可能重新访问原先的数据。主引导扇区内的信息可以通过任何一种基于某种操作系统的分区工具软件[6]写入，但和某种操作系统没有特定的关系，即只要创建了有效的主引导记录就可以引导任意一种操作系统(操作系统是创建在高级格式化的硬盘分区之上，是和一定的文件系统相联系的)。

12.3.1 DPT

DPT 的作用是将硬盘分成若干个区。硬盘分区有很多好处，比如考虑到每个区可以安装不同的操作系统，此时 MBR 就必须知道该将控制权转交给哪个分区。

DPT 的长度只有 64 个字节 (偏移 01BEH ～偏移 01FDH)，里面又分成四项，每项 16 个字节。所以，一个硬盘最多只能分四个一级分区，又叫做“主分区”(Primary)。每个主分区的 16 个字节，由 6 个部分组成：

[6]例如，MS-DOS Fdisk 实用工具通常仅当不存在主引导记录时才会更新主引导记录 (MBR)。

1. 第 1 个字节：如果为 0x80，就表示该主分区是激活分区，控制权要转交给这个分区。四个主分区里面只能有一个是激活的。
2. 第 2-4 个字节：主分区第一个扇区的物理位置(柱面、磁头、扇区号等等)。
3. 第 5 个字节：主分区类型。
4. 第 6-8 个字节：主分区最后一个扇区的物理位置。
5. 第 9-12 字节：该主分区第一个扇区的逻辑地址。
6. 第 13-16 字节：主分区的扇区总数。

根据 16 字节分区表的结构：当前分区的扇区数用 4 个字节表示，前面各分区扇区数的总和也是 4 个字节，而 $2^{32} \times 512$⊠2 199 023 255 552Byte。

最后的四个字节（“主分区的扇区总数”），决定了这个主分区的长度。也就是说，一个主分区的扇区总数最多不超过 2 的 32 次方。

Table 12.1: 硬盘分区结构信息

偏移	长度 (字节)	意义	备注
00H	1	分区状态	00–> 非活动分区；
			80–> 活动分区[7]；
			其它数值没有意义
01H	1		分区起始磁头号 (HEAD)，用到全部 8 位
02H	2		分区起始扇区号 (SECTOR)，占据 02H 的位 0 － 5；
			该分区的起始磁柱号 (CYLINDER)，占据 02H 的位 6 － 7 和 03H 的全部 8 位
04H	1		文件系统标志位
05H	1		分区结束磁头号 (HEAD)，用到全部 8 位
06H	2		分区结束扇区号 (SECTOR)，占据 06H 的位 0 － 5；
			该分区的结束磁柱号 (CYLINDER)，占据 06H 的位 6 － 7 和 07H 的全部 8 位
08H	4		分区起始相对扇区号
0CH	4		分区总的扇区数

下面是一个例子，如果某一分区在硬盘分区表的信息如下：

80 01 01 00 0B FE BF FC 3F 00 00 00 7E 86 BB 00

则我们可以看到，

- 最前面的 “80” 是一个分区的激活标志，表示系统可引导；
- “01 01 00” 表示分区开始的磁头号为 1，开始的扇区号为 1，开始的柱面号为 0；
- “0B” 表示分区的系统类型是 FAT32，其他比较常用的有 04（FAT16）、07（NTFS）；
- “FE BF FC” 表示分区结束的磁头号为 254，分区结束的扇区号为 63、分区结束的柱面号为 764；
- “3F 00 00 00” 表示首扇区的相对扇区号为 63；
- “7E 86 BB 00” 表示总扇区数为 12289662。

对于现代大于 8.4G 的硬盘，CHS 已经无法表示，BIOS 使用 LBA 模式，对于超出的部分，CHS 值通常设为 FEFFFF，并加以忽略，直接使用 08-0f 的 4 字节相对值，再进行内部转换。

一个硬盘的分区个数还要受到分区大小的限制，因为硬盘是按照柱面分区的：一个分区至少要占一个柱面。但有一点需要注意，由于现在的硬盘结构已经和老式硬盘有了很大区别，其寻址结构也不

[7]对于一个操作系统而言，系统分区设为活动分区并不是必须的，这主要视引导程序而定，如果使用的引导程序是 Grub4Dos，MBR 中的引导代码仅仅按照分区的顺序依次探测第二阶段引导器 grldr 的位置，并运行第一个探测到的 grldr 文件。

再是 CHS 寻址，所以这里的柱面大小不同于相关软件显示的柱面大小。对于物理结构上有 n 个面的硬盘，其分区空间的最小值为：

$$n \times 扇区/磁道 \times 512字节$$

从主引导记录的结构可以知道，它仅仅包含一个 64 个字节的硬盘分区表。由于每个分区信息需要 16 个字节，所以对于采用 MBR 型分区结构的硬盘，最多只能识别 4 个主要分区（Primary partition）。所以对于一个采用此种分区结构的硬盘来说，想要得到 4 个以上的主要分区是不可能的，这时就需要引出扩展分区了。扩展分区也是主要分区的一种，但它与主分区的不同在于理论上可以划分为无数个逻辑分区。

扩展分区中逻辑驱动器的引导记录是链式的。每一个逻辑分区都有一个和 MBR 结构类似的扩展引导记录 (EBR)，其分区表的第一项指向该逻辑分区本身的引导扇区，第二项指向下一个逻辑驱动器的 EBR，分区表第三、第四项没有用到。

Windows 系统默认情况下，一般都是只划分一个主分区给系统，剩余的部分全部划入扩展分区。这里有下面几点需要注意：

- 在 MBR 分区表中最多 4 个主分区或者 3 个主分区＋ 1 个扩展分区，也就是说扩展分区只能有一个，然后可以再细分为多个逻辑分区。
- 在 Linux 系统中，硬盘分区命名为 sda1 － sda4 或者 hda1 － hda4（其中 a 表示硬盘编号可能是 a、b、c 等等）。在 MBR 硬盘中，分区号 1 － 4 是主分区（或者扩展分区），逻辑分区号只能从 5 开始。
- 在 MBR 分区表中，一个分区最大的容量为 2T，且每个分区的起始柱面必须在这个 disk 的前 2T 内。你有一个 3T 的硬盘，根据要求你至少要把它划分为 2 个分区，且最后一个分区的起始扇区要位于硬盘的前 2T 空间内。如果硬盘太大则必须改用 GPT。

如果每个扇区为 512 个字节，就意味着单个分区最大不超过 2TB。再考虑到扇区的逻辑地址也是 32 位，所以单个硬盘可利用的空间最大也不超过 2TB。如果想使用更大的硬盘，只有 2 个方法：一是提高每个扇区的字节数，二是增加扇区总数。

与支持最大卷为 2 TB（Terabytes）并且每个磁盘最多有 4 个主分区（或 3 个主分区，1 个扩展分区和无限制的逻辑驱动器）的 MBR 磁盘分区的样式相比，GPT 磁盘分区样式支持最大卷为 128 EB（Exabytes）并且每磁盘的分区数没有上限，只受到操作系统限制（由于分区表本身需要占用一定空间，最初规划硬盘分区时，留给分区表的空间决定了最多可以有多少个分区，IA-64 版 Windows 限制最多有 128 个分区，这也是 EFI 标准规定的分区表的最小尺寸）。与 MBR 分区的磁盘不同，至关重要的平台操作数据位于分区，而不是位于非分区或隐藏扇区。另外，GPT 分区磁盘有备份分区表来提高分区数据结构的完整性。

12.4 GPT

全局唯一标识分区表（GUID Partition Table，GPT[4]）是一个实体硬盘的分区表的结构布局的标准。

GPT 是可扩展固件接口（EFI）标准的一部分，被用于替代 BIOS 系统中的 32bits 来存储逻辑块地址和大小信息的主引导记录（MBR）分区表。对于那些扇区为 512 字节的磁盘，MBR 分区表不支持容量大于 2.2TB（2.2×10^{12} 字节）的分区。然而，一些硬盘制造商（诸如希捷和西部数据）注意到了这个局限性，并且将他们的容量较大的磁盘升级到了 4KB 的扇区，这意味着 MBR 的有效容量上限提升到了 16 TiB。这个看似“正确的”解决方案，在临时地降低了人们对改进磁盘分配表的需求的同时，也给市场带来了关于在有较大的块 (block) 的设备上从 BIOS 启动时，如何最佳的划分磁盘分区的困惑。

GPT 分配 64bits 给逻辑块地址，因而使得最大分区大小在 $2^{64}-1$ 个扇区成为了可能。对于每个扇区大小为 512 字节的磁盘，那意味着可以有 9.4ZB(9.4 x 10^{21} 字节) 或 8 ZiB 个 512 字节（9,444,732,965,739,290,426,880 字节或 18,446,744,073,709,551,615（$2^{64}-1$）个扇区 x 512（29）字节每扇区）。

在 MBR 硬盘中，分区信息直接存储于主引导记录（MBR）中（主引导记录中还存储着系统的引导

程序)。但在 GPT 硬盘中,分区表的位置信息储存在 GPT 头中。但出于兼容性考虑,硬盘的第一个扇区仍然用作 MBR,之后才是 GPT 头。

跟现代的 MBR 一样,GPT 也使用逻辑区块地址(LBA)取代了早期的 CHS 寻址方式。传统 MBR 信息存储于 LBA 0,GPT 头存储于 LBA 1,接下来才是分区表本身。64 位 Windows 操作系统使用 16,384 字节(或 32 扇区)作为 GPT 分区表,接下来的 LBA 34 是硬盘上第一个分区的开始。

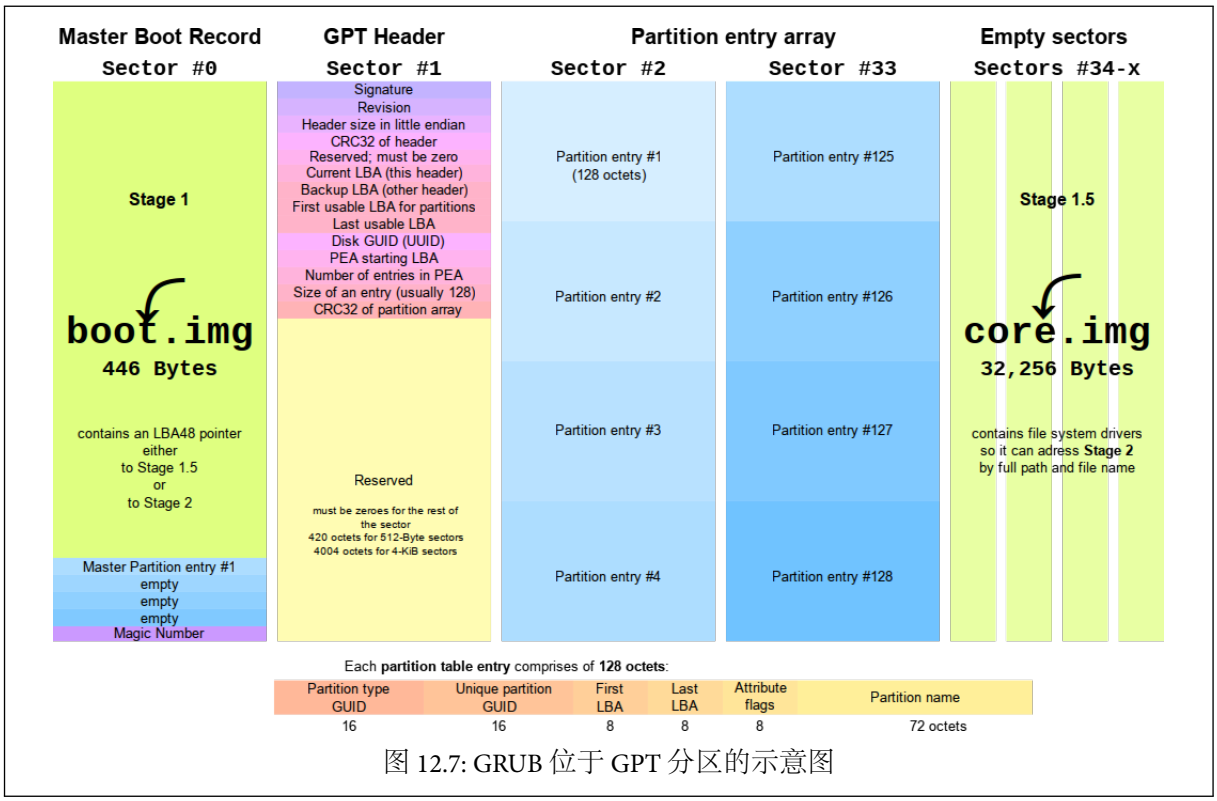


图 12.7: GRUB 位于 GPT 分区的示意图

苹果公司曾经警告说:“不要假定所有设备的块大小都是 512 字节。”一些现代的存储设备如固态硬盘可能使用 1024 字节的块,一些磁光盘(MO)可能使用 2048 字节的扇区(但是磁光盘通常是不进行分区的)。一些硬盘生产商在计划生产 4096 字节一个扇区的硬盘,但截至 2010 年初,这种新硬盘使用固件对操作系统伪装成 512 字节一个扇区。

为了减少分区表损坏的风险,GPT 在硬盘最后保存了一份分区表的副本。

对于不标准的 MBR/GPT 混合硬盘,不同的系统中的实现有些不一致。除非另加说明,操作系统在处理混合硬盘时优先读取 GPT 分区表

12.4.1 Legacy MBR

在 GPT 分区表的最开头,处于兼容性考虑仍然存储了一份传统的 MBR,用来防止不支持 GPT 的硬盘管理工具错误识别并破坏硬盘中的数据,这个 MBR 也叫做保护 MBR。在支持从 GPT 启动的操作系统中,这里也用于存储第一阶段的启动代码。

在这个 MBR 中,只有一个标识为 0xEE 的分区,以此来表示这块硬盘使用 GPT 分区表。不能识别 GPT 硬盘的操作系统通常会识别出一个未知类型的分区,并且拒绝对硬盘进行操作,除非用户特别要求删除这个分区。这就避免了意外删除分区的危险。另外,能够识别 GPT 分区表的操作系统会检查保护 MBR 中的分区表,如果分区类型不是 0xEE 或者 MBR 分区表中有多个项,也会拒绝对硬盘进行操作。

在使用 MBR/GPT 混合分区表的硬盘中,这部分存储了 GPT 分区表的一部分分区(通常是前四个分区),可以使不支持从 GPT 启动的操作系统从这个 MBR 启动,启动后只能操作 MBR 分区表中的分区。如 Boot Camp 就是使用这种方式启动 Windows。

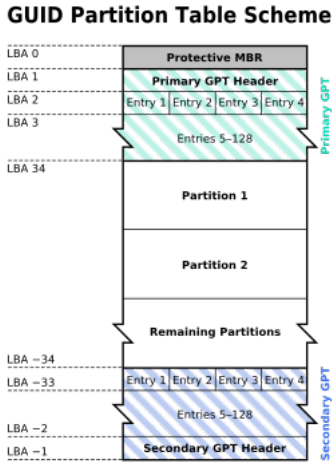


图 12.8: GPT 分区表的结构。此例中，每个逻辑块（LBA）为 512 字节，每个分区的记录为 128 字节。负数的 LBA 地址表示从最后的块开始倒数，–1 表示最后一个块。

12.4.2 Partition Table Header

分区表头定义了硬盘的可用空间以及组成分区表的项的大小和数量。在使用 64 位 Windows Server 2003 的机器上，最多可以创建 128 个分区，即分区表中保留了 128 个项，其中每个都是 128 字节。（EFI 标准要求分区表最小要有 16,384 字节，即 128 个分区项的大小）

分区表头还记录了这块硬盘的 GUID，记录了分区表头本身的位置和大小（位置总是在 LBA 1）以及备份分区表头和分区表的位置和大小（在硬盘的最后）。它还储存着它本身和分区表的 CRC32 校验。固件、引导程序和操作系统在启动时可以根据这个校验值来判断分区表是否出错，如果出错了，可以使用软件从硬盘最后的备份 GPT 中恢复整个分区表，如果备份 GPT 也校验错误，硬盘将不可使用。所以 GPT 硬盘的分区表不可以直接使用 16 进制编辑器修改。

Table 12.2: 分区表头的格式

起始字节	长度	内容
0	8 字节	签名（“EFI PART”, 45 46 49 20 50 41 52 54）
8	4 字节	修订（在 1.0 版中，值是 00 00 01 00）
12	4 字节	分区表头的大小（单位是字节，通常是 92 字节，即 5C 00 00 00）
16	4 字节	分区表头（第 0 – 91 字节）的 CRC32 校验，在计算时，把这个字段作为 0 处理，需要计算出分区串行的 CRC32 校验后再计算本字段
20	4 字节	保留，必须是 0
24	8 字节	当前 LBA（这个分区表头的位置）
32	8 字节	备份 LBA（另一个分区表头的位置）
40	8 字节	第一个可用于分区的 LBA（主分区表的最后一个 LBA + 1）
48	8 字节	最后一个可用于分区的 LBA（备份分区表的第一个 LBA – 1）
56	16 字节	硬盘 GUID（在类 UNIX 系统中也叫 UUID）
72	8 字节	分区表项的起始 LBA（在主分区表中是 2）
80	4 字节	分区表项的数量
84	4 字节	一个分区表项的大小（通常是 128）
88	4 字节	分区串行的 CRC32 校验

起始字节	长度	内容
92	*	保留，剩余的字节必须是 0(对于 512 字节 LBA 的硬盘即是 420 个字节)

主分区表和备份分区表的头分别位于硬盘的第二个扇区(LBA 1)以及硬盘的最后一个扇区。备份分区表头中的信息是关于备份分区表的。

12.4.3 Partition Entries

GPT 分区表使用简单而直接的方式表示分区。一个分区表项的前 16 字节是分区类型 GUID。例如，EFI 系统分区的 GUID 类型是 {C12A7328-F81F-11D2-BA4B-00A0C93EC93B}。接下来的 16 字节是该分区唯一的 GUID(这个 GUID 指的是该分区本身，而之前的 GUID 指的是该分区的类型)。再接下来是分区起始和末尾的 64 位 LBA 编号，以及分区的名字和属性。

Table 12.3: GPT 分区表项的格式

起始字节	长度	内容
0	16 字节	分区类型 GUID
16	16 字节	分区 GUID
32	8 字节	起始 LBA(小端序)
40	8 字节	末尾 LBA
48	8 字节	属性标签(例如 60 表示“只读”)
56	72 字节	分区名（可以包括 36 个 UTF-16 (小端序)字符)

12.4.4 Partition type GUIDs

操作系统	分区类型	GUID
(None)	未使用	00000000-0000-0000-0000-000000000000
	MBR 分区表	024DEE41-33E7-11D3-9D69-0008C781F39F
	EFI 系统分区	C12A7328-F81F-11D2-BA4B-00A0C93EC93B
	BIOS 引导分区	21686148-6449-6E6F-744E-656564454649
Windows	微软保留分区	E3C9E316-0B5C-4DB8-817D-F92DF00215AE
	基本数据分区[8]	EBD0A0A2-B9E5-4433-87C0-68B6B72699C7
	逻辑软盘管理工具元数据分区	5808C8AA-7E8F-42E0-85D2-E1E90434CFB3
	逻辑软盘管理工具数据分区	AF9B60A0-1431-4F62-BC68-3311714A69AD
	Windows 恢复环境	DE94BBA4-06D1-4D40-A16A-BFD50179D6AC
	IBM 通用并行文件系统 (GPFS) 分区	37AFFC90-EF7D-4e96-91C3-2D7AE055B174
HP-UX	数据分区	75894C1E-3AEB-11D3-B7C1-7B03A0000000
	服务分区	E2A1E728-32E3-11D6-A682-7B03A0000000
Linux	数据分区	EBD0A0A2-B9E5-4433-87C0-68B6B72699C7
	RAID 分区	A19D880F-05FC-4D3B-A006-743F0F84911E
	交换分区	0657FD6D-A4AB-43C4-84E5-0933C84B4F4F
	逻辑卷管理器 (LVM) 分区	E6D6D379-F507-44C2-A23C-238F2A3DF928

[8] Linux 和 Windows 的数据分区使用相同的 GUID。

操作系统	分区类型	GUID
	保留	8DA63339-0007-60C0-C436-083AC8230908
FreeBSD	启动分区	83BD6B9D-7F41-11DC-BE0B-001560B84F0F
	数据分区	516E7CB4-6ECF-11D6-8FF8-00022D09712B
	交换分区	516E7CB5-6ECF-11D6-8FF8-00022D09712B
	UFS 分区	516E7CB6-6ECF-11D6-8FF8-00022D09712B
	Vinum volume manager 分区	516E7CB8-6ECF-11D6-8FF8-00022D09712B
	ZFS 分区	516E7CBA-6ECF-11D6-8FF8-00022D09712B
Mac OS X	HFS(HFS+) 分区	48465300-0000-11AA-AA11-00306543ECAC
	苹果公司 UFS	55465300-0000-11AA-AA11-00306543ECAC
	ZFS	6A898CC3-1DD2-11B2-99A6-080020736631
	苹果 RAID 分区	52414944-0000-11AA-AA11-00306543ECAC
	苹果 RAID 分区, 下线	52414944-5F4F-11AA-AA11-00306543ECAC
	苹果启动分区	426F6F74-0000-11AA-AA11-00306543ECAC
	Apple Label	4C616265-6C00-11AA-AA11-00306543ECAC
	Apple TV 恢复分区	5265636F-7665-11AA-AA11-00306543ECAC
Solaris	启动分区	6A82CB45-1DD2-11B2-99A6-080020736631
	根分区	6A85CF4D-1DD2-11B2-99A6-080020736631
	交换分区	6A87C46F-1DD2-11B2-99A6-080020736631
	备份分区	6A8B642B-1DD2-11B2-99A6-080020736631
	/usr 分区[9]	6A898CC3-1DD2-11B2-99A6-080020736631
	/var 分区	6A8EF2E9-1DD2-11B2-99A6-080020736631
	/home 分区	6A90BA39-1DD2-11B2-99A6-080020736631
	备用扇区	6A9283A5-1DD2-11B2-99A6-080020736631
		6A945A3B-1DD2-11B2-99A6-080020736631
	保留分区	6A9630D1-1DD2-11B2-99A6-080020736631
		6A980767-1DD2-11B2-99A6-080020736631
		6A96237F-1DD2-11B2-99A6-080020736631
		6A8D2AC7-1DD2-11B2-99A6-080020736631
NetBSD[10]	交换分区	49F48D32-B10E-11DC-B99B-0019D1879648
	FFS 分区	49F48D5A-B10E-11DC-B99B-0019D1879648
	LFS 分区	49F48D82-B10E-11DC-B99B-0019D1879648
	RAID 分区	49F48DAA-B10E-11DC-B99B-0019D1879648
	concatenated 分区	2DB519C4-B10F-11DC-B99B-0019D1879648
	加密分区	2DB519EC-B10F-11DC-B99B-0019D1879648

本表中的 GUID 使用小端序表示。例如, EFI 系统分区的 GUID 在这里写成 C12A7328-F81F-11D2-BA4B-00A0C93EC93B 但实际上它对应的 16 字节的串行是 28 73 2A C1 1F F8 D2 11 BA 4B 00 A0 C9 3E C9 3B ——只有前 3 部分的字节序被交换了。

[9]Solaris 系统中/usr 分区的 GUID 在 Mac OS X 上被用作普通的 ZFS 分区。
[10]NetBSD 的 GUID 在单独定义之前曾经使用过 FreeBSD 的 GUID。

12.5 Bootloader

这时,计算机的控制权就要转交给硬盘的某个分区了,这里又分成三种情况。

1. 情况 A:卷引导记录
   前面提到,四个主分区里面,只有一个是激活的。计算机会读取激活分区的第一个扇区,叫做“卷引导记录”(Volume boot record,缩写为 VBR)。VBR 的主要作用是告诉计算机,操作系统在这个分区里的位置。然后,计算机就会加载操作系统了。
2. 情况 B:扩展分区和逻辑分区
   随着硬盘越来越大,四个主分区已经不够了,需要更多的分区。但分区表只有四项,因此规定有且仅有一个区可以被定义成“扩展分区”(Extended Partition)。所谓“扩展分区”,就是指这个区里面又分成多个区,扩展分区里面的分区称为“逻辑分区”(Logical Partition)。
   - 计算机先读取扩展分区的第一个扇区,叫做“扩展引导记录”(Extended boot record,缩写为 EBR)。它里面也包含一张 64 字节的分区表,但是最多只有两项(也就是两个逻辑分区)。
   - 计算机接着读取第二个逻辑分区的第一个扇区,再从里面的分区表中找到第三个逻辑分区的位置,以此类推,直到某个逻辑分区的分区表只包含它自身为止(即只有一个分区项)。

   因此,扩展分区可以包含无数个逻辑分区。但是,似乎很少通过这种方式启动操作系统。如果操作系统确实安装在扩展分区,一般采用下一种方式启动。
3. 情况 C:启动管理器
   在这种情况下,计算机读取“主引导记录”前面 446 字节的机器码之后,不再把控制权转交给某一个分区,而是运行事先安装的“启动加载器”(bootloader),由用户选择启动哪一个操作系统。

启动程序(Bootloader)也称启动加载器和引导程序,是指位于计算机或其他计算机应用上用于引导操作系统启动的程序。

Bootloader 启动方式及程序视应用机型种类而不同,例如在个人电脑上的引导程序通常分为两部分:第一阶段引导程序位于主引导记录(MBR),用以引导位于某个分区上的第二阶段引导程序,如 NTLDR、GNU GRUB 等。

BIOS 开机完成后,Bootloader 就接手初始化硬件设备、创建存储器空间的映射,以便为操作系统内核准备好正确的软硬件环境,从而可以使操作系统准备好软件执行的环境来加载系统运行所需要的软件信息。

Bootloader 不依赖任何操作系统并且可以改变启动代码,从而可以指定使用哪个内核文件来引导操作系统启动,并实际加载内核到内存中解压缩与执行,此时内核就能够在内存中运行。接下来将会利用内核的功能再次检测所有硬件信息(相对于 BIOS 而言)和加载适当的驱动程序来使周边设备开始运行。也就是说,此时操作系统内核开始接管 BIOS 后面的工作了,这也就是 Bootloader 实现多系统引导的原理。

不过,随着计算机操作系统越来越复杂,不同操作系统的文件格式也不一致,每种操作系统都有自己相应的 Bootloader,因此位于主引导记录的空间已经放不下引导操作系统的代码,于是就有了第二阶段的引导程序,而 MBR 中代码的功能也从直接引导操作系统变为了引导第二阶段的引导程序。

其实,每个文件系统(filesystem 或 partition)都会保留一个引导扇区(boot sector)以提供操作系统安装 Bootloader。通常,操作系统默认都会将 Bootloader 安装到其根目录所在的文件系统的 boot sector 上,因此当安装了 Windows、Linux 多系统后,boot sector、bootloader 与 MBR 的相关性示意图如下:

12.5.1 GRUB

目前 Linux 环境中最流行的启动引导程序是 GRUB(Grand Unified Bootloader),主要用于选择操作系统分区上的不同内核,也可用于向这些内核传递启动参数。

GRUB 非常轻便,而且支持多种可执行格式。除了可适用于支持多启动的操作系统外,还可以通过链式启动功能支持诸如 Windows 和 OS/2 之类的不支持多启动的操作系统。

图 12.9: MBR、boot sector 与操作系统的关系

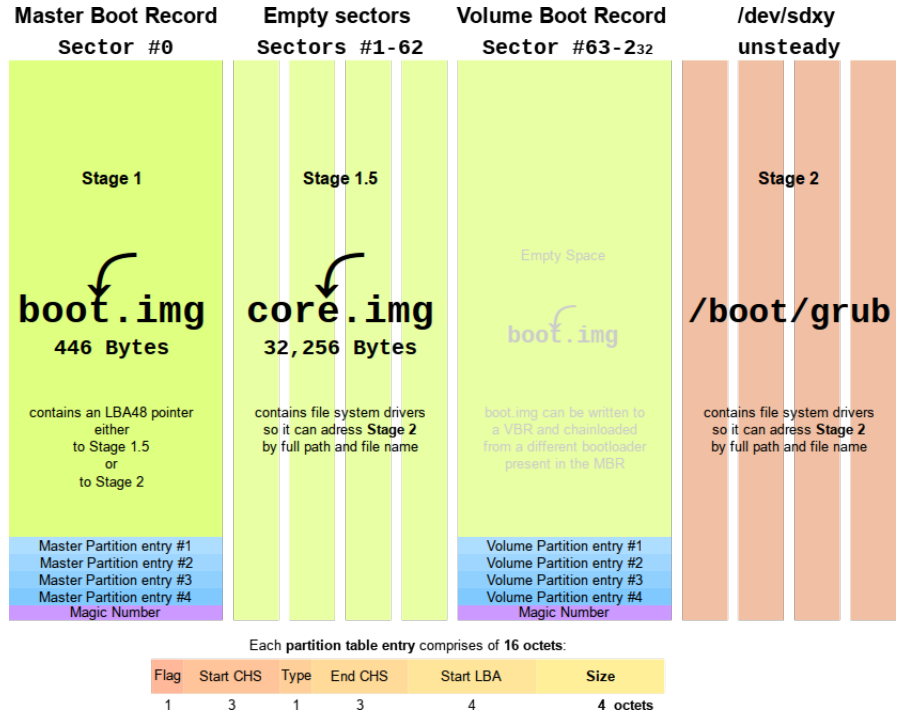


图 12.10: GRUB 位于 MBR 分区的示意图

GRUB 支持所有的 Unix 文件系统，也支持 Windows 适用的 FAT 和 NTFS 文件系统，还支持 LBA 模式。

GRUB 允许用户查看它支持的文件系统里文件的内容，这样就可以在启动时通过 GRUB 加载配置信息并进行修改（如选择不同的内核和 initrd）。为此目的，GRUB 提供了一个简单的类似 Bash 的命令行界面，它允许用户编写新的启动顺序。

与其它启动器不同，GRUB 可以通过 GRUB 提示符直接与用户进行交互。载入操作系统前，在 GRUB 文本模式屏幕下键入 c 键可以进入 GRUB 命令行。在没有作业系统或者有作业系统而没有”menu.lst” 文件的系统上，同样可以进入 GRUB 提示符。通过类似 bash 的命令，GRUB 提示符允许用户手工启动任何操作系统。把合适的命令记录在“menu.lst”文件里，可以自动启动一个操作系统。

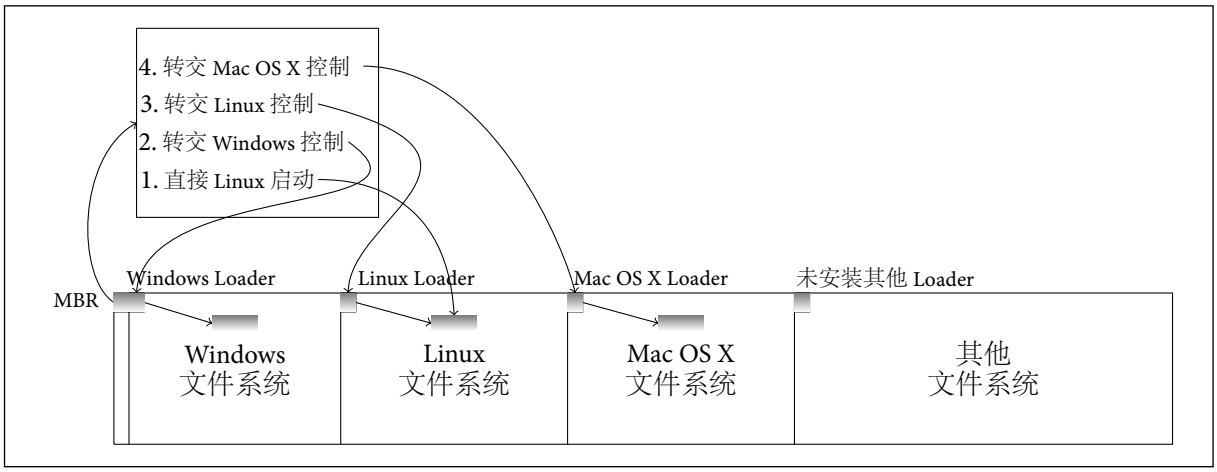
GRUB 拥有丰富的终端命令，在命令行下使用这些命令，用户可以查看硬盘分区的细节，修改分区设置，临时重新映射磁盘顺序，从任何用户定义的配置文件启动，以及查看 GRUB 所支持的文件系统上的其它启动器的配置。因此，即便不知道一台电脑上安装了什么，也可以从外部设备启动一个操作系统。

GRUB 具有多种用户界面，可以利用 GRUB 对图形界面的支持来提供定制的带有背景图案的启动菜单并支持鼠标。另外，通过对 GRUB 的文字界面的设定，可以通过串口实现远程终端启动。

- 提供选择菜单用以提示用户选择不同的启动选项，从而实现多重引导。
- 直接指向可启动的程序区段并执行以加载内核程序来引导操作系统。

- 将引导装载功能转交于其他 boot sector 内的 Boot Loader 来负责。

在 Linux 系统安装时，可以选择是否把 Bootloader 安装到 MBR，这样理论上 MBR 和 boot sector 都会保存一份 Bootloader 程序。而在 Windows 系统在安装时则默认主动将 MBR 和 boot sector 中都安装一份 Bootloader，因此在安装多重操作系统时，MBR 经常会被不同的操作系统的 boot sector 所覆盖。不过，Windows 的 Loader 默认不具有控制权转交的功能，因此也就不能使用 Windows 的 Loader 来加载 Linux 的 Loader。



通过链式启动，一个启动器可以启动另一个启动器。通过 GRUB 可以从 DOS、Windows、Linux、BSD 和 Solaris 等系统启动。如果有多个 Kernel Images 安装在当前系统中时，就可以通过 GRUB 选择哪一个被执行，从而进入不同的内核所控制的系统。如果用户没有任何输入的话，将会加载 GRUB 配置文件中指定的默认 Kernel Image 。

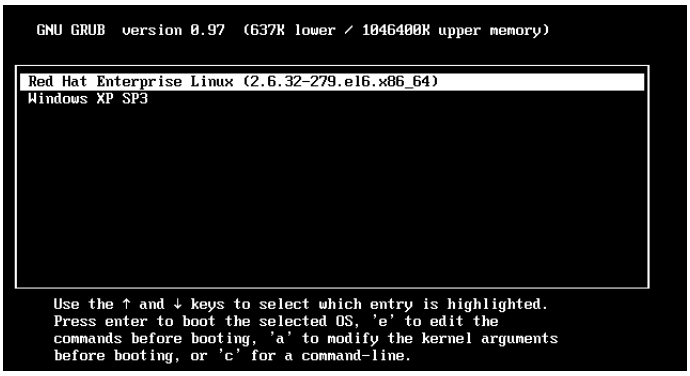


图 12.11: Grub 启动管理器

一旦选择了启动选项，GRUB 把选择的内核载入内存并把控制交给内核。在此步骤中，对于 Windows 之类不支持多启动标准的操作系统，GRUB 也可以通过链式启动把控制传给其它启动器。在这种情况下，其它操作系统的启动程序被 GRUB 保存了下来；与内核不同，其它操作系统如同直接自 MBR 启动。类似 Windows 的启动菜单，也许是另一个启动管理器，它允许在多个不支持多启动的操作系统中做进一步的选择。(在已有 Windows 的系统上面，或者包含多个 Windows 版本的系统上安装现代的 Linux 而不修改原操作系统，即属于这类情况。)

12.5.2 GRUB legacy

GRUB 的步骤 1 包含在 MBR 中。由于受 MBR 的大小限制，步骤一所做的几乎只是装载 GRUB 的下一步骤(存放在硬盘的其它位置)。步骤 1 既可以直接装载步骤 2，也可以装载步骤 1.5：GRUB 的步骤 1.5 包含在 MBR 后面的 30 千字节中。步骤 1.5 载入步骤 2。

当步骤 2 启动后，它将呈现一个界面来让用户选择启动的操作系统。这步通常采用的是图形菜单的形式，如果图形方式不可用或者用户需要更高级的控制，可以使用 GRUB 的命令行提示，通过它，用户可以手工指定启动参数。GRUB 还可以设置超时后自动从某一个内核启动。

12.5.3 GRUB 2

与 GRUB 第一版相似的是,boot.img 像步骤 1 一样在 MBR 或在启动分区中，但是，它可以从任何 LBA48 地址的一个扇区中读取，boot.img 将读取 core.img(产生于 diskboot.img) 的第一个扇区以用来后面读取 core.img 的剩余部分。core.img 正常情况下跟步骤 1.5 储存在同一地方并且有着同样的问题，可是，当它被移动到一个文件系统或一个纯粹的分区时会比在步骤 1.5 移动或删除引起更少的麻烦。一旦完成读取，core.img 会读取默认的配置文件和其他需要的模块。

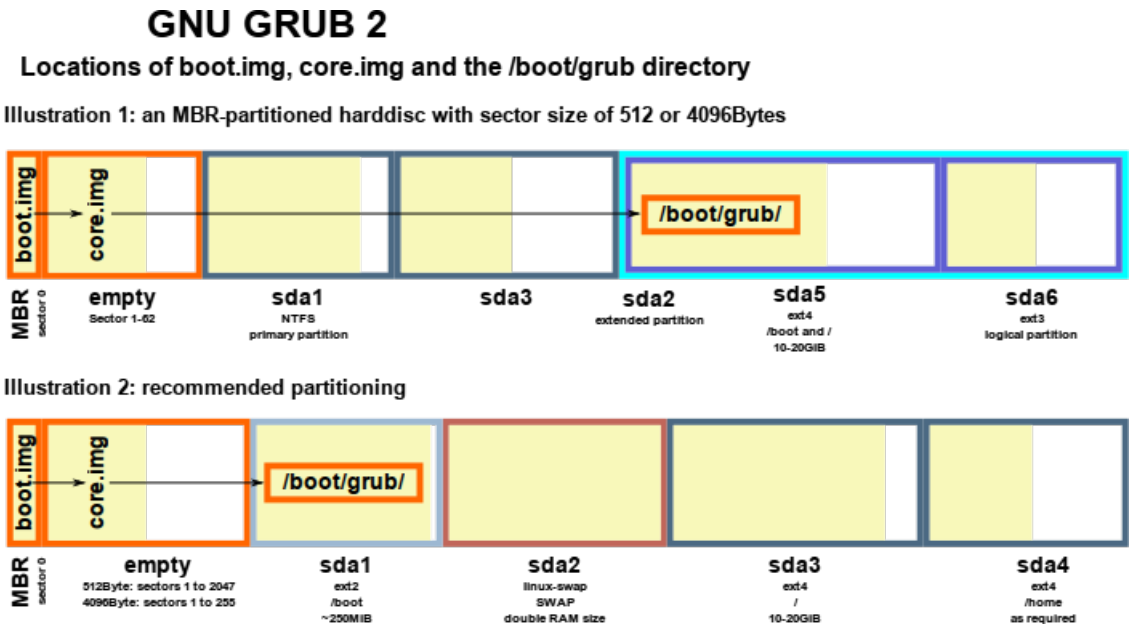


图 12.12: boot.img has the exact size of 446 Bytes and is written to the MBR (sector 0). core.img is written to the empty sectors between the MBR and the first partition, if available (for legacy reasons the first partition starts at sector 63 instead of sector 1, but this is not mandatory). The /boot/grub-directory can be located on an distinct partition, or on the /-partition.

GRUB 的一个重要的特性是安装它不需依附一个操作系统；但是，这种安装需要一个 Linux 副本。由于单独工作，GRUB 实质上是一个微型系统，通过链式启动的方式，它可以启动所有安装的主流操作系统。

与 LILO 不同，修改 GRUB 的配置文件后，不必把 GRUB 重新安装到 MBR 或者某个分区中。

在 Linux 中，“grub-install”命令是用来把 GRUB 的步骤 1 安装到 MBR 或者分区中的。GRUB 的配置文件、步骤 2 以及其它文件必须安装到某个可用的分区中。如果这些文件或者分区不可用，步骤 1 将把用户留在命令行界面。

GRUB 配置文件的文件名和位置随系统的不同而不同，例如在 Debian(GRUB Legacy) 和 OpenSUSE 中，这个文件为/boot/grub/menu.lst，而在 Fedora 和 Gentoo 中为/boot/grub/grub.conf。Fedora、Gentoo Linux 和 Debian(GRUB 2) 使用/boot/grub/grub.conf[11]。

除了硬盘外，GRUB 也可安装到光盘、软盘和闪存盘等移动介质中，这样就可以带起一台无法从硬盘启动的系统。

[11]Fedora 为了兼容文件系统层次结构标准提供了一个从/etc/grub.conf 到/boot/grub/grub.conf 的符号链接

当 Grub 被载入内存执行时，它首先会去解析配置文件/boot/grub/grub.conf (/etc/grub.conf is a link to this)[12]，然后加载内核映像到内存中，并将控制权转交给内核。

```
#boot=/dev/sda
default=0
timeout=15
#splashimage=(hd0,0)/grub/splash.xpm.gz hiddenmenu
serial --unit=0 --speed=115200 --word=8 --parity=no --stop=1
terminal --timeout=10 serial console

title Red Hat Enterprise Linux Server (2.6.17-1.2519.4.21.el5xen)
 root (hd0,0)
 kernel /xen.gz-2.6.17-1.2519.4.21.el5 com1=115200,8n1
 module /vmlinuz-2.6.17-1.2519.4.21.el5xen ro root=/dev/VolGroup00/LogVol00
 module /initrd-2.6.17-1.2519.4.21.el5xen.img
```

也就是说挂载 grub.conf 中指定“root=”的根目录，并把控制权转交给操作系统[13]，而操作系统内核会立即初始化系统中各设备并做相关的配置工作，其中包括 CPU、I/O、存储设备等。

这里，关于 Linux 的设备驱动程序的加载，有一部分驱动程序是直接被编译进内核镜像中，另一部分驱动程序则是以模块的形式放在 initrd(ramdisk) 中。

Linux 内核需要适应多种不同的硬件架构，但是将所有的硬件驱动编入内核又是不实际的，因此实际上 Linux 的内核镜像仅是包含了基本的硬件驱动，在系统安装过程中会检测系统硬件信息，根据安装信息和系统硬件信息将一部分设备驱动写入 initrd 。这样在以后启动系统时，一部分设备驱动就放在 initrd 中来加载。

12.6 Init

操作系统的启动实际上是非常复杂的，内核首先要检测硬件并加载适当的驱动程序，还要调用程序来准备好系统运行的环境来让用户能够顺利的使用计算机。

Bootloader 的最终运行结果都是加载操作系统内核文件，因此当将控制权转交给操作系统后，操作系统的内核首先被载入内存。

以 Linux 系统为例，先载入/boot 目录下面的 kernel。内核加载成功后，第一个运行的程序是/sbin/init。它根据配置文件(Debian 系统是/etc/initab)产生 init 进程。这是 Linux 启动后的第一个进程，pid 进程编号为 1[14]，其他进程都是它的后代。

initrd 的英文含义是 bootloader initialized RAM disk，就是由 bootloader 初始化的内存盘。在 linu2.6 内核启动前，Bootloader 会将存储介质中的 initrd 文件加载到内存，内核启动时会在访问真正的根文件系统前先访问该内存中的 initrd 文件系统。grub 将 initrd 加载到内存里，让后将其中的内容释放到内容中，内核便去执行 initrd 中的 init 脚本，这时内核将控制权交给了 init 文件处理。

在 Bootloader 配置了 initrd 的情况下，内核启动被分成了两个阶段，第一阶段先执行 initrd 文件系统中的 init 来完成加载驱动模块等任务，第二阶段才会执行真正的根文件系统中的 /sbin/init 进程。

在内核完全启动起来，root 文件系统被挂载之前，initrd 被 kernel 当做临时 root 文件系统。当然 initrd 还包含了一些编译好的驱动，这些驱动用来在启动的时候访问硬件。

initramfs 是在 kernel 2.5 中引入的技术，实际上它的含义就是：在内核镜像中附加一个 cpio 包，这个 cpio 包中包含了一个小型的文件系统，当内核启动时，内核将这个 cpio 包解开，并且将其中包含的文件系统释放到 rootfs 中，内核中的一部分初始化代码会放到这个文件系统中，作为用户层进程来执行。这样带来的明显的好处是精简了内核的初始化代码，而且使得内核的初始化过程更容易定制。

init 脚本的内容主要是加载各种存储介质相关的设备驱动程序。当所需的驱动程序加载完后，会

[12] 也有的系统中，GRUB 的配置文件为/boot/grub2/grub.cfg(Fedora)。
[13] 在个人电脑中，Linux 的启动是从 0xFFFF0 地址开始的。
[14] 因为/sbin/init 是 LINUX kernel 执行的第一个程序，所以/sbin/init 的 PID 为 1。

创建一个根设备,然后将根文件系统 rootfs 以只读的方式挂载。这一步结束后,释放未使用的内存,转换到真正的根文件系统上面去,同时运行/sbin/init 程序,执行系统的 1 号进程。此后系统的控制权就全权交给/sbin/init 进程了。

/sbin/init 进程是系统其他所有进程的父进程,当它接管了系统的控制权之后,在加载各项系统服务之前,它首先会去读取/etc/inittab 配置文件来执行相应的脚本进行系统初始化,例如设置键盘、字体,装载模块和设置网络等。

具体而言,/sbin/init 进程会从/etc/inittab 文件中得到默认启动级别,加载系统的各个模块并执行相应级别的程序(比如窗口程序和网络程序),直至执行/bin/login 程序,跳出登录界面,等待用户输入用户名和密码。

当 Linux 系统启动完成后,许多的服务进程也启动了,这些服务程序都放在相应 Linux 系统启动级别的目录中。根据 Linux 的默认启动级别,系统将会执行以下其中一个目录中的服务程序:

- Run level 0 – /etc/rc.d/rc0.d/
- Run level 1 – /etc/rc.d/rc1.d/
- Run level 2 – /etc/rc.d/rc2.d/
- Run level 3 – /etc/rc.d/rc3.d/
- Run level 4 – /etc/rc.d/rc4.d/
- Run level 5 – /etc/rc.d/rc5.d/
- Run level 6 – /etc/rc.d/rc6.d/

注意,在/etc 下面的那些文件有些是连接文件,例如/etc/rc0.d 连接到/etc/rc.d/rc0.d[10]。

第 13 章

Boot

在 BIOS 阶段, 计算机的行为基本上被写死了, 程序员可以做的事情并不多; 但是, 一旦进入操作系统, 程序员几乎可以定制所有方面。

下图显示了典型 Linux 计算机系统启动的 6 个主要阶段[1]:

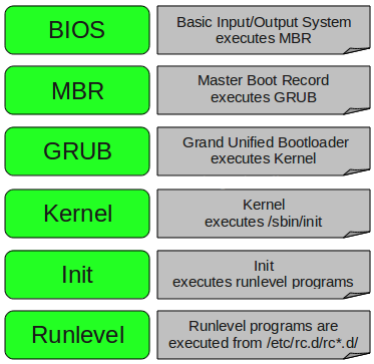


图 13.1: 通电 ->BIOS->MBR->GRUB->Kernel->/sbin/init->runlevel

这里, 针对 Linux 发行版来探讨操作系统接管硬件以后发生的事情, 也就是操作系统的启动流程[9]。

13.1 vmlinuz

内核文件一般放置于/boot 下并被命名为/boot/vmlinuz, 因此操作系统接管硬件以后, 会首先读入/boot 目录下的内核文件。

一般来说, 当创建一个可启动的核心时, 此核心会先经过 zlib 算法压缩, 而在核心内会包含一个相当小的解压缩程序 stub, 当 stub 解压缩核心程序的时候会对 console 视窗印出 “点” 来表示解压缩进度, 而解压缩所花费的时间在开机时间中所占程度来说其实是相当小的, 早期的 bzImage 的发展中对于核心的大小会有所限制(特别是 i386 架构), 在此情况下压缩则是必须的。

vmlinuz[?] 是 vmlinux 经过 gzip 和 objcopy 制作出来的压缩文件。vmlinuz 作为一种统称, 有两种具体的表现形式 zImage 和 bzImage[1]。bzimage 和 zImage 的区别在于本身的大小, 以及加载到内存时的地址不同, 其中 zImage 在 0 ~ 640KB, 而 bzImage 则在 1M 以上的位置。

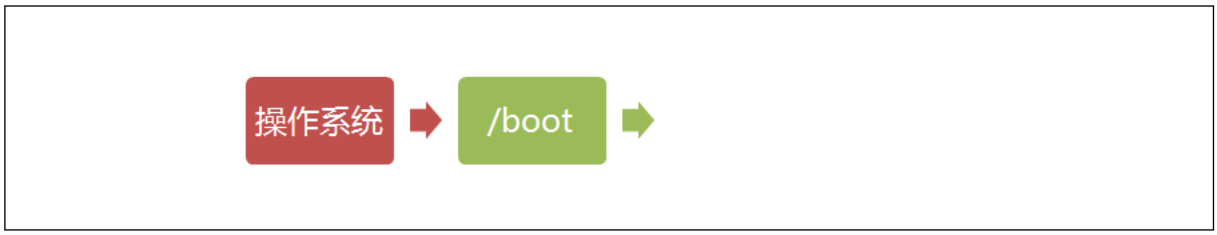
/boot 目录下面大概是这样一些文件:

```
1 $ ls /boot
2 % 对应的内核被编译时选择的功能与模块配置文件
3 config-3.2.0-3-amd64
```

[1]随着 Linux kernel 的成长, 核心的内容日益增加超越了原本的限制大小。bzImage (big zImage) 格式则为了克服此缺点开始发展, 利用将核心切割成不连续的存储器区块来克服大小限制。

bzImage 格式仍然是以 zlib 算法来做压缩, 虽然有一些广泛的误解就是因为以 bz- 为开头, 而让人误以为是使用 bzip2 压缩方式(bzip2 包所带的工具程序通常是以 bz- 为开头的, 例如 bzless, bzcat ...)。

bzImage 文件是一个特殊的格式, 包含了 bootsect.o + setup.o + misc.o + piggy.o 串接, 其中 piggy.o 包含了一个 gzip 格式的 vmlinux 文件(可以参看 arch/i386/boot/下的 compressed/Makefile piggy.o)。




```
config-3.2.0-4-amd64
efi
%引导装载程序GRUB相关的文件目录
grub2
%虚拟文件系统文件
initrd.img-3.2.0-3-amd64
initrd.img-3.2.0-4-amd64
%内核功能放置到内存地址的对应表
System.map-3.2.0-3-amd64
System.map-3.2.0-4-amd64
%对应的内核文件
vmlinuz-3.2.0-3-amd64
vmlinuz-3.2.0-4-amd64
```

Linux 内核可以通过动态加载内核模块，这些内核模块存储在/lib/modules 目录中。

由于内核模块位于根目录下，因此/lib 不能与/放置在不同的分区中，而且在启动的过程中内核必须要挂载根目录才能读取内核模块。为了避免影响到系统文件，在启动过程中根目录/是以只读的方式挂载的。

一般来说，非必要的且可以编译为模块的内核功能，Linux 都会将它们编译成模块，因此 USB、SATA、SCSI 等磁盘设备的驱动程序通常都被编译成模块的形式。

initrd（Boot Loader Initialized RAM Disk）就是由 Bootloader 初始化的内存盘。在 linux 2.6 内核启动前，Bootloader 会将存储介质中的/boot/initrd 文件加载到内存，内核启动时会在访问真正的根文件系统前先访问该内存中的 initrd 文件系统。GRUB 将 initrd 加载到内存里，让后将其中的内容释放到内容中并模拟成一个根目录，内核便去执行 initrd 中的 init 脚本，这时内核将控制权交给了 init 进程。

initrd 实质上是通过 cpio 命令生成的文件，其组成结构类似于：

```
lrwxrwxrwx 1 root root 7 Apr 12 13:08 bin -> usr/bin
drwxr-xr-x 2 root root 4096 Apr 12 13:08 dev
drwxr-xr-x 13 root root 4096 Apr 12 13:08 etc
lrwxrwxrwx 1 root root 23 Apr 12 13:08 init -> usr/lib/systemd/systemd
lrwxrwxrwx 1 root root 7 Apr 12 13:08 lib -> usr/lib
lrwxrwxrwx 1 root root 9 Apr 12 13:08 lib64 -> usr/lib64
drwxr-xr-x 2 root root 4096 Apr 12 13:08 proc
drwxr-xr-x 2 root root 4096 Apr 12 13:08 root
drwxr-xr-x 2 root root 4096 Apr 12 13:08 run
lrwxrwxrwx 1 root root 8 Apr 12 13:08 sbin -> usr/sbin
-rwxr-xr-x 1 root root 3041 Apr 12 13:08 shutdown
drwxr-xr-x 2 root root 4096 Apr 12 13:08 sys
drwxr-xr-x 2 root root 4096 Apr 12 13:08 sysroot
drwxr-xr-x 2 root root 4096 Apr 12 13:08 tmp
drwxr-xr-x 7 root root 4096 Apr 12 13:08 usr
drwxr-xr-x 3 root root 4096 Apr 12 13:08 var
```

通过上述执行文件的内容可以知道 initrd 加载了相关模块并尝试挂载了虚拟文件系统，因此在内核完全启动起来，root 文件系统被挂载之前，initrd 都被 kernel 当做临时 root 文件系统。当然 initrd 还包含了一些编译好的驱动，这些驱动用来在启动的时候访问硬件。

在 Bootloader 配置了 initrd 的情况下，内核启动被分成了两个阶段，第一阶段先执行 initrd 文件

系统中的 init 来完成加载启动过程中需要的内核模块等任务。在第一阶段结束时将完成加载 USB、RAID、LVM 和 SCSI 等文件系统与磁盘接口的驱动程序。接下来，在第二阶段在真正的根文件系统中内核将会调用/sbin/init 程序来开始后续的启动流程。

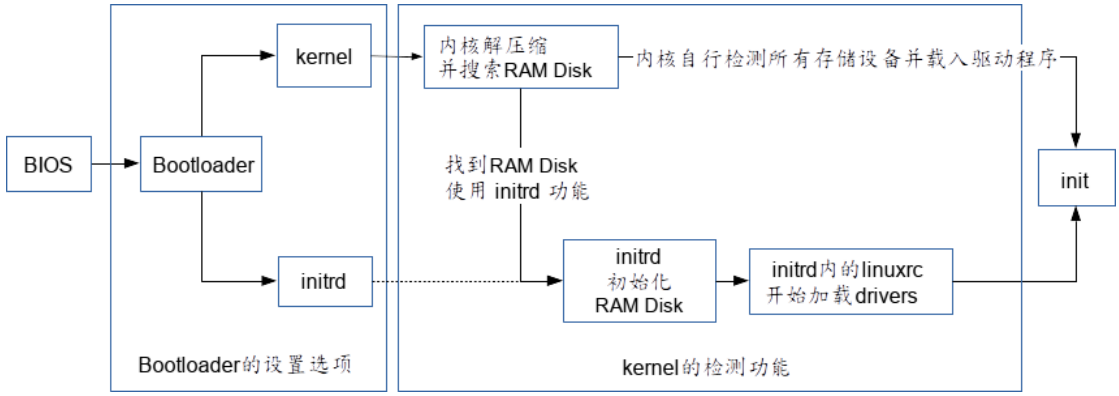


图 13.2: BIOS 与 Bootloader 以及内核加载流程示意图

Bootloader 可以加载 kernel 与 initrd，然后在内存中将 initrd 解压缩并模拟成根目录。该模拟根目录在内存中的文件系统能够提供一个可执行的程序来加载启动过程中所需要的内核模块，接下来 Kernel 就能够借此加载适当的驱动程序(通常是 USB、RAID、LVM 和 SCSI 等文件系统与磁盘接口的驱动程序)，最终释放虚拟文件系统并挂载实际的根目录文件系统，从而能够帮助内核重新调用/sbin/init 来开始后续的正常启动流程。

需要注意的是，initrd 只是应用在无法挂载根目录的情况下，如果 Linux 安装在 IDE 接口的磁盘上，并且使用默认的 Ext2/Ext3 文件系统，那么不需要 initrd 也能够顺利的启动 Linux 操作系统的。

init ramfs 是在 kernel 2.5 中引入的技术，实际上它的含义就是：在内核镜像中附加一个 cpio 包，这个 cpio 包中包含了一个小型的文件系统，当内核启动时，内核将这个 cpio 包解开，并且将其中包含的文件系统释放到 rootfs 中，内核中的一部分初始化代码会放到这个文件系统中，作为用户层进程来执行。这样带来的明显的好处是精简了内核的初始化代码，而且使得内核的初始化过程更容易定制。

13.2 init

内核文件完整加载以后，就可以开始运行第一个程序：/sbin/init。

/sbin/init 是 Unix 和类 Unix 系统中用来产生其它所有进程的程序。作为第一个运行的程序，其进程编号(pid)就是 1，其他所有进程都从它衍生，都是它的子进程。

/sbin/init 以守护进程的方式存在，其作用是系统初始化，具体来说就是指从 init 进程成功启动，一直到系统启动并弹出登录提示之间的过程，所有这些操作都会通过 init 的配置文件——/etc/inittab 来规划，具体包括系统的主机名、网络设置、语系设置、文件系统格式及其他服务的启动等。另外，initab 还确定了操作系统默认的 runlevel(启动执行等级)。

UNIX(例如 System III 和 System V) 系列中 init 的作用和 UNI-like 和 BSD 衍生版本相比，发生了一些变化。

具体来说，BSD init 运行存放于 '/etc/rc' 的初始化 shell 脚本，然后启动基于文本模式的终端 (getty)

或者基于图形界面的终端 (窗口系统)。这里没有运行模式的问题, 因为文件 'rc' 决定了 init 如何执行。这种做法的优点是简单且易于手动编辑, 但如果第三方软件需要在启动过程执行它自身的初始化脚本, 它必须修改已经存在的启动脚本, 一旦这种过程中有一个小错误, 都将导致系统无法正常启动。

现代的 BSD 衍生系统一直支持使用 'rc.local' 文件的方式, 它将在正常启动过程接近最后的时间以子脚本的方式来执行。这样做减少了整个系统无法启动的风险。然后, 第三方软件包可以将它们独立的 start/stop 脚本安装到一个本地的 'rc.d' 目录中 (通常这是由 ports collection/pkgsrc 完成的)。与 SysV 类似, FreeBSD 和 NetBSD 现在默认使用 rc.d, 该目录中所有的用户启动脚本, 都被分成更小的子脚本。rcorder 通常根据在 rc.d 目录中脚本之间的依赖关系来决定脚本的执行顺序。

- Apple Mac OS X 使用 SystemStarter, 用来替代 launchd;
- Ubuntu 使用 Upstart 来完全代替 init;
- Fedora 使用 systemd 来完全替代 init, 并可并行启动服务来减少在 shell 上的系统开销。

作为 Linux 下的一种 init 软件, systemd 的开发目标是提供更优秀的框架以表示系统服务间的依赖关系, 并依此实现系统初始化时服务的并行启动, 同时达到降低 Shell 的系统开销的效果, 最终代替现在常用的 System V 与 BSD 风格 init 程序。

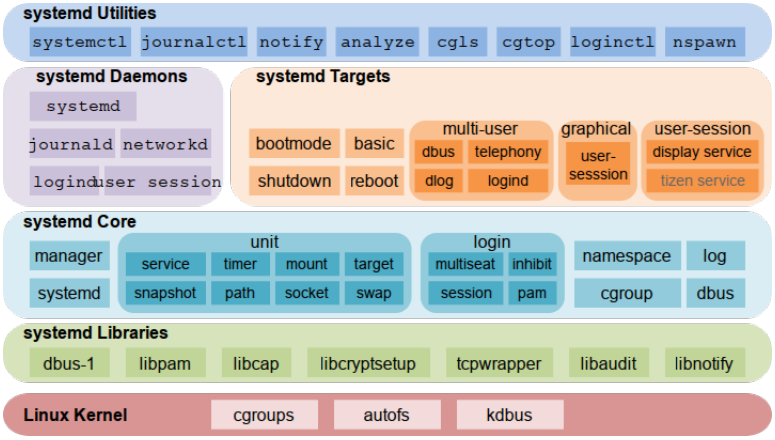


图 13.3: systemd componentos

与多数发行版使用的 System V 风格 init 相比, systemd 采用了以下新技术:

- 采用 Socket 激活式与总线激活式服务, 以提高相互依赖的各服务的并行运行性能;
- 用 cgroups 代替 PID 来追踪进程, 以此即使是两次 fork 之后生成的守护进程也不会脱离 systemd 的控制。

通常在 /etc/inittab 文件中定义了各种运行模式的工作范围, 例如 System V init 会检查 '/etc/inittab' 文件中是否含有 'initdefault' 项, 然后通知 init 系统是否有一个默认运行模式。如果没有默认的运行模式, 那么用户将进入系统控制台来手动决定进入何种运行模式。System V 中运行模式描述了系统各种可能的状态, 通常会有 8 种运行模式, 即运行模式 0 到 6 和 S 或者 s, 其中运行模式 3 为 "保留的" 运行模式。

- 0. 关机
- 1. 单用户模式
- 6. 重启

除了模式 0、1 和 6 之外, 每种 Unix 和 Unix-like 系统对运行模式的定义不太一样, 例如通常模式 5 是多用户图形环境 (X Window System), 通常还包括 X 显示管理器, 然而在 Solaris 操作系统中, 模式 5 被保留用来执行关机和自动切断电源。

大多数 Linux 发行版是和 System V[2] 相兼容的, 但是一些发行版如 Arch 和 Slackware 采用的是 BSD

[2] 从 System V 开始, pidof 或者 killall5 被用在很多发行版中。

风格,其它的如 Gentoo 是自己定制的。Ubuntu 和其它一些发行版现在开始采用 Upstart 来代替传统的 init 进程。

当 init 进程成功启动后,它会根据配置文件/etc/inittab[?] 中的设置初始化系统,这个过程主要完成的工作包括重新挂载文件系统、运行系统需要的进程和服务等。

```
[root@localhost ~]# cat /etc/inittab
#
# inittab
#
# This file describes how the INIT process should set up the system in a certain run-level
    .
#
# Author: Miquel van Smoorenburg, <miquels@drinkel.nl.mugnet.org>
#           Modified for RHS Linux by Marc Ewing and Donnie Barnes
#

# Default runlevel. The runlevels used by RHS are:
# 0 - halt (Do NOT set initdefault to this)
# 1 - Single user mode
# 2 - Multiuser, without NFS (The same as 3, if you do not have networking)
# 3 - Full multiuser mode
# 4 - unused
# 5 - X11
# 6 - reboot (Do NOT set initdefault to this)
#
#设置系统默认的运行级别
id:5:initdefault:

#初始化系统脚本
# System initialization.
si::sysinit:/etc/rc.d/rc.sysinit

#启动系统服务以及需要启动的服务的script的放置路径
l0:0:wait:/etc/rc.d/rc 0
l1:1:wait:/etc/rc.d/rc 1
l2:2:wait:/etc/rc.d/rc 2
l3:3:wait:/etc/rc.d/rc 3
l4:4:wait:/etc/rc.d/rc 4
l5:5:wait:/etc/rc.d/rc 5
l6:6:wait:/etc/rc.d/rc 6

#定义Ctrl+Alt+Delete键的作用
# Trap CTRL-ALT-DELETE
ca::ctrlaltdel:/sbin/shutdown -t3 -r now

#设置电源选项
# 电源失效时的设置
# When our UPS tells us power has failed, assume we have a few minutes
# of power left. Schedule a shutdown for 2 minutes from now.
# This does, of course, assume you have powerd installed and your
# UPS connected and working correctly.
pf::powerfail:/sbin/shutdown -f -h +2 "Power Failure; System Shutting Down"

# 电源恢复时的设置
# If power was restored before the shutdown kicked in, cancel it.
pr:12345:powerokwait:/sbin/shutdown -c "Power Restored; Shutdown Cancelled"


```

```
53 #启动终端
54 # Run gettys in standard runlevels
55 1:2345:respawn:/sbin/mingetty tty1
56 2:2345:respawn:/sbin/mingetty tty2
57 3:2345:respawn:/sbin/mingetty tty3
58 4:2345:respawn:/sbin/mingetty tty4
59 5:2345:respawn:/sbin/mingetty tty5
60 6:2345:respawn:/sbin/mingetty tty6
61
62 # Run xdm in runlevel 5
63 x:5:respawn:/etc/X11/prefdm -nodaemon
```

事实上，/etc/inittab 的设置类似于 shell script，上面的有效行都是按顺序从上往下处理的，而且它们都有一个共同的格式，就是使用冒号“:”将设置字段分隔为 4 个不同的字段。

/etc/inittab 中每一行的格式如下:

```
1 #[设置选项]:[runlevel]:[init的操作行为]:[命令选项]
2 id:runlevels:action:process
```

- id：配置行在配置文件中的标识符，最长可以由 4 个字符组成，代表 init 的主要工作选项。对于大多数文本行来说，这个字段没有太大的意义，但要求每行的 id 值在该文件中是唯一的。
- runlevels：配置行起作用的运行级别列表。如果作用于多个运行级别，可以将其写在一起，例如 35 则代表 runlevel 3/5 都会执行。
- action：配置 init 应该执行的动作。通常有 initdefault、sysinit、wait、ctrlaltdel、powerfail、powerokwait 和 respawn 这几个动作。
- process：表示配置行要执行的脚本、命令及参数等内容。

inittab 设置值	意义
initdefault	代表默认的 runlevel 设置值；
sysinit	代表系统初始化的操作选项；
ctrlaltdel	代表 [ctrl]+[alt]+[del] 三个按键是否可以重新启动的设置；
wait	代表后面字段设置的命令项目必须要执行完毕才能继续后面其他的操作；
respawn	代表后面字段的命令可以无限制的再生（重新启动）。例如，tty1 的 mingetty 产生的可登录界面，在用户注销结束后，系统可以再生成一个一个新的可登录界面等待下一个登录。

由上述可知，根据当前/etc/inittab 的设置，init 的处理流程如下：

1. 取得 runlevel 即默认执行等级；
2. 使用/etc/rc.d/rc.sysinit 进行系统初始化；
3. 执行`5:5:wait:/etc/rc.d/rc5` 中的脚本，其他略过；
4. 设置 [ctrl]+[alt]+[del] 三个按键是否可以重新启动；
5. 设置电源选项中 pf 和 pr 机制；
6. 启动 mingetty 的 6 个终端机（tty1 ～ tty6）；
7. 以`/etc/X11/perfdm -nodaemon` 启动图形界面。

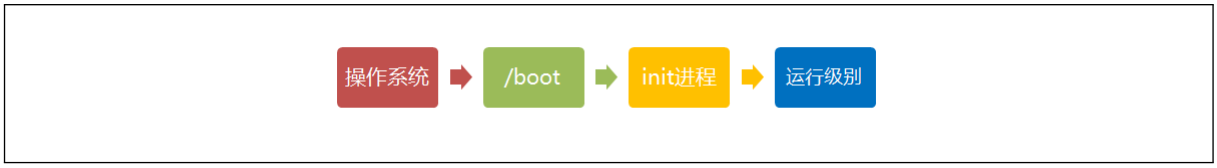
13.3 initdefault

许多程序需要开机启动。它们在 Windows 叫做“服务”（service），在 Linux 中叫做“守护进程”（daemon）。Linux 通过设置 runlevel 来规定系统使用不同的服务来启动，从而进入不同的使用环境，基本上根据有无网络与有无 X Window 而将 runlevel 分为 7 个等级。

具体而言，Linux 通过/etc/inittab 配置文件来决定 Linux 的运行等级(runlevel)，包括：

- 0 – halt(关机)
- 1 – Single user mode (单用户维护模式)
- 2 – Multiuser, without NFS(多用户模式，无 NFS 服务)
- 3 – Full multiuser mode(包含完整网络功能的纯文本多用户模式)
- 4 – unused(用户自定义)
- 5 – X11(多用户模式，有网络，有图形界面)
- 6 – reboot (重新启动)

init 进程的一大任务，就是去运行这些开机启动的程序。但是，不同的场合需要启动不同的程序，比如用作服务器时，需要启动 Apache，用作桌面就不需要。Linux 允许为不同的场合，分配不同的开机启动程序，这就叫做" 运行级别"(runlevel)。也就是说，启动时根据"运行级别"，确定要运行哪些程序。



Linux 预置七种运行级别(0-6)。一般来说，0 是关机，1 是单用户模式(也就是维护模式)，6 是重启。运行级别 2-5，各个发行版不太一样，对于 Debian 来说，都是同样的多用户模式(也就是正常模式)。

init 进程首先读取文件 /etc/inittab，它是运行级别的设置文件。

/etc/inittab 文件中设置的默认的运行模式在:initdefault: 项中，initdefault 用于设置系统启动时的默认运行级别，即系统启动后将自动进入到运行级别。这里，initdefault 的值是 5，表明系统启动时的运行级别为 5。

```
1 id:5:initdefault:
```

如果需要指定其他级别，可以手动修改 initdefault 值，但是同一时间内只能使用一个运行级别。在 root 权限下，运行 telinit 或者 init 命令可以改变当前的运行模式。

大多数操作系统的用户可以用下面的命令来判断当前的运行模式是什么：

```
1 $ runlevel
2 N   5
3 $ who -r
4     run-level 5 2014-04-12 12:16
```

修改系统默认的运行级别时，注意不要将字段设置为 0 和 6，否则系统将无法正常开机。如果黑客将此字段修改为 0 和 6，通常我们将其视为拒绝服务式攻击(Denial of Service，简称 DoS)。

那么，运行级别 5 有些什么程序呢，系统怎么知道每个级别应该加载哪些程序呢？..... 回答是每个运行级别在/etc 目录下面，都有一个对应的子目录，指定要加载的程序。

```
1 /etc/rc0.d
2 /etc/rc1.d
3 /etc/rc2.d
4 /etc/rc3.d
5 /etc/rc4.d
6 /etc/rc5.d
7 /etc/rc6.d
```

上面目录名中的 "rc"，表示 run command(运行程序)，最后的 d 表示 directory(目录)[3]。下面让我们看看 /etc/rc5.d 目录中到底指定了哪些程序。

[3].d 结尾是代表与 SysV init 相关的配置文件[2]，init 是有很多种实现的，上一代的是 SysV init，现在是两种——即 Ubuntu 所使用的 Upstart 实现以及主流的 systemd 实现，两种新的实现都兼容老的 SysV init 配置文件，但实际上几乎没有什么主流 distro 继续在用 SysV init 了，都是通过兼容实现的，所以有必要看一下 systemd 和 upstart 的手册和文档(runlevel 这个概念其实也是老的 SysV init 里面的)

```
$ ls /etc/rc5.d

README
K50netconsole
K90network
S00livesys
S95jexec
S99livesys-late
...
```

可以看到，除了第一个文件 README 以外，其他文件名都是” 字母 S+ 两位数字 + 程序名” 的形式。字母 S 表示 Start，也就是启动的意思（启动脚本的运行参数为 start），如果这个位置是字母 K，就代表 Kill（关闭），即如果从其他运行级别切换过来，需要关闭的程序（启动脚本的运行参数为 stop）。后面的两位数字表示处理顺序，数字越小越早执行。数字相同时，则按照程序名的字母顺序启动。

这个目录里的所有文件（除了 README），就是启动时要加载的程序。如果想增加或删除某些程序，不建议手动修改 /etc/rcN.d 目录，最好是用一些专门命令进行管理。

Linux 系统中，现代的 bootloader（包括 LILO 或 GRUB）允许用户在初始化过程中以最后启动的进程来取代默认的 /sbin/init。通常是在 bootloader 环境中通过执行 init=/foo/bar 命令。例如，如果执行 init=/bin/bash，启动单用户 root 的 shell 环境，无需用户密码。

BSD 的大多数变种可以设置 bootstrap 程序被中断后执行 `boot -s` 命令进入单用户模式。

当系统发现启动过程中报错时，比如不正常关机造成的文件系统的不一致的情况时，操作系统会主动进入单用户维护模式。

单用户模式并没有跳过 init，它仍然可以执行 /sbin/init，但是它将使 init 询问 exec() 将要执行的命令 (默认为 /bin/sh) 的路径，而不是采用正常的多用户启动顺序。如果内核启动时在 /etc/ttys 文件中被标注为 “不安全” (在某些系统中，当前的 “安全模式” 可能会有些变化)，在允许这种情况 (或者回退到单用户模式，如果用户执行 CTRL+D)，init 将首先询问 root 用户的密码。如果该程序退出，内核将在多用户模式下重新执行 init。如果系统从多用户模式切换到单用户模式，还将碰到上述的情况。

如果内核加载后，init 不能被正常启动将导致 panic 错误，此时系统将不可使用。想要通过 init 自身来改变 init 的路径，不同的版本情况不太一样。

- NetBSD 中可执行 `boot -a`；
- FreeBSD 中利用 init_path 命令装载变量。

13.4 rc.sysinit

/sbin/init 进程执行系统初始化脚本 (/etc/rc.d/rc.sysinit) 来对系统进行基本的配置（例如网络、时区等），并以读写方式挂载根文件系统（/）及其它文件系统。也就是说，/etc/rc.d/rc.sysinit 的作用就是在加载各项系统服务之前，为操作系统设置好系统环境。

```
#初始化系统脚本
# System initialization.
si::sysinit:/etc/rc.d/rc.sysinit
```

上面这行中的 runlevels 字段为空，表示 init 将会忽略这一字段，并在每个运行级别都执行系统初始化脚本文件/etc/rc.d/rc.sysinit。

系统初始化脚本文件/etc/rc.d/rc.sysinit[4]是一个可执行文件，它所做的事情如下：

1. 获取网络环境与主机类型。
   首先会读取网络环境设置文件”/etc/sysconfig/network”，获取主机名称与默认网关（gateway）等网络环境。

[4]根据不同的 Linux 发行版，/etc/rc.d/rc.sysinit 可能有些差异。例如，在 SUSE Server 9 中使用/etc/init.d/boot 与/etc/init.d/rc 来进行系统初始化的。

2. 测试与载入内存设备/proc 及 USB 设备/sys。
   除了挂载/proc 外，系统会主动检测是否有 USB 设备，并主动加载 USB 驱动，尝试载入 USB 文件系统。
3. 决定是否启动 SELinux。
   SELinux 检测是否需要启动，并检测是否需要为所有文件重新编写标准的 SELinux 类型（autorelabel）。
4. 接口设备的检测与即插即用（PnP）参数的测试。
   根据内核在启动时检测的结果（/proc/sys/kernel/modprobe）开始进行 IDE/SCSI/Network/Audio 等接口设备的检测，以及利用已加载的内核模块进行 PnP 设备的参数测试。
5. 用户自定义模块的加载。用户可以在”/etc/sysconfig/modules/*.modules”加入自定义的模块，此时会加载到系统中。
6. 加载内核的相关设置。系统会主动去读取”/etc/sysctl.conf”这个文件的设置值并配置内核相关功能。
7. 设置主机名与初始化电源管理模块（ACHI）
8. 设置系统时间（clock）与时区设置，并载入/etc/sysconfig/clock 设置。
9. 设置终端的控制台的字体。
10. 设置显示于启动过程中的欢迎画面。
11. 初始化磁盘阵列 RAID 及 LVM 等硬盘功能，主要是通过/etc/mdadm.conf 来设置。
12. 以 fsck 方式查看并检验磁盘文件系统。
13. 进行磁盘配额 Quota 的转换（非必要）。
14. 重新以可读写模式挂载系统磁盘。
15. 启动 Quota 功能。
16. 启动系统伪随机数生成器（pseudo-random）[5]。
    随机数生成器可以产生随机数，并用于系统进行密码加密演算，在此需要启动两次随机数生成器。
17. 清除启动过程中的临时文件。
18. 将启动相关信息加载到”/var/log/dmesg”文件中。
    通过查看/var/log/dmesg 可以了解启动过程中的各个步骤，执行：

```
1 dmesg
```

可以列出所有与启动过程有关的信息。

当/etc/rc.d/rc.sysinit 执行结束后，系统的基本设置就完成了，此时系统就可以顺利工作了。

基本上，在/sbin/init 执行过程中的很多工作的默认设置文件其实都在/etc/sysconfig/目录中。另外，用户还可以在系统初始化过程中加载自定义模块或驱动程序。

如果想要加载自定义内核模块，可以将整个模块写入到/etc/sysconfig/modules/*.modules 中，此时就需要系统自定义的设备与模块的对应文件（/etc/modprobe.conf）。

一般来说，/etc/modprobe.conf 大多用于指定系统内的硬件所需要的模块，通常它可以由系统来自行配置。只有当系统检测到错误的驱动程序或需要使用更新的驱动程序时来需要用户手动配置。

下一步需要启动系统相关的服务与网络服务等，这样主机才可以提供相关的网络和主机功能，因此便会执行接下来的脚本。

系统启动过程中必须要从/etc/sysconfig 目录中读取的相关配置文件包括：

- authconfig

[5] Theodore Y. Ts'o（曹子德）于 1994 年在 Linux 内核中实现了/dev/random 以及对应的核心驱动程序，让 Linux 成为所有操作系统中第一个实现了以系统背景噪音产生的真正随机数生成器。/dev/random 可以不用依靠硬件随机乱数产生器来独立运作，提升效能的同时也节省了成本。其他的守护行程（例如 rngd）可以从硬件取得随机数，提供给/dev/random，应用程序可以经由/dev/random 取得随机数。在/dev/random 与/dev/urandom 实现出来之后，很快就成为在 Unix、Linux、BSD 与 Mac OS 的共通标准接口。

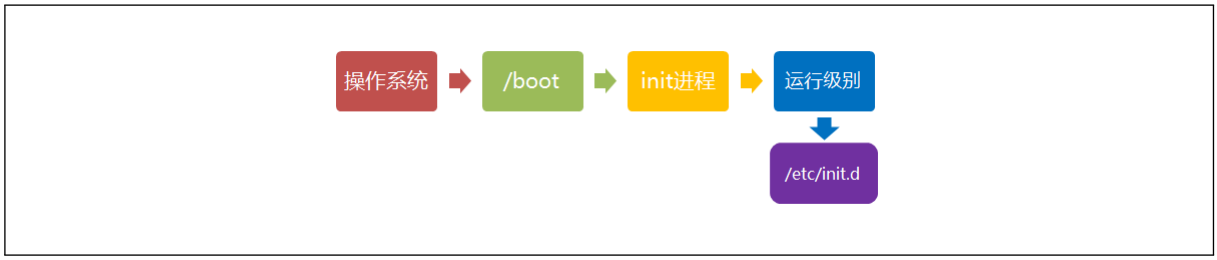
设置用户的身份认证的机制，包括是否使用本机的/etc/passwd、/etc/shadow 等，/etc/shadow 密码记录使用的加密算法，以及是否使用外部密码服务器(NIS、LDAP 等)提供的认证功能等。
系统默认使用 MD5 加密算法，并且不使用外部的身份认证机制。

- clock
  设置主机的时区，可以使用格林威治时间(GMT)，也可以使用本地时间(local)。基本上，在 clock 文件内的设置选项“ZONE”所参考的时区位于/usr/share/zoneinfo 目录下的相对路径中。如果要修改时区，需要把/etc/localtime 链接到/usr/share/zoneinfo 目录下相应的时区文件。
- i18n
  设置主机的语系。例如，为了在命令行下正确的显示日期，需要修改该文件中的 LC_TIME。
- keyboard
  设置主机的键盘。
- mouse
  设置主机的鼠标。
- network
  设置主机的网络选项，包括是否要启动网络、设置主机名和网关等。
- network-scripts
  设置主机的网卡选项。

13.5 rc.d

Linux 预设的“运行级别”各自有一个目录，存放需要开机启动的程序。Linux 将/etc/rcN.d 目录里列出的程序都设为链接文件，并统一指向另外一个目录 /etc/init.d ，/etc/init.d 这个目录名最后一个字母 d，是 directory 的意思，表示这是一个目录，用来与程序 /etc/init 区分。

真正的启动脚本都统一放在/etc/init.d 目录中，init 进程逐一加载开机启动程序，其实就是运行这个目录里的启动脚本。



/etc/rc.d/rc.sysinit 通过分析/etc/inittab 文件来确定系统的启动级别，然后执行相应的/etc/rc.d/rc*.d 下的脚本来启动相应的服务。

```
#设置系统默认的运行级别
id:5:initdefault:

#初始化系统脚本
# System initialization.
si::sysinit:/etc/rc.d/rc.sysinit

#启动系统服务
l0:0:wait:/etc/rc.d/rc 0
l1:1:wait:/etc/rc.d/rc 1
l2:2:wait:/etc/rc.d/rc 2
l3:3:wait:/etc/rc.d/rc 3
l4:4:wait:/etc/rc.d/rc 4
l5:5:wait:/etc/rc.d/rc 5
```

```
15 l6:6:wait:/etc/rc.d/rc 6
```

在确定了系统的运行级别后，接下来通过对应的/etc/rc.d/rc 脚本的执行如下：

- 通过外部第 1 号参数（$1）来取得要执行的脚本目录，这里得到的是/etc/rc.d/rc5.d 目录来准备处理相关的脚本程序；
- 定义相关服务启动的顺序是先 K 后 S，而具体的每个运行级别的服务状态是放在/etc/rc.d/rc*.d（*=0 ～ 6）目录下；
- 所有的文件均是指向/etc/init.d 下相应文件的符号链接。

```
1 /etc/init.d-> /etc/rc.d/init.d
2 /etc/rc ->/etc/rc.d/rc
3 /etc/rc*.d ->/etc/rc.d/rc*.d
4 /etc/rc.local-> /etc/rc.d/rc.local
5 /etc/rc.sysinit-> /etc/rc.d/rc.sysinit
```

/etc 目录下的 init.d、rc、rc*.d、rc.local 和 rc.sysinit 均是指向/etc/rc.d 目录下相应文件和文件夹的符号链接。这样做的另一个好处，就是如果要手动关闭或重启某个进程，直接到目录 /etc/init.d 中寻找启动脚本即可。比如要重启 Apache 服务器，就运行下面的命令：

```
1 $ sudo /etc/init.d/apache2 restart
```

这里以启动级别 5 为例来简要说明。/etc/rc.d/rc5.d 目录下的内容全部都是以 S 或 K 开头的链接文件，都链接到”/etc/rc.d/init.d”目录下的各种 shell 脚本，其中：

- S 表示的是启动时需要 start 的服务内容。
  对于/etc/rc5.d/S??.* 开头的文件，执行`/etc/rc5.d/K??.* stop`的操作。
- K 表示关机时需要关闭的服务内容。
  对于/etc/rc5.d/K??.* 开头的文件，执行`/etc/rc5.d/S??.* start`的操作。

/etc/rc.d/rc*.d 中的系统服务会在系统后台启动或关闭，实质上系统服务主要就是以/etc/int.d/服务文件名 {start, stop} 来启动或关闭的。如果要对某个运行级别中的服务进行更具体的定制，可以通过 chkconfig 命令来完成，或者通过 setup、ntsys、system-config-services 来完成。

如果用户需要自己增加启动的内容，可以在 init.d 目录中增加相关的 shell 脚本，然后在 rc*.d 目录中建立链接文件指向该 shell 脚本。

各个不同的服务其实是有依赖关系的，这里的 shell 脚本的启动或结束顺序是由 S 或 K 字母后面的数字决定，数字越小的脚本越先执行。例如，/etc/rc.d/rc5.d /S01sysstat 就比/etc/rc.d/rc5.d /S99local 先执行。这里/etc/rc.d/rc5.d/S99local 就是最后一个执行的选项，也就是/etc/rc.d/rc.local。

13.6 rc.local

在完成了默认的 runlevel 指定的各项服务的启动后，init 执行用户自定义引导程序/etc/rc.d/rc.local。

其实当执行/etc/rc.d/rc5.d/S99local 时，它就是在执行/etc/rc.d/rc.local。S99local 是指向 rc.local 的符号链接。就是一般来说，自定义的程序不需要执行上面所说的繁琐的建立 shell 增加链接文件的步骤，只需要将命令放在 rc.local 里面就可以了，这个 shell 脚本就是保留给用户自定义启动内容的。

13.7 mingetty

在完成了系统所有服务的启动后，init 启动终端模拟程序 mingetty 来启动 login 进程或 X Window 来等待用户登录。

```
1 #启动终端
2 # Run gettys in standard runlevels
3 1:2345:respawn:/sbin/mingetty 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

#启动X Window
# Run xdm in runlevel 5
x:5:respawn:/etc/X11/prefdm -nodaemon
```

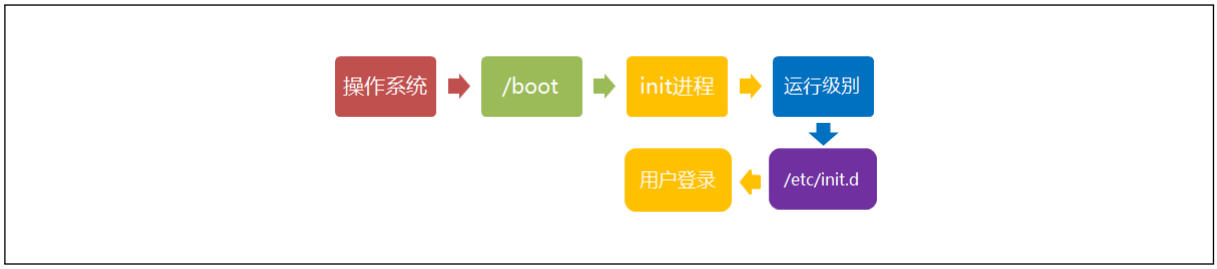
mingetty 就是启动终端的命令，上述设置表示在 run level 2/3/4/5 时，都会执行/sbin/mingetty。这里执行了 6 个，所以会有 6 个纯文本终端。

除了这 6 个文本模式终端之外，还会执行"`/etc/X11/prefdm-nodaemon`"来启动 X-Window。

respawn 表示当后面的命令被终止时，init 会主动重新启动该选项，因此当退出终端后会重新生成新的终端来等待输入。

13.8 login

开机启动程序加载完毕以后，就要让用户登录了。



一般来说，用户的登录方式有三种：

1. 命令行登录
2. ssh 登录
3. 图形界面登录

这三种情况，都有自己的方式对用户进行认证。

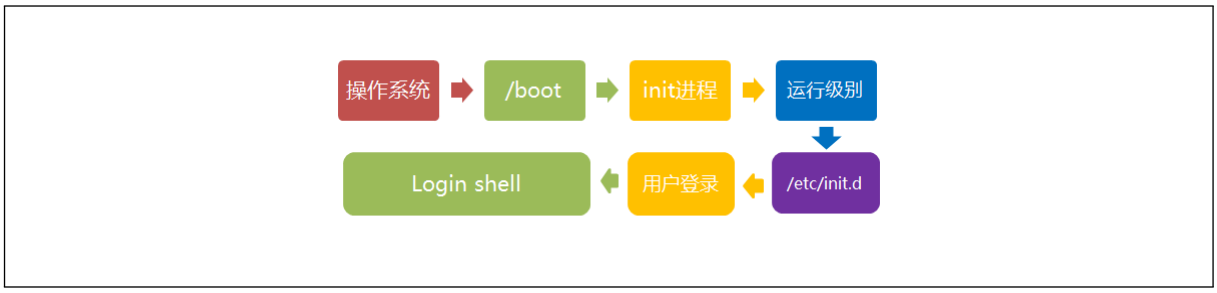
1. 命令行登录：init 进程调用 getty 程序（意为 get teletype），让用户输入用户名和密码。输入完成后，再调用 login 程序，核对密码（Debian 还会再多运行一个身份核对程序/etc/pam.d/login）。如果密码正确，就从文件 /etc/passwd 读取该用户指定的 shell，然后启动这个 shell。
2. ssh 登录：这时系统调用 sshd 程序（Debian 还会再运行/etc/pam.d/ssh ），取代 getty 和 login，然后启动 shell。
3. 图形界面登录：init 进程调用显示管理器，Gnome 图形界面对应的显示管理器为 gdm（GNOME Display Manager），然后用户输入用户名和密码。如果密码正确，就读取/etc/gdm3/Xsession，启动用户的会话。

13.8.1 Login Shell

所谓 shell，简单说就是命令行界面，让用户可以直接与操作系统对话。用户登录时打开的 shell，就叫做 login shell。

Debian 默认的 shell 是 Bash，它会读入一系列的配置文件。上一步的三种情况，在这一步的处理，也存在差异。

1. 命令行登录：首先读入 /etc/profile，这是对所有用户都有效的配置；然后依次寻找下面三个文件，这是针对当前用户的配置。




```
~/.bash_profile
~/.bash_login
~/.profile
```

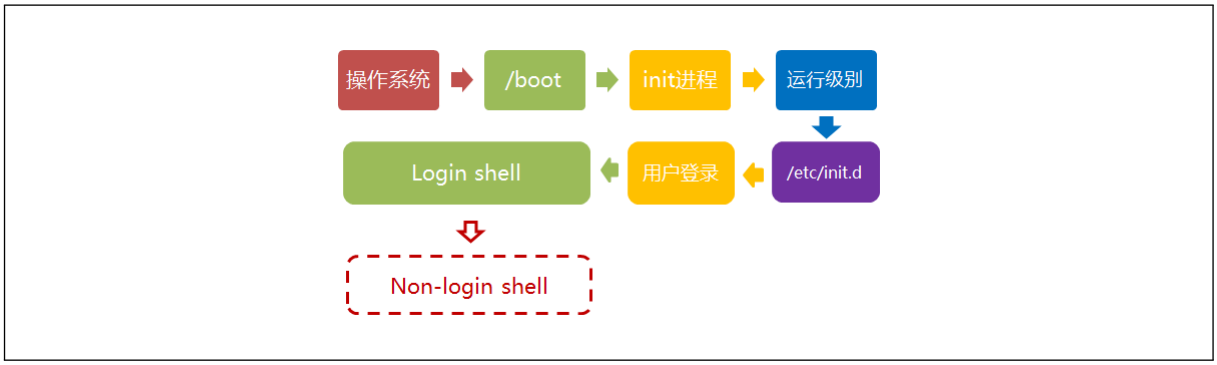
需要注意的是，这三个文件只要有一个存在，就不再读入后面的文件了。比如，要是~/.bash_profile 存在，就不会再读入后面两个文件了。

2. ssh 登录：与第一种情况完全相同。
3. 图形界面登录：只加载 /etc/profile 和~/.profile。也就是说，~/.bash_profile 不管有没有，都不会运行。

13.8.2 Non-login Shell

上一步完成以后，Linux 的启动过程就算结束了，用户已经可以看到命令行提示符或者图形界面了。但是，为了内容的完整，必须再介绍一下这一步。

用户进入操作系统以后，常常会再手动开启一个 shell。这个 shell 就叫做 non-login shell，意思是它不同于登录时出现的那个 shell，不读取/etc/profile 和.profile 等配置文件。



non-login shell 的重要性，不仅在于它是用户最常接触的那个 shell，还在于它会读入用户自己的 bash 配置文件~/.bashrc。大多数时候，我们对于 bash 的定制，都是写在这个文件里面的。

也许会问，要是不进入 non-login shell，岂不是.bashrc 就不会运行了，因此 bash 也就不能完成定制了？事实上，Debian 已经考虑到这个问题了，在文件~/.profile 中可以看到下面的代码：

```
if [ -n "$BASH_VERSION" ]; then
  if [ -f "$HOME/.bashrc" ]; then
    . "$HOME/.bashrc"
  fi
fi
```

上面代码先判断变量 $BASH_VERSION 是否有值，然后判断主目录下是否存在.bashrc 文件，如果存在就运行该文件。第三行开头的那个点是 source 命令的简写形式，表示运行某个文件，写成“source ~/.bashrc”也是可以的。

因此，只要运行~/.profile 文件，~/.bashrc 文件就会连带运行。但是上一节的第一种情况提到过，

如果存在～/.bash_profile 文件，那么有可能不会运行～/.profile 文件。解决这个问题很简单，把下面代码写入.bash_profile 就行了。

```
if [ -f ~/.profile ]; then
  . ~/.profile
fi
```

这样一来，不管是哪种情况，.bashrc 都会执行，用户的设置可以放心地都写入这个文件了。

Bash 的设置之所以如此繁琐，是由于历史原因造成的。早期的时候，计算机运行速度很慢，载入配置文件需要很长时间，Bash 的作者只好把配置文件分成了几个部分并分阶段载入。

- 系统的通用设置放在 /etc/profile；
- 用户个人的、需要被所有子进程继承的设置放在.profile；
- 不需要被继承的设置放在.bashrc。

除了 Linux 以外，Mac OS X 使用的 shell 也是 Bash。但是，它只加载.bash_profile，然后在.bash_profile 里面调用.bashrc。而且，不管是 ssh 登录，还是在图形界面里启动 shell 窗口，都是如此。

第 14 章

Run Level

运行级别(runlevel[?])指的是 Unix[1]或者 Linux 等类 Unix 操作系统下不同的运行模式。运行级别通常分为 7 等,分别是从 0 到 6,但如果必要的话也可以更多。例如在大多数 linux 操作系统下一共有如下 6 个典型的运行级别:

- 0 停机
- 1 单用户,Does not configure network interfaces, start daemons, or allow non-root logins
- 2 多用户,无网络连接 Does not configure network interfaces or start daemons
- 3 多用户,启动网络连接 Starts the system normally.
- 4 用户自定义
- 5 多用户带图形界面
- 6 重启

在 Debian Linux 中,2 ~ 5 这四个运行级别都集中在级别 2 上,这个级别也是系统预设的正常运行级别。在 Debian Linux 中,下列路径对应不同的运行级别。当系统启动时,通过其中的脚本文件来启动相应的服务。

- /etc/rc0.d Run level 0
- /etc/rc1.d Run level 1
- /etc/rc2.d Run level 2
- /etc/rc3.d Run level 3
- /etc/rc4.d Run level 4
- /etc/rc5.d Run level 5
- /etc/rc6.d Run level 6

可以通过 chkconfig 来添加服务到不同的运行级别或者取消服务的自动启动。例如,使用 chkconfig 命令来配置服务的示例如下:

```
#将服务添加到服务列表中，可以使用service camsd start 来启动服务
chkconfig --add camsd
#将服务删除出服务列表
chkconfig --del camsd
```

设置服务自动运行的示例如下:

```
使camsd服务在运行级别3和运行级别5自动运行。
chkconfig --level 35 camsd on
使camsd服务在运行级别3和运行级别5不再自动运行。
chkconfig --level 35 camsd off
```

查看服务的自启动状态的示例如下:

```
chkconfig --list camsd
```

事实上,与 runlevel 有关的启动其实是在/etc/rc.d/rc.sysinit 执行结束之后才开始的。也就是说,其实 runlevel 的不同仅是/etc/rc[0-6].d 目录中启动的服务不同而已,依据启动是否自动进入不同的 runlevel 的设置,可以说:

[1]The BSD[?] variants don't use the concept of run levels, although on some versions init(8) provides an emulation of some of the common run levels.

- 若要每次启动都执行某个默认的 runlevel，可以修改/etc/inittab 内的设置选项。
- 如果仅是暂时更改系统的 runlevel，可以使用`init [0~5]` 来进行更改。
- 每次系统重新启动时都以/etc/inittab 中的设置选项为准。

不同的 runlevel 只是加载的服务不同而已，因此当从 runlevel 5 切换到 runlevel 5 时，系统执行的配置如下：

- 首先比较/etc/rc.d/rc3.d 与/etc/rc.d/rc5.d 中以 K 与 S 开头的文件；
- 在新的 runlevel 中以 K 开头的文件，予以关闭；
- 在新的 runlevel 中以 S 开头的文件，予以启动。

查看当前 runlevel 的示例如下：

```
[root@theqiong ~]# runlevel
N 5
#左边代表前一个runlevel，右边代表当前的runlevel。
```

另外，可以分别利用`init 0` 和`init 6` 来关机和重启。

第 15 章

Usage

接下来需要思考一下,计算机系统是如何工作的。

举例来说,计算机屏幕上面显示的信息,是如何显示出来的?是通过显卡与屏幕显示的。那么如果要看视频呢?这时需要:

- 有视频文件;
- 可以转换视频文件输出的中央处理器;
- 可以显示图像的显示芯片(显卡);
- 可以传输声音的音效芯片(声卡);
- 可以输出图像的显示器;
- 可以发出声音的播放器。

也就是说,所有在“工作”的设备都是“硬件”,而且这些就是硬件的工作之一。

现在问题在于,计算机所进行的工作都是硬件来完成的,但是为什么这些硬件知道如何播放视频文件呢?原因就是操作系统在正确的控制硬件的工作。

操作系统可以管理计算机的硬件,包括控制 CPU 进行正确的运算,识别硬盘里的文件并进行读取,还能够识别所有的适配卡,这样才能让所有的硬件全部正确的操作。所以如果没有操作系统,计算机是没有用处的。

虽然操作系统可以完整的控制所有的硬件资源,但是对于用户来说还是不够的。因为操作系统虽然可以控制所有的硬件,但如果用户无法与操作系统交互,那么这个操作系统也就没有什么用处了。简单的来说,以上面的视频文件为例,虽然操作系统可以控制硬件播放视频,但是如果用户没有办法控制何时要播放,那么到底用户要怎么看视频呢?所以说,一个比较“完整的操作系统”应该要包含两部分,分别是:

- 核心与其提供的接口工具;
- 利用核心提供的接口工具所开发出来的软件。

计算机系统由硬件组成,操作系统的出现是为了更有效地控制计算机的硬件资源。操作系统在提供计算机运行所需要的功能(如网络功能)之外,还提供软件开发环境,也就是提供了一整套系统调用接口来满足软件开发需求。

现在用 Windows 电脑来做一个简单的说明。打开 Windows 电脑里面的“我的电脑”时,就会显示出硬盘中的文件。这个“显示硬盘里面的文件”,就是核心帮我们做的,但是如果我们要核心去显示硬盘中的某一个目录下的文件,则是由资源管理器这个工具来完成。

那么核心有没有做不到的事?当然有的,举例来说,如果曾经自行安装了比较新的显卡在计算机上,那么常常会发现 Windows 系统会提示:“找不到合适的驱动程序”,也就是说即使有最新的显卡安装在计算机上,而且也有播放视频文件的程序,但是因为“核心”无法使用这个最新的显卡,这时就无法正常的播放视频文件了。虽然所有硬件是由核心来管理的,但如果核心不能识别硬件,那么就无法使用该硬件设备。

> 在定义上,只要能够让电脑硬件正确无误的运行,那就算是操作系统了。所以说操作系统其实就是核心与其提供的接口工具,不过就如同上面所说,因为核心缺乏与用户沟通的界面,所以目前一般我们提到的“操作系统”都会包含核心与相关的应用软件。

核心是操作系统的最底层的东西，由它来管理所有硬件资源的工作状态。每个操作系统都有自己的核心，当有新的硬件加入到系统中的时候，若核心无法识别该硬件，那么这个新的硬件就肯定是无法工作的，因为控制它的操作系统并不能识别它。

一般来说，操作系统核心为了给出用户所需要的正确运算结果，必须要管理的事项有：

1. 系统调用接口（System Call Interface）
   为了方便开发者与操作系统核心的沟通来进一步的利用硬件的资源，需要操作系统核心提供接口来方便程序开发者。
2. 进程管理（Process Control）
   可能同一时间有很多的工作进入到 CPU 等待计算处理，操作系统核心必须要能够控制并调度这些工作进程，让 CPU 的资源得到有效的分配。
3. 内存管理（Memory Management）
   控制整个系统的内存管理，若内存不足，操作系统核心最好还能够提供虚拟内存的功能。
4. 文件系统管理（File System Management）
   文件系统的管理包括数据的输入输出（I/O）和不同文件格式的支持等，如果核心不能识别某个文件系统，那么将无法使用该文件格式的文件。
5. 设备驱动程序（Device Drivers）
   管理硬件是操作系统核心的主要工作之一，当然设备的驱动就是核心需要做的事情。在目前都有所谓的“可加载模块”功能，可以将驱动程序编译成模块，从而就不需要重新编译核心。

所有硬件的资源都是由操作系统核心来管理的，而用户要完成一些工作时，除了通过核心本身提供的功能之外，还可以借助其他的应用软件来完成。举个例子来说，要看视频文件时，除了系统提供的默认媒体播放程序之外，也可以自行安装视频播放软件来播放，这个播放软件就是应用软件，而这个应用软件可以帮助用户去控制核心来工作（就是播放视频）。可以这样说，核心是控制整个硬件系统的，也是一个操作系统的最底层。然而要让整个操作系统更完备的话，那还需要包含相当丰富的由核心提供的工具以及与核心相关的应用软件。

其实 Linux 就是一个操作系统，它含有最主要的操作系统核心[1]以及操作系统核心提供的工具。或者从本质上讲 Linux 就是一组软件，具体从计算机系统的分层架构来说，就是操作系统核心与系统调用接口。

现在，Linux 提供了一个完整的操作系统中最底层的硬件控制与资源管理的完整架构，这个架构沿袭了 UNIX 良好的传统。此外由于 Linux 的架构可以在 x86 架构计算机上面运行，所以很多的软件开发者将软件也移植到这个架构上面，于是也就有了很多的应用软件。虽然 Linux 仅是其核心与核心提供的工具，不过由于核心、核心工具与这些软件开发者提供的软件的整合，使得 Linux 成为一个更完整的、功能强大的操作系统。

约略了解 Linux 是什么之后，接下来我们要谈一谈，为什么说 Linux 是很稳定的操作系统以及它是如何来的。

如果能够参考硬件的功能函数来修改已有的操作系统或软件代码，那经过修改后的操作系统或软件就能够在其他的硬件平台上运行，这种操作就是我们通常所说的“软件移植”。

因为 Windows 操作系统本来就是针对个人计算机 x86 架构的硬件设计的，所以它当然只能在 x86 的个人电脑上面运行。也就是说，每种操作系统都是在对应的机器上面运行的，这一点需要先了解。不过由于 Linux 是 Open Source（开放源代码）的操作系统，所以其程序代码可以被修改成适合在各种架构的设备上面运行，也就是说 Linux 是具有“可移植性”的。

现代企业对于数字化的目标在于提供消费者或员工一些产品方面的信息以及整合整个企业内部的数据同一性（例如统一的账号管理/文件管理系统等）。另外，金融业等则强调在数据库、安全性等重大关键应用，而学术单位则需要强大的运算能力来进行模拟等，因此 Linux 的应用至少有下面这些：

1. 网络服务器

[1] Torvalds 在开发出 Linux 0.02 内核的时候，其实该内核仅能“驱动 386 所有的硬件”而已，即所谓的“让 386 计算机开始运行，并且等待用户命令输入”，事实上当时能够在 Linux 上面运行的软件还很少。

由上面的说明中，我们知道硬件是由操作系统“核心”来控制的，而每种操作系统都有其自己的核心，这就产生了一个很大的问题。因为早期硬件的开发者所开发的硬件架构或多或少都不相同，举例来说，2006 年以前的麦金塔是使用 IBM 公司开发的硬件系统（即所谓的 PowerPC），苹果公司则在该硬件架构上开发麦金塔操作系统，而 Windows 则是运行在 x86 架构上的操作系统。由于不同的硬件的功能函数并不相同，IBM 的 PowerPC CPU 与 Intel 的 x86 架构是不一样的，因此 Windows 就不能在苹果计算机上面运行。或者说，如果想要让 x86 上面运行的操作系统也能够在 PowerPC CPU 上运行，就要对操作系统进行修改才可以。不过在 2006 年以后，苹果公司转而使用 Intel x86 硬件架构，因此在 2006 年以后的苹果计算机，其硬件则可能“可以”支持安装 Windows 操作系统了。

承袭了 UNIX 高稳定性的良好传统，Linux 具有稳定而强大的网络功能，因此作为网络服务器来提供诸如 WWW、Mail Server、File Server、FTP Server 的功能等都成为了 Linux 的强项。

金融业与大型企业也逐渐开始采用 x86 架构主机，学术机构的研究经常需要自己开发应用软件，而 Linux 自身的性能和软件开发及编译环境正好符合这些需求。在实际应用场景中，为了加强整体系统的性能，计算机集群系统（Cluster）的并行计算能力需求也很突出。并行计算指的是将原来的工作分成多份然后交给多台主机去运算，最后再把结果收集起来。这些主机一般是通过高速网络连接起来的，使用计算机集群就可以缩短运算时间，例如气象预报、数值模拟等。

2. 工作站计算机

   工作站计算机与服务器不一样的地方大概就是在于网络服务。工作站计算机本身是不应该提供 Internet 的服务的（LAN 内的服务则可接受）。此外，工作站计算机与桌上型计算机不太一样的地方在于工作站通常得要应付比较重要的计算任务，例如流体力学的数值模式运算、娱乐事业的特效处理、软件开发者的工作平台等。

   Linux 上面有强大的运算能力以及支持度相当广泛的 GCC 编译软件，因此在工作站当中也是相当良好的一个操作系统选择。

3. 桌上型计算机

   桌上型计算机其实就是在办公室使用的计算机，一般称之为 Desktop。

4. 嵌入式系统

   Linux 的核心拥有可定制性以及高效率等特性，从而在嵌入式设备的市场当中具有很大的竞争优势。

   Linux 核心核心里面包含了很多嵌入式设备可能用不到的模块，所以可以将所有不需要的功能移除仅保留需要的程序，这对于嵌入式设备锱铢必较的存储空间来说是很关键的。由于 Linux 核心非常小巧精致，因此可以在很多省电以及较低硬件资源的环境下运行。

目前 Linux 有两种主要的操作模式，分别是图形界面与文字界面。用户所看到的图形界面是使用 X Server 提供的显示相关硬件的功能来达到图形显示的“Window Mananger”所产生的，也就是说，WM 是在 X Server 上面来运行的一套显示图形界面的软件，例如常见的 KDE、GNOME 等都是 WM。如果没有图形界面的辅助，那么将对 Desktop 用户造成很大的困扰。

Linux 早期都是由工程师所发展的，对于图形界面并不是很需要。如果仅想要了解 Linux，并且利用 Linux 来作为桌上型计算机，那么只需要了解 Linux 桌面设定，例如中文输入法、打印机设定、网络设定等概念即可，这时可以选择专为桌上型计算机发行的 Linux distributions。

在 1986 年图形界面就已经在 UNIX 上面出现了，那时图形界面被简称为 X 系统，而后来到了 1994 年的时候正式被整合在 Linux 中，微软的 Windows 则是在 1995 年才出现。X Window System 就是以 XFree86 这个计划开发的 X11 视窗软件为管理显示核心的一套图形界面的软件，简称为图形用户界面（Graphical User Interface，GUI）。

XFree86 只是 Linux 核心上面的一套软件而已，其主要工作就是管理图形界面输出时几乎所有与显示相关的硬件的控制，如显卡、显示器、键盘、鼠标等都是由 XFree86 管理的。或者，可以称 XFree86

为 X Window System 的服务器，简称为 X Server。X Window System 只是 Linux 上面的一套软件，所以即使 X Window 崩溃，对 Linux 也可能不会有直接的影响。

TLDP（The Linux Documentation Project）几乎列出了所有 Linux 上面可以看到的文献资料等，除了基本的 FAQ 之外，其实还有更重要的问题查询方法，那就是利用 Google 去搜寻答案。

注意，发生错误的时候，可以先自行以显示器前面的信息来进行 debug（除错）操作，然后如果是网络服务的问题时，可以到/var/log 目录里去查看一下 log file（日志文件），这样可以几乎解决大部分的问题了。一般而言，从 Linux 在发送命令的过程当中或者是 log file 里就可以自己查得错误信息了。

第 16 章

Linux 主机规划

为了配置合理的 Linux 主机系统，除了考虑后续的维护之外首先要根据使用的 distributions 的特性、软件、升级需求、硬件扩展性以及主机预期的“工作任务”来选择最合适的硬件。

俗话说“钱要花在刀口上”，没有必要仅仅为了 IP 共享功能买双 CPU 的硬件架构，但是简单的个人计算机也确实无法满足中大型企业的工作环境需求，下面要介绍的就是在开始安装 Linux 之前，应该要先思考哪些工作来方便后续的主机维护。

因为 Linux 对于较新的硬件的支持不足，原因在于 Linux 内核针对新硬件所纳入的驱动程序模块比不上硬件更新的速度，并且硬件厂商针对 Linux 推出驱动程序的速度较慢，所以首先必须要了解主机是否被 Linux 支持。此外也必须要先了解 Linux 的预计功能，这样在选择硬件时才会了解哪些是最重要的。

举例来说，需要桌面环境的用户应该会用到 X Window，此时显卡的优劣与内存的大小就要注意。如果要做文件服务器，那么硬盘或者是其他的储存设备应该就是最先要增购的部件。

Linux 对于计算机各部件/设备的识别与 Microsoft Windows 系统完全不一样，因为各个部件或设备在 Linux 系统下都是“一个文件”。

16.1 主机的硬件配置

Linux 早期是与 x86 架构的个人计算机系统紧密结合的，而且硬件与操作系统的关系也很大。事实上 Linux 主机的硬件配备与主机功能是很相关的。

> Tips：
> 个人计算机的发展不断的向前延伸，各种接口也在不断的改进，PCI Express、AGP 渐渐被淘汰，IDE 接口被 SATA 接口所取代，这也导致很多旧的配备无法被重复利用。
> 目前全世界前两大的个人计算机 CPU 制造商为 Intel 与 AMD。早期的 CPU 规格都是由 Intel 来拟定，后来 AMD 在 x86 的架构上发布自己的 CPU 脚位，CPU 结构的变更使得脚位的定义越来越多。由于规格太多而且 CPU 的插脚的脚位都不一样，有的即使一样但是 CPU 的运算电压不同也导致无法兼容。

不同的 CPU 之间不可以单纯用时钟频率来判断运算的效率，时钟频率目前仅能用来比较同样的 CPU 的速度。

另外比较特别的是 CPU 的“倍频”与“外频”，其中外频是 CPU 与周边设备进行数据传输/运算的速度，倍频则是 CPU 本身运算时候加上去的一个运算速度，两者相乘才是 CPU 的时钟频率。

与 CPU 外频有关的是内存与主板芯片组。一般来说，越快的时钟频率代表越快的 CPU 运算速度，以 Intel 的 P III 时钟频率 933MHz 为例说明如下：

CPU 外频与倍频：133(外频)× 7(倍频)MHz；

RAM 频率：通常与 CPU 的外频相同，为 133MHz；

PCI 接口(包含网卡、声卡等的接口)：133/4=33MHz；

AGP 接口：133/2=66MHz(这是 AGP 正常的频率)

外频是可以超频的，原本的 CPU 外部频率假设是 133，如果通过某些工具或者主板自身提供的工具就可以将 133 提升到比较高的频率，这就是所谓的超频。

超频可以在比较便宜的 CPU 上面让频率升到比较高。不过超频本身的风险很高，如果是超外频的话，例如到 166MHz 时，AGP 将达(166/2=83)而 PCI 也将达(166/4=41.5)，高出正常值很多。

通常，越快的外频会让所有的设备运算频率都会提升，所以可以让效率提高不少，但也可能会造成系统不稳定，例如常常死机或者是缩短某些部件的寿命等。

不建议在 Linux 系统中超频，因为即使 CPU 可以支持较高的时钟频率，但是系统的运算是全面性的，超频之后系统频率高出正常值太多，所以容易造成系统不稳定。

另一个需要注意的是 CPU 是分等级的，目前很多的程序都针对 CPU 作了优化，所以就会有所谓的 i386、i586、i686 等的扩展名。

基本上，在 P MMX 以及 K6-III 都称为 586 的 CPU，而 Intel 的赛扬以上等级与 AMD 的 K7 以上等级，就被称为 686 的 CPU。

计算机真正运算的核心是 CPU，但是真正传送给 CPU 运算数据的是内存。操作系统的核心、软硬件的驱动程序、所有要读取的文件等都需要先读入内存之后才会传输到 CPU 来进行数据的运算。

此外，操作系统也会将常用的文件或程序等数据常驻在内存内而不直接移除，这样下次取用这个数据时就不需要在去周边存取设备再重新读取一次。

对于一个系统来说，通常越大的内存代表越快速的系统，这是因为系统不用常常释放一些内存内部的数据。以服务器为例，内存容量的重要性有时比 CPU 的速度还要高，而且如果内存不够大，系统就会使用一部分硬盘空间作为虚拟内存，因此内存太小可能会影响到整体系统的性能。

显卡对于图形接口有相当大的影响，要将视频数据显示到显示器时就需要使用到显卡 (VGA Card) 的相关硬件功能。目前 3D 的画面在计算机游戏接口与工作接口被大量的使用，但是如果这些 3D 画面没有先经过处理而直接进入 CPU，将会影响到整体运算的速度。

显卡开发商在显卡上面安装可以处理这些很耗 CPU 运算时间的硬件来处理这些画面数据，这样不但图形画面处理的速度加快，CPU 的资源也会被用来执行其他的工作。

另外显存的大小可以影响显示器输出的解析度与像素，显存是直接嵌入显卡的，与系统内存没有关系。服务器没有 X Window，显卡并不重要，个人计算机是需要使用到图形接口的，那么这个显存的容量就比较重要。

在个人计算机上面，主流的硬盘存取接口应该是 SATA 与 IDE，SATA 硬盘存取效率要比传统的 IDE 接口高。此外 SATA 的特色就是它与主板连接的排线可以比较长（可长达 1m），并且排线比较细，可以改善主机机壳内部的通风散热。

在 Linux 中 SATA 与 IDE 接口的命名方法稍有不同。一个 IDE 插槽可以接两个 IDE 接口的设备，通过 IDE 设备的跳针（Jumper）来设定主盘（Master）和从盘（Slave），可以在一个 IDE 接口接的两个设备上面以排线接一个 Master 以及一个 Slave 的设备，而 Master 与 Slave 可以在任何一个 IDE 设备上面找到。也就是说，如果有两个硬盘，那么可以将任何一个设成 Master，但是另外一个则必须为 Slave，否则 IDE 接口会无法识别。

这四个 IDE 设备的文件名如下表所示：

IDE/Jumper	Master	Slave
IDE1(Primary)	/dev/hda	/dev/hdb
IDE2(Secondary)	/dev/hdc	/dev/hdd

再以 SATA 接口来说，由于 SATA/USB/SCSI 等磁盘接口都是使用 SCSI 模块来驱动的，因此这些接口的磁盘设备文件名都是/dev/sd[a-p] 的格式，但是与 IDE 接口不同的是，SATA/USB 接口的磁盘根本就没有一定的顺序，因此只能根据 Linux 内核检测到的磁盘的顺序来命名设备代号，而与实际插槽代号无关。

另外，USB 磁盘设备代号要开机完成后才能被系统识别。

通常在 586 之后生产的主板上面都有两条接排线的接口，而我们称这种接口为 IDE 接口，这也是之前的主流硬盘接口（目前已被 SATA 取代），为了区分硬盘读取的先后顺序，主板上面的这两个接口分别被称为 Primary（主要的）与 Secondary（次要的）IDE 接口，或者被称为 IDE1（Primary）与 IDE2（Secondary）。每一条排线上面还有两个插孔，也就是说一条排线可以接两个 IDE 接口的设备（硬盘或光驱），而现在有两条排线，因此主板在默认的情况下，应该都可以连接四个 IDE 接口的设备。

每条排线上面该如何判别哪一个是主盘（Master），哪一个是从盘（Slave ）呢？

这个时候就需要调整硬盘上面的跳针（jumper）才知道，可以通过查看一下硬盘上的图示说明来确定。

另外硬盘的 master/slave 判断方法中，除了利用 jumper 主动调整之外，还可以通过 cable 自动选择。

如果光驱已经占用了一个硬盘位置，那么最多就只能再安装三个 IDE 接口的硬盘到主机上。

硬盘与 Linux 的硬盘代号有关，由 IDE 1（Primary IDE）的 Master 硬盘开始计算，然后是 IDE 2 的 Slave 硬盘，所以各个硬盘的代号是：

```
IDE\Jumper Master Slave
IDE1(Primary) /dev/hda /dev/hdb
IDE2(Secondary) /dev/hdc /dev/hdd
```

如果只有一个硬盘，而且该硬盘接在 IDE 2 的 Master 上面，那么它在 Linux 里面的代号就是/dev/hdc。如果这个硬盘被分区成两个硬盘分区（partition），那么每一分区在 Linux 里面的代号又是什么，另外如何知道每个 partition 的代号呢？

目前的主机系统硬件要求中，硬盘的转速至少为 7200 转/分，缓冲内存最好大一些。如果主机作为备份服务器或者是文件服务器，那就可能要考虑使用磁盘阵列（RAID），磁盘阵列是利用硬件或软件技术将数个硬盘整合成为一个大硬盘的方法，操作系统只会看到最后被整合起来的大硬盘。由于磁盘阵列是由多个硬盘组成，所以可以在速度、性能、备份等方面有明显优势。

PCI 接口的设备也包括主板内置的 PCI 设备等，其中网卡可以用来连接 Internet，网卡目前都已经可以支持 10/100/1000Mbps 的传输速度，但是网卡的好坏却差别很大。同样是支持 10/100/1000Mbps 的

网卡，Intel 与 3Com 的网卡要比一般的杂牌网卡稳定性好，并且占用 CPU 资源低，还会具备其他特殊功能等。

SCSI 接口卡可以用来连接 SCSI 接口的设备。以硬盘为例，目前的硬盘除了个人计算机主流的 IDE/SATA 接口之外，还有 SCSI 接口。由于 SCSI 接口的设备比较稳定，而且运转速度较快，因而速度也会快的多，而且占用 CPU 的资源也比较低。

主板负责沟通所有接口的工作，而沟通所有上面提到硬件的就是主板的芯片组。由于主板上面的芯片组将负责与 CPU、RAM 及其他相关的输出、输入设备的数据通信，所以芯片组设计的好坏也相差很多。

需要再次强调的是，CPU 的外频就是指 CPU 与其他周边设备沟通的速度，而主板芯片组就负责管理各种不同设备的时钟频率操作，因此芯片组的好坏对于系统的影响也是相当大的。另外，目前很多技术可以提升各个设备与芯片组之间沟通的时钟频率。

16.1.1 设备 I/O 地址与 IRQ 中断

主板负责各个计算机系统部件之间的沟通，但是计算机的东西又太多了，既有输出输入，又有不同的存储设备，主板芯片组主要通过设备 I/O 地址与 IRQ 来完成这些功能。

I/O 地址有点类似门牌号码，每个设备都有它自己的地址，一般来说不能有两个设备使用同一个 I/O 地址，否则系统就会不知道该如何运算，例如如果你家门牌与隔壁家的相同，那么邮差就没法正确送信到你家。

不过，万一还是造成不同的设备使用了同一个 I/O 地址和而造成 I/O 冲突时，就需要手动的设定一下各个设备的 I/O 地址。

除了 I/O 地址之外，还有 IRQ 中断。如果 I/O 可以想象成是门牌号码的话，那么 IRQ 就可以想象成是各个门牌连接到邮件中心(CPU)的专门路径，IRQ 可以用来沟通 CPU 与各个设备。

目前 IRQ 只有 15 个，周边接口太多时可能就会不够用，此时可以选择将一些没有用到的周边接口关掉以空出一些 IRQ 来给真正需要使用的接口，当然也有所谓的 sharing IRQ 的技术。

BIOS 是 Basic Input/Output System 的缩写，输出与输入以及 I/O 地址，IRQ 等都是通过 BIOS 或操作系统来进行设定的。

16.2 硬件设备在 Linux 中的代号

以 Windows 系统的观点来看，一块硬盘可以被分区为 C:、D:、E: 等分区，但是在 Linux 系统中，每个设备都被当成一个文件来处理，举例来说，硬盘的文件名称可能就是/dev/hd[a-d]，其中括号内的字母为 a-d 当中的任何一个，也就是/dev/hda，/dev/hdb，/dev/hdc 及/dev/hdd 这四个文件，这四个文件分别代表一个硬盘，另外光驱与软驱分别是/dev/cdrom 和/dev/fd0。

在 Linux 系统当中，几乎所有的硬件设备代号文件都在/dev 这个目录中，所以会看到/dev/hda、/dev/cdrom 等。

下面列出几个常见的设备与其在 Linux 当中的代号：

设备	设备在 Linux 内的代号
IDE 硬盘驱动器	/dev/hd[a-d]
SCSI/SATA/USB 硬盘驱动器	/dev/sd[a-p]
USB 存储器	/dev/sd[a-p]（与 SCSI 硬盘）
当前 CD ROM/DVD ROM	/dev/cdrom
软驱	/dev/fd[0-1]
打印机	25 针：/dev/lp[0-2] USB：/dev/usb/lp[0-15]
当前鼠标	/dev/mouse
磁带机	IDE：/dev/ht0 SCSI：/dev/st0

需要特别留意的是硬盘驱动器代号（不论是 IDE/SCSI/USB 都一样），每个硬盘驱动器的分区（partition）不同时，其硬盘代号还会改变。

/dev 是放置设备文件的目录，需要特别注意的是磁带机的代号，在某些不同的 distribution 当中可能会发现不一样的代号。

16.2.1 查看内核版本

Linux 内核有三个不同的命名方案，其中第一个版本的内核是 0.01，其次是 0.02、0.03、0.10、0.11、0.12（第一 GPL 版本）、0.95[1]、0.96、0.97、0.98、0.99 及 1.0。

接下来的旧计划（1.0 和 2.6 版之间）的版本格式为 A.B.C，其中 A、B、C 分别代表：

- A 大幅度转变的内核。这是很少发生变化，只有当发生重大变化的代码和核心发生才会发生。在历史上曾改变两次的内核：1994 年的 1.0 及 1996 年的 2.0。
- B 是指一些重大修改的内核。内核使用了传统的奇数次要版本号码的软件号码系统（用偶数的次要版本号码来表示稳定版本）。
- C 是指轻微修订的内核。这个数字当有安全补丁，bug 修复，新的功能或驱动程序，内核便会有变化。

自 2.6.0（2003 年 12 月）发布后，人们认识到更短的发布周期将是有益的，也就自那时起，版本的格式改为 A.B.C.D，其中 A、B、C、D 分别代表：

- A 和 B 是无关紧要的
- C 是内核的版本
- D 是安全补丁

自 3.0（2011 年 7 月）发布后，版本的格式为 3.A.B，其中 A、B 分别代表：

- A 是内核的版本；
- B 是安全补丁。

查看当前的内核版本的代码示例如下：

[1] 从 0.95 版有许多的补丁发布于主要版本版本之间。

```
$ uname -a
Linux theqiong 3.12.10-300.fc20.x86_64 #1 SMP Thu Feb 6 22:11:48 UTC 2014 x86_64 x86_64 x86_64
     GNU/Linux
$ uname -r
3.12.10-300.fc20.x86_64
$ cat /proc/version
Linux version 3.12.10-300.fc20.x86_64 (mockbuild@bkernel02) (gcc version 4.8.2 20131212 (Red
    Hat 4.8.2-7) (GCC) ) #1 SMP Thu Feb 6 22:11:48 UTC 2014
```

16.2.2 查看系统版本

```
$ lsb_release -a
LSB Version: :core-4.1-amd64:core-4.1-noarch:cxx-4.1-amd64:cxx-4.1-noarch:desktop-4.1-amd64:
    desktop-4.1-noarch:languages-4.1-amd64:languages-4.1-noarch:printing-4.1-amd64:printing
    -4.1-noarch
Distributor ID: Fedora
Description: Fedora release 20 (Heisenbug)
Release: 20
Codename: Heisenbug
```

每个版本的 Linux 都是使用http://www.kernel.org所发布的核心，它们都遵循 LSB 与 FHS 等的架构，所以差异性其实不大。不过每个 Linux distributions 在发布的时候都有相应的用户群，因此在默认配置中每个版本都有比较适合的用户群体。

如果要查看当前使用的 Linux distribution 使用的 Linux 标准（Linux Standard Base）以及 Linux 内核版本，可以使用如下的命令：[7]

```
[root@linux ~]# lsb_release -a

[root@linux ~]# uname -r
```

举例来说，Ubuntu 就比较适合 Desktop 计算机使用，因为 X Window 整合得很好。RHEL、CentOS 与 SuSE Enterprise Linux Server 就比较适合企业的 Linux 主机，因为他们的系统服务整合的比较好。

但是上面提到的都是默认配置下的使用状态，事实上因为每个 linux distributions 差异性不大，当然可以随意选择一个 distributions 来加以改造以符合自己的应用需求。

要选择的 distributions 的标准之一就是，“选择比较新的 distribution”，这是因为比较新的版本在持续维护套件的安全性上可以让系统比较稳定，而且比较新的 distributions 在新硬件的支持上面当然也会比较好。

Linux 开发商在发布 distribution 之前，都会针对该版本所默认可以支持的硬件做说明，因此除了可以在 Linux 的 How-To 文件中查询硬件的支持信息之外，也可以到各个相关的 Linux distribution 网站去查询。

```
Red Hat 的硬件支持:https://hardware.redhat.com/hcl/?pagename=hcl
Mandriva 的硬件支持:http://hcl.mandrake.com/
Open SuSE 的硬件支持:http://en.opensuse.org/Hardware?LANG=en_UK
Linux 对打印机的支持:http://www.openprinting.org/
Linux 对笔记本电脑的支持:http://www.linux-laptop.net/
显卡对 XFree86/Xorg 的支持:http://www.linuxhardware.org/
```

16.3 主机的服务规划与硬件的关系

由于主机的服务目的不同，所需要的硬件等级与配备自然也就不一样。在某些情况之下可能会想要在主机上安装两套以上的操作系统，那么就可能会需要使用到“多重启动”系统，也就是在系统开机时可以选择进入哪一种操作系统的程序。

可以在主机上安装两套操作系统在不同的硬盘分区内，此时就能够以一台主机来运行两个或多个操作系统。

Tips：

一般来说，还可以在 Windows 操作系统中安装 VMware 等软件，从而可以在 Windows 系统上面使用 Linux 系统，此时就是两个操作系统同时启动。不过那样的环境比较复杂，而且很多硬件都是模拟仿真的，导致很难理解系统控制原理。基本上是不建议使用这样的方式来学习 Linux 的。

基本上，多重启动都要涉及到硬盘规划的问题。

一般而言，对于非企业或者是小型企业或者是学校单位，通常需要的服务有下面这几个：

1、NAT

如果是小型企业或者是中小学学校，那么该单位对外的连线应该通常是：

申请一个固定 IP ，然后通过 IP 共享（IP sharing）来让所有计算机都可以连上 Internet。在当初规划 IPv4 协议时就考虑到可能的 IP 不足，于是就有了专门给内部网域设定用的 Private IP（或者称为私有 IP 或保留 IP）。需要注意的是，这些 Private IP 都不能直接与 Internet 上面的 Public IP 互相通信。

NAT（Network Address Translation）功能可以让内部计算机通过 NAT 技术将内部计算机的数据封包中关于 IP 的设定都设定成 NAT 主机的公共 IP，然后才传送到 Internet，这样内部计算机虽然是使用私有 IP，但是在连上 Internet 时就可以通过 NAT 技术将 IP 来源改变，从而可以向 Internet 请求数据。

NAT 主机服务中，比较重要的就是网卡，其他的比如 CPU、内存、硬盘等的影响就小得多。事实上，只是利用 Linux 作为 NAT 主机来完成路由功能是很不明智的，因为 PC 的耗电量比路由器大得多。但是 Linux NAT 可以安装很多分析软件，可以用来分析客户端的连接，或者是用来控制带宽与流量来使得带宽的使用更公平。

2、SAMBA

在 Windows 里面可以很轻易的就以网上邻居来分享彼此的文件数据，在 Linux 中可以使用 SAMBA，这也是最普遍的 file server（文件服务器）。由于分享的数据量可能较大，那么系统的网卡与硬盘的大小及速度就比较重要，如果针对不同的用户提供文件服务器功能，那么/home 可以考虑独立出来，并且加大容量。

SAMBA 与网上邻居相比，Windows XP 的网上邻居一般只能同时分享 10 个客户端连接，而 SAMBA 则没有客户端连接数的限制，而且性能也不错。

3、Mail

Linux 可以提供 Sendmail 或 Postfix 的邮件服务，在 Mail Server 上面比较重要的也是硬盘容量与网卡速度，在此应用中也可以将/var 独立出来，并加大容量。

4、Web 服务

在 Web server 上面，CPU 的等级不能太低，而最重要的则是 RAM。要增加 Web 服务器系统的稳定性，提升 RAM 是一个不错的办法。

5、DHCP

设置好 NAT 之后，每一个 Client（客户端）都需要经过设定才能上网，这时使用 DHCP 就可以解决这个问题，这样 Client 端都不必进行任何设定就可以“自动获取 IP”上网。

6、Proxy

在带宽较不足的环境下，Proxy 可以有效的解决带宽不足的问题，也可以在家庭内部安装 Proxy，但是 Proxy 服务器的硬件要求可以说是相对而言最高的，不但需要较好的 CPU 来运算，对于硬盘的速度与容量要求也很高。

7、FTP

FTP 对主机硬盘容量与网卡好坏要求较高。

安装 NAT 服务时通常会建议安装两块网卡。在主机上面可以解决内部计算机的安全问题。如果需要 Mail 与 Web 服务器，就建议申请 DNS 或者是直接申请免费的动态 DNS 系统的域名（domain name）。如果需要 Proxy 服务，那么在进行硬盘规划时就要仔细考虑硬盘的分区，因为不同的分区方式会使得 Proxy 效率有差别。

第 17 章

Disk Structure

在系统管理中要管理好自己的分区和文件系统，每个分区不可太大也不能太小，太大会造成磁盘容量的浪费，太小则会产生文件无法存储的困扰，而明白硬盘的组成及文件系统的概念之后，才能理解 Linux 文件系统（File system）如何记录文件以及读取文件，而文件的权限与属性就要引入文件系统的 inode 和 block 的概念。

文件系统是创建在硬盘[3] 上面的，硬盘又是由一些圆形硬盘片组成，根据硬盘片能够容纳的数据量而有所谓的单片（一块硬盘里面只有一个硬盘片）或者是多片（一块硬盘里面含有多个硬盘片）的硬盘。

- 圆形的硬盘片是主要记录数据的部分。
- 硬盘里面有所谓的磁头（Head）可以读写硬盘片上的数据，而磁头是固定在机械臂上面的，机械臂上有多个磁头可以进行读取的动作。
- 主轴马达可以转动硬盘片，当磁头固定不动（假设机械臂不动），硬盘片转一圈所画出来的圆就是所谓的磁道（Track）。
- 一块硬盘里面可能具有多个硬盘片，所有硬盘片上面相同半径的那一个磁道就组成了所谓的柱面（Cylinder）。
- 硬盘片上面的同一个磁道就是一个柱面，柱面是硬盘分区（Partition）时的最小单位。
- 由圆心向外划直线，则可将磁道再细分为一个一个的扇区（Sector），扇区就是硬盘片上面的最小存储物理量。
- 通常一个扇区的大小约为 512 Bytes。

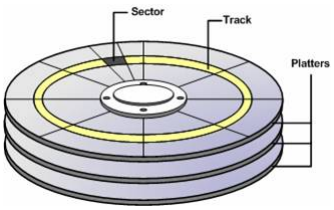


图 17.1: Hard Disk Schematic

在计算整个硬盘的存储量时，简单的计算公式就是：

Cylinder × Head × Sector × 512 Bytes

另外，硬盘在读取时主要是“硬盘片会转动，利用机械臂将磁头移动到正确的数据位置（单方向的前后移动），然后将数据依序读出”。在这个操作的过程当中，由于机械臂上的磁头与硬盘片的接触是很细微的空间，如果有震动或者是脏污在磁头与硬盘片之间时，就会造成数据的损坏或者是实体硬盘整个损坏。

正确使用计算机的方式应该是在计算机通电之后就绝对不要移动主机并避免震动到硬盘而导致整个硬盘数据发生问题。另外也不要随便将插头拔掉就以为是顺利关机，因为机械臂必须要归回原位，所以使用操作系统的正常关机方式才能够比较好的保养硬盘，因为这会让硬盘的机械臂归回原位。

所谓的硬盘分区就是告知操作系统这块硬盘可以存取的区域是由 A 柱面到 B 柱面的块，这样操

作系统才能够控制硬盘磁头去 A–B 范围内的柱面存取数据。如果没有告诉操作系统这个信息，操作系统就无法使用硬盘来进行数据的存取，因为操作系统将无法知道它要去哪里读取数据。

硬盘分区的重点就是指定每一个分区(Partition)的起始与结束柱面。事实上，MBR 就是在一块硬盘的第 1 个扇区上面，也是计算机开机之后要使用该硬盘时必须要读取的第一个区域，在这个区域内记录的就是硬盘里面的所有分区信息以及开机所用的引导加载程序写入的地方。

当一个硬盘的 MBR 坏掉时，由于分区的数据不见了，那么这个硬盘也就几乎可以说是报废了，因为操作系统不知道该去哪个柱面上读取数据。

MBR 最大的限制来自于它的大小不够大，无法存储所有分区与引导加载程序的信息，其中分区表仅有 64 Bytes，因此 MBR 仅提供最多 4 个 Partition 的存储，这就是所谓的 Primary(P)与 Extended(E)的分区最多只能有 4 个的原因，所以说如果预计分区超过 4 个，那么势必需要使用 3P + 1E，并且将所有的剩余空间都分给扩展分区才行，而且扩展分区最多只能有一个，否则只要 3P + E 之后还有剩下的空间将不能使用。

以上叙述的结论就是如果要进行硬盘分区并且已经预计规划使用完 MBR 所提供的 4 个分区(3P + E 或 4P)，那么硬盘的全部容量就需要使用完，否则剩下的容量也不能再被使用。不过如果仅是分区出 1P + 1E 的话，那么剩下的空间就还能再分区成两个主分区(Primary Partition)。

- 主分区与扩展分区最多可以有 4 个(硬盘的限制)；
- 扩展分区最多只能有 1 个(操作系统的限制)；
- 逻辑分区是由扩展分区持续分出来的分区；
- 能够被格式化后作为数据访问的分区为主分区（Primary）与逻辑分区（Logical），扩展分区(Extended)无法格式化；
- 逻辑分区的数量根据操作系统而不同，Linux 操作系统中的 IDE 硬盘最多有 59 个逻辑分区(5 号到 63 号)，SATA 硬盘则有 11 个逻辑分区(5 号到 15 号)。

4K 对齐是一种高级硬盘使用技术，用特殊方法将文件系统格式与硬盘物理层上进行契合，为提高硬盘寿命与高效率使用硬盘空间提供解决方案。因该技术将物理扇区与文件系统的每簇 4096 字节对齐而得名。

传统机械硬盘的每个扇区一般大小为 512 字节。当使用某一文件系统将硬盘格式化时，文件系统会将硬盘扇区、磁道与柱面统计整理并定义一个簇为多少扇区以方便快速存储。

例如：现时 Windows 中常见使用的 NTFS 文件系统，默认定义为 4096 字节大小为一个簇，但 NTFS 文件格式因为主引导记录占用了一个磁道共 63 个扇区，真正储存用户文件的扇区一般都在 63 号扇区之后，那么依照计算得出前 63 个扇区大小为：

$$512\text{B} \times 63 = 32256\text{B}$$

并按照默认簇大小得出 63 扇区为多少个簇：

$$32256\text{B} \div 4096\text{B} = 7.875$$

即为每 7 个簇后的第 8 个簇都会是跨越 2 个物理扇区并造成 3584 字节的空间浪费，该结果累计起来将会是个庞大数目，并长时间如此使用将会产生硬盘扇区新旧程度不同导致寿命下降。现时一般使用一些硬盘分区软件在主引导记录的 63 个扇区后作牺牲地空出数个扇区以对齐文件系统的 4096B 每簇，用以节约空间、均匀使用硬盘的各个部分以延长寿命并能避免过多的磁头读写操作，也一定程度上能提升读写速度。

在磁盘发展早期，每扇区为 512 字节比较适合当时硬盘的储存结构。但随着单盘容量的增加，储存密度的上升会明显降低磁头读取磁盘的信噪比，虽然可以用 ECC 校验保证数据可靠性，但消耗的空间会抵消储存密度上升带来的多余空间。所以提出了以 4KB 为一个扇区为主的改变。现时硬盘厂商新推出的硬盘，都将遵循先进格式化(4KB 扇区)的设计以对应新的储存结构和文件系统问题。

相对于机械硬盘来说 4K 对齐对于固态硬盘用意更大，现时的固态硬盘多为使用 NAND Flash 闪存作存储核心，该闪存是有删除写入次数限制的，当次数用完后该固态硬盘便会性能下降甚至报废。

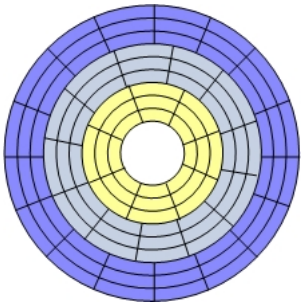


图 17.2: Zoned Bit Recording(ZBR)

很多厂商设计固态硬盘存储方式为不在短时间内删除写入同一个位置,尝试全面地均匀地使用每一个扇区以达到期望寿命,然而在没有 4K 对齐的电脑上这将会使固态硬盘寿命快速下降。

17.1 主机硬盘的规划

系统对于硬盘的需求跟主机开放的服务有关,那么除了这点之外还有就是数据的分类与安全性的考量。这里所谓的“数据安全”并不是指数据被网络入侵所破坏,而是指当主机系统的硬件出现问题时能安全保存用户文件。

另外,由于 Linux 是多用户多任务的环境,因此很可能主机上面已经有很多人的数据在其中了,所以硬盘的分区规划是相当重要的。事实情况是,硬盘的分区规划需要对于 Linux 文件结构有相当程度的认知之后才能够做比较完善的规划。

Linux 安装的过程中,至少要有两个 partition,一个是“/”,另一个则是交换分区“swap”。在预设的情况下,由于 Linux 的操作系统都在/usr/中,所以可以将/usr/设定的大一点,另外由于用户的信息都是在/home 下,因此这个也可以大一些,而/var 下是记录所有预设服务器的日志记录,而且 Mail 与 WWW 预设的路径也在/var,因此这个空间可以加大一些。下面的目录结构是比较符合容量大且(或)读写频繁的目录结构:

```
/
/usr
/home
/var
swap
```

一般来说,学习 Linux 最麻烦也最重要的地方就是安装,当然也可以利用类似 VMWare 的软件来学习 Linux,只是 VMWare 里面的硬件很多都是模拟的。

此外,在硬盘的分区方面建议先暂时以/及 swap 两个分区即可,而且还要预留一个未分区的空间。在熟悉 Linux 文件结构之后再对硬盘进行规划时,就要先分析主机的未来用途,然后根据用途去分析需要较大容量的目录,以及读写较为频繁的目录,将这些重要的目录分别独立出来而不与根目录放在一起,这样当读写频繁的分区有问题时,至少不会影响到根目录的系统数据,而且书局恢复也比较容易。

另一个容易出现问题的地方就是用户常常会找不到某些命令,导致无法按照书上的说明去执行某些命令。

17.2 大硬盘引起的开机问题

Linux 的开机程序“可能”会找不到 BIOS 提供的硬盘信息,这个不是 Linux 的问题,而是 BIOS 本身无法支持大硬盘的问题。

Linux 的核心会“取代 BIOS”而成功的检测到大硬盘，如果 Linux 内核顺利开机启动后，它会重新再去检测一次整个硬件而不理会 BIOS 所提供的信息。

虽然 BIOS 识别的磁盘容量不对，但是至少还可以读写整块硬盘的前面的扇区，因此可以根据这一点在磁盘的最前面分出一个 BIOS 能够识别的小分区，并将系统启动文件放入其中，一般是直接将开机分区安装在 1024 柱面以前，这样就可以避免 Linux 变成“可以安装，但是无法开机”的情况。

在进行安装的时候，规划出三个分区，分别是：

```
/boot
/
swap
```

/boot 只要给 100M Bytes 以内即可，而且/boot 要放在整块硬盘的最前面。

第 18 章

Partition

磁盘分区是使用分区编辑器(partition editor)在磁盘上划分出若干逻辑部分,磁盘一旦划分成数个分区(Partition),不同种类的目录与文件可以存储进不同的分区。越多分区,也就有更多不同的地方,可以将文件的性质区分得更细,按照更为细分的性质,存储在不同的地方以管理文件。合理的磁盘分区可以提高磁盘使用效率,因而必须要规划出大小适当的磁盘分区。

通过磁盘分区可以允许在一个磁盘上有多个文件系统,但是由于不同的操作系统所使用的文件系统(file system)并不相同,例如 Windows 所使用的是 FAT、FAT32、NTFS 格式,而 Linux 所使用的是 ext2/3/4 文件格式,这两种格式完全不相同。Linux 可以支持 Windows 的 FAT 文件格式,但是 Windows 无法读取 Linux 的文件格式。

- 系统一般单独放在一个分区,其他分区不会受到系统分区出现磁盘碎片的性能影响。
- 碍于技术限制(例如旧版的微软 FAT 文件系统不能访问超过一定的磁盘空间;旧的 PC BIOS 不允许从超过硬盘 1024 个柱面的位置启动操作系统)
- 如果一个分区出现逻辑损坏,不会影响到整块硬盘。
- 在一些操作系统(如 Linux)交换文件通常自己就是一个分区。在这种情况下,双重启动配置的系统就可以让几个操作系统使用同一个交换分区以节省磁盘空间。
- 避免过大的日志或者其他文件占满导致整个计算机故障,将它们放在独立的分区,这样可能只有那一个分区出现空间耗尽。
- 两个操作系统经常不能存在同一个分区上或者使用不同的“本地”磁盘格式。为了安装不同的操作系统,可以将磁盘分成不同的逻辑磁盘。
- 许多文件系统使用固定大小的簇将文件写到磁盘上,这些簇的大小与所在分区文件系统大小直接成比例。如果一个文件大小不是簇大小的整数倍,文件簇组中的最后一个将会有不能被其它文件使用的空闲空间。这样,使用簇的文件系统使得文件在磁盘上所占空间超出它们在内存中所占空间,并且越大的分区意味着越大的簇大小和越大的浪费空间。所以,使用几个较小的分区而不是大分区可以节省空间。
- 每个分区可以根据不同的需求定制。例如,如果一个分区很少往里写数据,就可以将它加载为只读。如果想要许多小文件,就需要使用有许多节点的文件系统分区。
- 在运行 Unix 的多用户系统上,有可能需要防止用户的硬连接攻击。为了达到这个目的,/home 和/tmp 路径必须与如/var 和/etc 下的系统文件分开。

磁盘分区可做看作是逻辑卷管理(Logical volume management,LVM)前身的一项简单技术。LVM 为计算机中的大量存储设备(Mass storage devices)提供更有弹性的硬盘分区方式。

> All problems in computer science can be solved by another level of indirection.——Da
> 计算机科学领域的任何问题都可以通过增加一个间接的中间层来解决。

LVM 作为一种抽象化存储技术,实现的方式会根据操作系统而有所不同。基本上,它是在驱动程序与操作系统之间增加一个逻辑层,以方便系统管理硬盘分区系统。

18.1 Windows

Microsoft Windows 的标准分区机制是创建一个分区 C:, 其中操作系统、数据和程序都在这个分区上。另外, 它推荐创建不同的分区或者使用不同的硬盘, 其中一个分区上存储操作系统; 而其它分区或者驱动器, 则供应用程序或者数据使用。

如果可能的话, 在不包含操作系统的硬盘上, 为交换文件创建一个单独的分区, 尽管这并不意味着两个硬盘都不会断电。在进行预分区工作之后, 很容易就可实现操作系统不存储在 C 分区上甚至是 C 分区根本就不存在。这样做有一些益处, 一些设计拙劣的病毒或者特洛伊木马将不能覆盖关键的系统文件或者控制系统。“我的文档”文件夹、“特殊文件夹”主目录可以加载到一个独立分区上以利用所有空闲空间。

18.2 UNIX

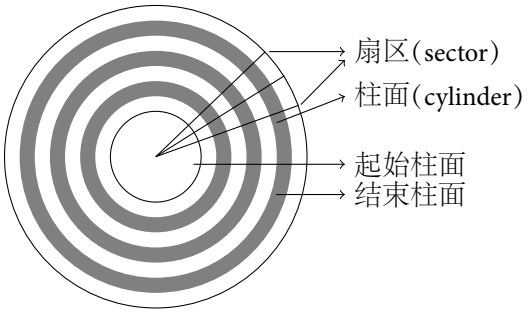
对于基于 UNIX 或者如 Linux 这样类似于 Unix 的操作系统来说, 分区系统创建了 /、/boot、/home、/tmp、/usr、/var、/opt 和交换分区。这就保证了如果其中一个文件系统损坏, 其它的数据 (其它的文件系统) 不受影响, 这样就减少了数据丢失。这样做的一个缺点是将整个驱动器划分成固定大小的小分区, 例如, 一个用户可能会填满 /home 分区并且用完可用硬盘空间, 即使其它分区上还有充足的空闲空间。

良好的实现方法要求用户预测每个分区可能需要的空间, 比如典型的桌面系统使用另外一种约定。

- “/”(根目录)分区包含整个文件系统和独立的交换分区。
- /home 分区对于桌面应用来说是一个有用的分区, 因为它允许在不破坏数据的前提下干净地重新安装(或者另外一个 Linux 发行版的更新安装)。

硬盘主要由盘片、机械臂、磁头和主轴马达等组成, 实际上硬盘是以 sectors(扇区)、cylinder(柱面)、partitions(分区)来作为存储的单位, 而最底层的实体硬盘单位就是 sectors, 通常一个 sector 大约是 512 bytes 左右, 不过在磁盘进行格式化时可以将数个 sector 格式化成为一个逻辑扇区(logical block), 通称为 block, blocks 为一个文件系统存取的最小量。

假如硬盘只有一个盘片, 那么盘片组成可以如下所示:



基本上, 硬盘是由最小的物理组成单位——扇区(sector)所组成, 数个扇区组成一个同心圆时就称为柱面(cylinder), 最后构成整个硬盘。

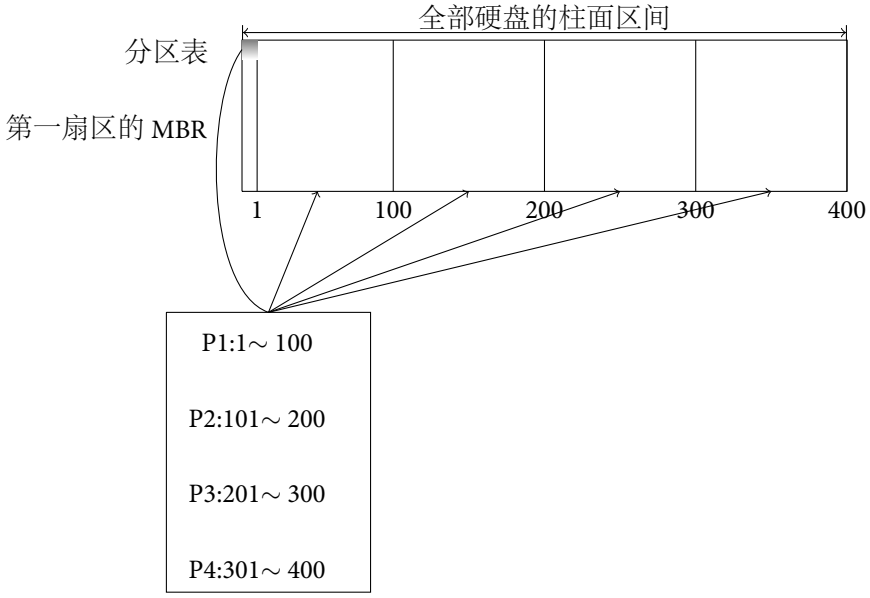
整块硬盘的第一个扇区特别重要, 它记录了整块硬盘的重要信息, 主要包括两个重要的信息, 分别是:

1、主引导分区(Master Boot Record, MBR): 可以安装引导加载程序的地方, 有 446bytes。

2、分区表(partition table): 记录整块硬盘分区的状态, 有 64bytes。

MBR 是很重要的, 系统在开机的时候会主动去读取这个区块的内容, 这样系统才会知道系统的安装位置以及应该如何开机, 尤其是需要通过多重引导系统来选择相应的操作系统的时候。

若将硬盘以长条形来看,然后将柱面以柱形图来看,那么这 64bytes 的分区表记录区段可以用下图表示:



至于 partition table,简单的说“硬盘分区”就是在修改这个 partition table 而已,操作系统访问硬盘是利用参考柱面号码的方式来进行的,基本上 partition table 就定义了“第 n 个硬盘区块是由第 x 柱面到第 y 个柱面”,所以每次当系统要去读取第 n 硬盘区块时,就只会去读取第 x 到 y 个扇区之间的数据。

不过,由于这个分区表的容量有限,在分区表所在的 64bytes 容量中,当初设计的时候就只设计成 4 组记录区,每组记录区记录了该区段的起始与结束的柱面号码,这些分区记录又被称为 Primary(主分区)及 extended(扩展分区),也就是说一块硬盘最多可以有 4 个(Primary + extended)的扇区,其中 extended 只能有一个,因此如果要分区成四个硬盘分区的话,那么最多就是可以按照:

P + P + P + P
P + P + P + E

的情况来分区了。

其中需要特别注意的是,如果上面的情况中,3P + E 只有三个“可用”的硬盘分区,如果要四个都“可用”,就得分区成 4P。extended 不能直接被使用,还需要分区成 Logical 才可以。

> 关于主分区、扩展分区与逻辑分区
> 1、主分区与扩展分区最多可以有 4 个(硬盘自身限制);
> 2、扩展分区最多只能有一个(操作系统自身限制);
> 3、逻辑分区是由扩展分区持续细分出来的分区;
> 4、能够被格式化后作为数据访问的分区为主分区和逻辑分区,扩展分区无法格式化;
> 5、逻辑分区的树林根据操作系统有所不同。
> 在 Linux 系统中,IDE 硬盘最多有 59 个逻辑分区(5 号到 63 号),SATA 硬盘则有 11 个逻辑分区(5 号到 15 号)。

假设上面的硬盘设备文件名为/dev/hda 时,那么这四个分区在 Linux 系统中的设备文件名如下所示,这里重点在于文件名后名会再接一个数字,这个数字与该分区所在的位置有关。

```
P1:/dev/hda1
P2:/dev/hda2
P3:/dev/hda3
P4:/dev/hda4
```

那么为什么要有 extended? 这是因为如果我们要将硬盘分成 5 个分区的话,就需要 extended。由于 MBR 仅能保存四个 partition 的数据记录,超过 4 个以上时系统允许在额外的硬盘空间放置另一份硬盘分区信息,那就是 extended。

假设将硬盘分区成为 3P + E,那么那个 E 其实是告诉系统,硬盘分区表在另外的那份 partition table 中,也就是说,那个 extended 其实就是“指向(point to)”正确的那个额外的 partition table。

本身 extended 是不能在任何系统上面被使用的,还需要再额外的将 extended 分区成 Logical(逻辑)分区才能被使用,所以通过 extended 就可以分区超过 5 个可以利用的 partition。

不过,在实际的分区时还是容易出现问题的,下面我们来思考看看:

思考一:如果要将硬盘“暂时”分区成四个 partition,同时还有其他的空间可以在以后进行规划,那么该如何分区?

说明:

Primary + extended 最多只能有四个 partition,而如果要超过 5 个 partition,那就需要 extended,因此在这个例子中,千万不能分区成四个 Primary ,假如 20 GB 的硬盘,而 4 个 Primary 共用去了 15 GB,那么剩下的 5 GB 将完全不能使用,这是因为已经没有多余的 partition table 记录区可以记录了,因此也就无法进行额外的分区,当然空间也就被浪费了。

因此,如果要分区超过 4 个分区以上时,记得一定要有 extended 分区,而且必须将所有剩下的空间都分配给 extended,然后再以 Logical 的分区来规划 extended 的空间。另外考虑到硬盘的连续性,一般建议将 extended 的分区放在最后面的柱面内。

思考二:可不可以仅分区 1 个 Primary 与 1 个 extended 呢?

可以!

说明:

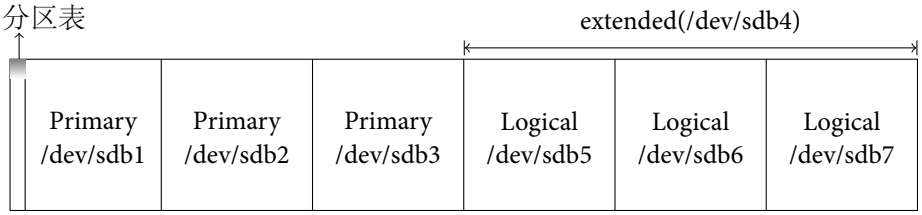
基本上,Logical 的号码可达 63,因此可以仅分一个主分区并且将所有其他的分区都给 extended,利用 Logical 分区来进行其他的 partition 规划,这也是早期 Windows 操作系统的操作方式。

此外,逻辑分区的号码在 IDE 可达 63 号,SATA 可达 15 号,因此仅分区为 1 个 Primary 和 1 个 extended 时,可以通过 extended 分区继续分出 Logical 分区。

思考三:如果要将 SATA 硬盘分区成 6 个可以使用的硬盘分区,那么每个硬盘在 Linux 下面的代号是什么?

说明:

由于硬盘在 Primary + extended 最多可以有 4 个,因此在 Linux 下已经将 partition table 1~4 先留下来了,如果只用了 2 个 P + E 的话,那么将会空出两个 partition number。具体来说,假设将四个 P + E 都用完了,那么硬盘的实际分区会如下图所示:



针对安装的硬盘的类型,实际可以使用的是:

SATA:

/dev/sda1,/dev/sda2,/dev/sda3,/dev/sda5,/dev/sda6,/dev/sda7

IDE:

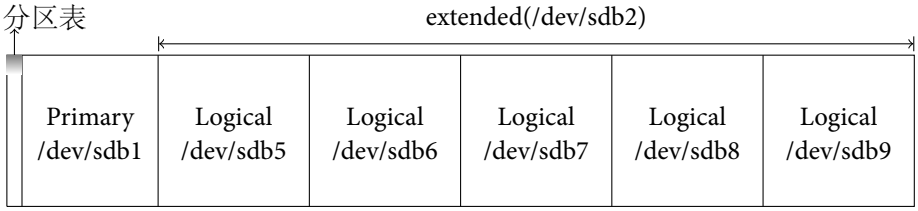
/dev/hda1,/dev/hda2,/dev/hda3,/dev/hda5,/dev/hda6,/dev/hda7

这六个 partition。

至于/dev/sda4(/dev/hda4)这个 extended 分区本身仅是用来规划出来让 Logical 可以利用的硬盘空间。

其实在每个 partition 的最前面扇区都会有一个特殊的区块，称为 super block，extended 指向的就是/dev/hda4 的 super block 处，该处就是额外记录的那个 partition table。

如果只想要分区 1 个 Primary 与 1 个 extended，这个时候的硬盘分区会变成如下所示：



注意到了吗？因为 1 ～ 4 号已经被预留下来了，所以第一个 Logical 的代号由 5 号开始计算，而后面在被规划的就以累加的方式增长，其中/dev/sda3，/dev/sda4 这两个代号则是空的。

举例来说，假设硬盘总共有 1024 个 cylinder（利用 blocks 结合而成的硬盘计算单位），那么在硬盘的起始柱面（就是磁盘分区表，可以想象成要读取一块硬盘时最先读取的地方），如果写入 partitions 共有两块，一块是 Primary，一块是 extended，而且 extended 也只规划成一个 Logical，那么硬盘就是只有两个分区（对于系统来说，真正能使用的有 Primary 与 Logical 的扇区，extended 并无法直接使用的，需要再加以规划成为 Logical 才行），而且在 partition table 中也会记录 Primary 是由“第 n_1 个 cylinder 到第 n_2 个 cylinder”，所以当系统要去读取 Primary 时，就只会在 $n_1 \sim n_2$ 之间的实体硬盘区域活动。

基本上，硬盘分区时仅支持一个 Primary 与一个 extended，其中 extended 可以再细分成多个 Logical 的硬盘分区。Linux 基本上最多可以有 4 个 Primary 的硬盘，可以支持到 3 个 Primary 与一个 extended，其中 extended 若再细分成 Logical 的话，则全部 Primary + extended + Logical 应该可以支持到 64 个（针对 IDE 硬盘而言）和 16 个（针对 SATA 硬盘而言）。

18.2.1 硬盘分区步骤

硬盘分区主要可分为下面几个步骤：

1、将旧有的分区表删除；

2、建立新的主分区（Primary）及扩展分区（extended）；

3、保存分区表；

4、以工具格式化已分区的硬盘。

删除分区之后硬盘中就没有分区表的存在了，所以这个硬盘的系统种类就变成了未分区。

分区表被删除后，重新分区时需要确定分区类型是主分区还是扩展分区。主分区在 Microsoft Windows 系统下就是 C 盘，其他的是扩展分区并非逻辑分区。注意，所谓的“逻辑分区”是包含在扩展分区中的，可以使用逻辑分区将扩展分区分为几个分区，这些新分区就是“逻辑分区”。

分区时通常在第一步是输入“启始柱面”，然后会要求输入“结束柱面”，结束柱面的输入方法有两种模式，一种是输入柱面区，一种是输入所需要的 MB 数，通常是输入 MB 数。

> 关于硬盘分区：
> 1、其实所谓的“分区”只是针对那个 64bytes 的分区表进行设置而已；
> 2、硬盘默认的分区表仅能写入四组分区信息；
> 3、这四组分区信息被称为主（Primary）或扩展（extended）分区；
> 4、分区的最小单位为柱面（cylinder）；
> 5、当系统要写入硬盘时，一定会参考分区表才能针对某个分区进行数据处理。

18.3 Linux

安装 Linux 时可选的分区方式[?] 包括：

1. 删除硬盘上的所有分区,并建立自动 Linux 默认分区表。
2. 删除所选硬盘上已有的 Linux 分区,并自动建立 Linux 默认分区表。
3. 使用硬盘上所剩余的自由空间自动建立 Linux 分区表。
4. 自定义分区。

当硬盘上不存在任何操作系统或是想要删除现有操作系统,并且只想安装 Linux 系统的话,可以选择第一种方式进行分区,它会删除现有的所有分区,并自动建立一套 Linux 分区。

如果当前硬盘上已经安装了一份 Linux 系统,并且想要覆盖该系统的话,可以选择第二种方式,安装程序同样会删除现有的 Linux 分区并自动建立一套 Linux 分区。

如果硬盘上面还有未分配的硬盘空间的话,你可以选择第三种方式,安装程序不会修改现有分区,而会在未分配的自由分区自动建立一套 Linux 分区。

如果对 Linux 分区非常熟悉,或者需要自定义分区大小,可以选择第四种方式。在选择了自定义分区后,那么下一步就需要手动分配磁盘分区了,Linux 可以分为这样几个分区:

- /(根分区);
- /boot(启动分区);
- /home(家目录分区);
- /tmp(临时文件分区);
- /usr(程序分区);
- /var(日志分区);
- swap(虚拟内存,又称交换分区)。

其中必须要有的是/(根分区)和/boot(启动分区),此外的/home、/tmp、/usr、/var 可以单独建立分区,也可以不建立分区。如果/home、/tmp、/usr、/var 不建立分区,它们会以文件夹的形式自动挂载到/(根分区)下。

- /分区是每个 Linux 系统必须要有的分区,根分区/是 Linux 文件系统的起点。
- /boot 分区是存放启动文件的,也是必须要有的分区。
- /home 分区是存放用户个人文件的地方,可以视用户文件的多少、大小而确定该分区的大小。
- 在 Linux 中绝大多数程序默认是安装在/usr 下面的,/usr 分区又称为程序分区。
- /var 是用来存放系统日志的地方。
- /tmp 是用来存放系统临时文件的地方。
- swap 分区是虚拟内存分区。

第 19 章

Formatting

格式化是指对磁盘或磁盘中的分区(partition)进行初始化的一种操作,这种操作通常会导致现有的磁盘或分区中所有的文件被清除。

格式化通常分为低级格式化和高级格式化。如果没有特别指明,对硬盘的格式化通常是指高级格式化,而对软盘的格式化则通常同时包括这两者。

- 低级格式化处理碟片表面格式化赋予磁片扇区数的特质;
- 低级格式化完成后,硬件碟片控制器(disk controller)即可看到并使用低级格式化的成果;
- 高级格式化处理“伴随着操作系统所写的特定信息”。

19.1 Low-Level Formatting

低级格式化(Low-Level Formatting)又称低层格式化或物理格式化(Physical Format),对于部分硬盘制造厂商,它也被称为初始化(initialization)。

最早,伴随着应用 CHS 编址方法、频率调制(FM)、改进频率调制(MFM)等编码方案的磁盘的出现,低级格式化被用于指代对磁盘进行划分柱面、磁道、扇区的操作。现今,随着软盘的逐渐退出日常应用,应用新的编址方法和接口的磁盘的出现,这个词已经失去了原本的含义,大多数的硬盘制造商将低级格式化(Low-Level Formatting)定义为创建硬盘扇区(sector)使硬盘具备存储能力的操作。

现在,人们对低级格式化存在一定的误解。多数情况下,低级格式化往往是指硬盘的填零操作,而且“非必要”的情况下尽量不对硬盘进行低级格式化。

对于一张标准的 1.44 MB 软盘,其低级格式化将在软盘上创建 160 个磁道(track)(每面 80 个),每磁道 18 个扇区(sector),每扇区 512 位位组(byte);共计 1,474,560 位组。需要注意的是:软盘的低级格式化通常是系统所内置支持的。通常情况下,对软盘的格式化操作即包含了低级格式化操作和高级格式化操作两个部分。

19.2 High-Level Formatting

高级格式化又称逻辑格式化,它是指根据用户选定的文件系统(如 FAT12、FAT16、FAT32、NTFS、EXT2、EXT3 等),在磁盘的特定区域写入特定数据,以达到初始化磁盘或磁盘分区、清除原磁盘或磁盘分区中所有文件的一个操作。高级格式化包括对主引导记录中分区表相应区域的重写、根据用户选定的文件系统,在分区中划出一片用于存放文件分配表、目录表等用于文件管理的磁盘空间,以便用户使用该分区管理文件。

在 DOS 环境下,有多种软件可以执行格式化的操作,系统通常也以外部命令的形式提供一个命令行界面的格式化软件“Format”。

Format 命令的参数包括将被执行格式化的磁盘,以及一些其他次要参数,如簇的大小、文件系统的格式等。

Format 命令通常的格式是:

```
1 Format X:
```

X 为所希望被执行格式化操作的盘符，如希望格式化 C 盘，则将 X 替换为 C，如此类推。加入 “Q” 参数可以执行快速格式化。

在 Windows 环境下，格式化的操作相对简单。除了可以使用图形化的操作界面执行格式化操作之外，也可以使用命令行的方式进行操作，具体的方法与 DOS 环境下类似。不过对硬盘执行格式化操作时，用户需要拥有系统管理员权限。

在 Unix/Linux 环境下，通常使用 mkfs 命令执行格式化操作，mkfs 命令需要的参数有设备路径和文件系统格式等。对硬盘执行格式化操作时，用户同样需要拥有 root 权限。

19.3 Advanced Format

先进格式化（Advanced Format）是 IDEMA（International Disk Drive Equipment and Materials Association）于 2009 年 12 月 [1] 制定的硬盘格式化标准。IDMEA 在 2005 年与 Hitachi、Seagate、WD、LSI、Intel、Microsoft、Dell、HP、Lenovo 等硬软件厂商制定出 1024 字节、2048 位组和 4096 字节三种容量扇区配置，先进格式化是规范中的 4096 字节（4KB）配置。

从 2011 年 1 月 1 日起，硬盘厂商新推出的硬盘时，都将遵循先进格式化（4KB）的设计。

在磁盘发展早期，每扇区为 512byte 比较适合当时硬盘的储存结构。但随着单盘容量的增加，储存密度的上升会明显降低磁头读取磁盘的信噪比，虽然可以用 ECC 校验保证数据可靠性，但消耗的空间会抵消储存密度上升带来的多余空间。所以提出了以 4kbyte 为一个扇区为主的改变。

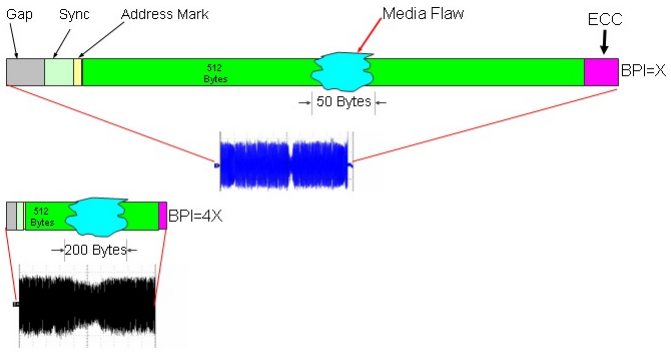


图 19.1: 512byte 和 4Kbyte 扇区受到物理污染时所产生的电磁信号影响示意

最主要的好处减少 ECC 的占用和提升 ECC 校验效率。因为 512byte 扇区需要另外 40byte 作为 ECC 校验空间，而 4kbyte 扇区只需要 100byte，所以，同样提供 4kbyte 扇区空间，使用先进格式化能节约出 220byte 的储存空间，而且能令 ECC 校验完成更多空间的检验纠错，提高 ECC 校验的效率。

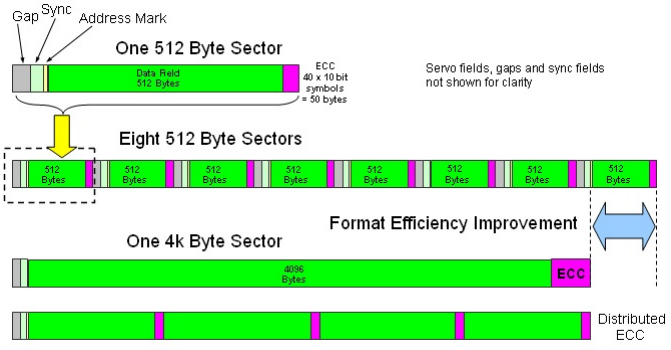


图 19.2: 512byte 和 4Kbyte 的物理空间占用比较示意

另外，通常在 x86 架构下的内存分页容量为 4KB，而且很多磁盘文件系统（如 NTFS、ext3、HFS+ 等）的簇容量也为 4KB，而如果使用 4KB 为一个扇区，硬盘对一个扇区的读写数据量刚好装满一个内存页或对应文件系统分区的一个簇，能避免过多的磁头读写操作，一定程度上能提升读写速度。

现在推行主要问题为 Windows 5.x 核心系统（Windows 2000、Windows XP、Windows Server 2003）读取分区无法对准扇区而读取出错和文件系统，簇横跨多个扇区造成转换延迟影响随机写入性能。除外一些较旧版本的磁盘管理工具在不支持 4Kbyte 扇区的情况下也会发生类似的情况。WD 提供了固件模拟和工具校正的方法（WD Align 程序）临时解决，但最根本的解决为升级原生支持 4Kbyte 扇区的 Windows 6.x 核心操作系统，如 Windows Vista、Windows 7。

较新的 Linux、Mac OS X 由于较早开始对 4kbyte 扇区的支持，所以基本能不做调整就能直接使用先进格式化后的硬盘。

第 20 章

Installaton

在规划硬盘分区安装 Linux 时，如果硬盘大于 60GB 时可能会出现找不到启动分区的问题，那就必须要独立出/boot 这个分区。

20.1 规划硬盘分区

空间管理、访问许可与目录搜索的方式，依属于安装在分区上的文件系统。当改变大小的能力依属于安装在分区上的文件系统时，需要谨慎地考虑分区的大小。

实际上在 Linux 安装的时候，已经提供了相当多的默认模式让用户选择分区的方式，不过分区的行为可能都不是很符合用户主机的需求，毕竟每个人的“想法”都不太一样。强烈建议使用“自定义安装，Custom”这个安装模式。在某些 Linux distribution 中会将这个模式称为“Expert，专家模式”。

20.1.1 目录树结构：Directory tree

Linux 内的所有数据都是文件，整个 Linux 系统最重要的地方就是目录树结构。目录树结构就是以根目录为主，然后向下呈现分枝状目录形式的一种文件结构。

整个目录树结构最重要的就是根目录（root directory），其表示方法为一条斜线“/”，所有的文件都与目录树有关，所有的文件都是由根目录（/）衍生来的，而次目录下还能够有其他的数据存在。

Linux 系统使用的是目录树结构，而实际上文件数据是存放在磁盘分区中的，而如果要理解目录树结构与磁盘内的数据的关系，就要引入“挂载（mount）”的概念。

20.1.2 目录树与文件系统的关系

所谓的“挂载”就是利用一个目录当成进入点，将磁盘分区的数据放置在该目录下，也就是说，进入该目录就可以读取该分区，这个操作就称为“挂载”，那个进入点的目录称为“挂载点”。Linux 系统最重要的就是根目录，因此根目录一定要挂载到某个分区，至于其他的目录则可以根据用户自己的需求挂载到其他不同的分区。

另外，默认情况下 Linux 是将光驱的数据存放到/media/cdrom 里，但光驱也可以被用户挂载到其他目录。

20.1.3 挂载点与磁盘分区的规划

在安装 Linux 系统时就得要规划好磁盘分区与目录树的挂载，初次接触 Linux 时可能只需要最简单的分区方式，即只要分区“/”及“swap”即可，其中直接以最大的分区“/”来安装系统。

另外，/usr/是 Linux 的可执行程序及相关的文件目录。建议分区时预留一个备份的分区。由于 Linux 默认的目录是固定的，所以通常我们会将/var 及/home 这两个目录稍微加大一些。另外建议可以预留一个分区来备份系统核心与脚本（scripts），当 Linux 重新安装的时候，一些文件马上就可以直接在硬盘当中找到。

另外，Linux 安装之前会自动的把硬盘格式化，而且硬盘至少需要 2GB 以上才可以选择“Server”安装模式。

目前几乎所有的 Linux distributions 都是支持光盘启动的，只是必须要确定系统的第一个启动设备为光驱，可以在 BIOS 里面设定引导设备的次序。

如果使用光盘启动时发生错误，很可能是由于硬件不支持，建议再仔细地确认一下硬件是否有超频或者其他不正常的现象，另外安装光盘来源也需要确认。

一般 Linux 都会支持至少两种安装以上的安装模式，分别是文字（text）模式与图形（graphic）界面。

若想要以文字界面来安装，可以在启动时输入“linux text”来让安装程序以命令行模式安装。不过要注意的是，如果在 10 秒左右没有在 boot: 后输入任何提示的话，安装程序就会以默认的图形界面来安装。

硬件检测完毕之后会出现一个是否校验光盘的画面，如果要检查光盘的话会花去很多时间，所以如果确定光盘来源没有问题，可以直接跳过该选项即可。

略过光盘校验工作后，由于使用的是图形界面的安装模式，安装程序就会去检测显示器、键盘、鼠标等相关的硬件。

在完成了一些硬件方面的检测之后就可以进入图形界面的安装。基本上，图形安装界面分为左右两个区域，左边主要是作为“说明”之用，右边才是真正的操作区域。

安装程序可以使用不同的语言，可以选择相应的语言来进行安装。因为每个地区的键盘上面的字母设置都不一样，如果使用英文的键盘配置，可以选择“English（United States）”。

如果主机用来进行软件开发，就需要安装“系统开发工具”，这样就可以把 gcc、kernel-headers、kernel-source 等安装到主机上。

基本上，Linux distribution 为用户规划好一些主机使用模式了，举例来说，如果想要使用 Desktop 型计算机的功能，那么可以选择“个人计算机”项目，它会主动的进行相应的硬盘分区以及相关的软件选择。不过可能硬盘 partition 就交给系统主动去判断处理，而且系统的默认分区与套件的选择也不见得就会跟用户需求一致，因此建议务必选择“自定义安装”。

可以使用 Linux distribution 附带的 Disk Druid 工具来进行硬盘分区。注意，如果是新硬盘，可能会发生错误告知用户安装程序找不到 partition table。接下来的画面则是在操作硬盘分区的主要画面，这个画面主要分为三大区块，最上方为硬盘的分区示意图，目前因为硬盘并未分区，所以呈现的就是一整块而且为 Free 的字样。中间则是命令区，下方则是每个分区（partitions）的设备文件名、挂载点与目录、文件系统类型、是否需要格式化、分区所占容量大小以及开始与结束的柱面号码等。

至于命令区，总共有六大区块，其中 RAID 与 LVM 是硬盘特殊的应用，命令的作用如下：

（1）“新增”是指增加新分区，也就是建立新的硬盘分区；

（2）“编辑”是指编辑已经存在的硬盘分区，可以在实际状态显示区选择想要修改的分区，然后再选择“编辑”即可进行该分区的编辑动作。

（3）“删除”是指删除一个硬盘分区。同样地，要在实际状态显示区点击选择想要删除的分区。

（4）“重设”则是恢复最原始的硬盘分区状态。

20.2 创建磁盘分区

20.2.1 创建根目录分区

创建根目录“/”分区时，默认使用 Linux 提供的文件系统格式，比如 ext2/3/4。挂载点的选择可以手动输入或者使用系统默认，此外这里要说明 Linux 支持的文件系统类型包括：ext2/3/4、physical volume（LVM）、software RAID、swap、vfat 等。

- ext3：ext3 比 ext2 文件系统多了日志的记录，在系统恢复时比较快。
- physical volume（LVM）：这是用来弹性调整文件系统大小的一种机制，可以让文件系统大小变大或变小而不改变原有的文件数据的内容。
- software RAID：利用 Linux 系统的特性，通过软件模拟仿真动态磁盘阵列的功能。

- swap：内存交换空间，由于 swap 并不会使用到目录树的挂载，所以用 swap 就不需要指定挂载点。
- vfat：同时被 Linux 和 Windows 所支持的文件系统类型，如果有数据交换的需要，可以构建一个 vfat 的文件系统。

20.2.2 创建 boot 分区

如果将/boot 独立分区，务必要把该分区放在整块硬盘的最前面，因此针对/boot 就要选择“强制为主分区”选项。

20.2.3 创建内存交换分区

简单的说，内存交换分区（swap）可以被看作是“虚拟内存”，它不需要挂载点，因此分区时就就没有“/”挂载点前缀。如果物理内存不足，当系统负荷突然之间加大时，此时可以使用硬盘的 swap 分区来模拟内存的数据存取。不过，虚拟内存的速度会比较慢。当有数据被存放在物理内存里，但是这些数据又不是常被 CPU 所取用时，那么这些不常被使用的程序将会被放到虚拟内存当中，而将速度较快的物理内存空间释放出来给真正需要的程序使用。

在传统的 Linux 说明文件中，通常 swap 建议的值大约是“内存的 1.5 到 2 倍之间”，但是这个数值也需要根据实际情况而定，比如说如果系统内存达到 4G 以上时，swap 也可以不必额外设置。

20.2.4 创建/home 目录分区

接下来就是创建/home 目录分区。

20.3 针对笔记本电脑与其他类 PC 计算机的参数

笔记本电脑与一般计算机略有不同,它加入了非常多的省电机制或者是其他硬件的管理机制,因此在安装 Linux 操作系统过程加载内核时不要加载一些功能,于是就要输入下面这些参数:

```
boot:linux nofb apm=off acpi=off pci=noacpi
```

其中 apm(Advanced Power Management)是早期的电源管理模块, acpi(Advanced Configuration and Power Interface)则是近期的电源管理模块。这两者都是硬件本身就提供支持的,但是笔记本电脑本身可能不需要这些功能,因此安装系统时如果启动这些机制就会造成一些错误,导致无法顺利安装。

如果笔记本电脑的显卡是集成的,就需要使用 nofb 来取消显卡的缓冲存储器检测,这样 Linux 安装程序就不会去检测集成显卡模块。

上面这些在安装系统时所输入的参数就称为“内核参数”。

20.4 配置引导加载程序

完成硬盘分区后接下来就来选择引导加载程序,在 Linux 里面主要有 LILO 与 GRUB 两种引导加载程序,不过目前 LILO 已经较少使用,取而代之的就是 GRUB。值得注意的是,引导加载程序可以被安装在 MBR 也可以安装在每个 partition 最前面的 super block 处。

如果安装在/dev/hda 内,那就是 MBR 的安装点,如果是类似/dev/hda1 这个就是 super block 的安装点。在 GRUB 中可以通过“新增”、“编辑”与“删除”来管理启动菜单要显示的项目。

举例来说,如果已经把 Microsoft Windows 安装在当前系统中,就可以通过“新增”项目操作将 Windows 启动分区加到当前启动菜单中,从而实现多系统引导启动。

多系统启动是有很多风险存在的,而且也不能随时更改多重操作系统的启动分区。

如前所述,GRUB 接下来就会去读取内核文件来进行硬件检测,并加载适当的硬件驱动程序,随后便开始启动操作系统的各项服务等。

GRUB 引导加载程序还提供密码管理机制,虽然这会影响到远程管理主机。另外,如果有特殊需求,可以把 GRUB 安装到每个分区的启动扇区(boot sector)。

20.4.1 单硬盘多系统引导

多系统引导时要特别注意:

(1)Windows 的环境中最好将 Linux 的根目录与 swap 取消挂载,否则在 Windows 环境中可能会提示用户格式化这些分区。

(2)Linux 不能随意删除,开机时 GRUB 会去读取 Linux 根目录下的/boot 目录内容,一旦删除 Linux 会导致同时 Windows 也就无法启动了,因为此时整个开机启动菜单都会丢失。

20.4.2 多硬盘多系统引导

多个硬盘就会有多个 MBR,这时就需要在 BIOS 中调整引导开机的默认设备顺序,只有第一个可引导开机设备内的 MBR 会被主动读取。

理论上是不能把 Windows 的引导加载程序安装到/dev/hda(或/dev/sda)而将 Linux 安装到/dev/hdb(或/dev/sdb)上,而是必须把引导加载程序安装到指定的第一个引导开机硬盘上。

由于 SATA 类型的设备文件名是利用检测到顺序来决定的,与 SATA 插槽无关,因此建议此时 BIOS 使用确定好的开机顺序,然后由 GRUB 来控制全部的开机菜单,否则对于 Linux 的运行会产生影响。

20.5 网络设定

如果网卡可以被系统检测到就可以设定网络参数，目前各大版本几乎都会默认网卡 IP 的获取方式为“自动取得 IP”，也就是所谓的“DHCP”网络协议。不过，由于这个协议需要有 DHCP 主机的辅助，搜索的过程中可能会等待一段时间，因此也可以改成手动设定。不过无论如何，都要与网络环境相同才是。

20.6 防火墙与 SELinux

操作系统的防火墙是一套可以设置允许或拒绝从其他机器上访问主机系统的服务，同时可以防止来自外界的，未经验证的系统对主机系统的访问。

另外 SELinux 的设定值得特别留意，SELinux 是 Security Enhanced Linux 的简写，由美国国家安全局 NSA（National Security Agency，http://www.nsa.gov/selinux/）开发。

现在 SELinux 被集成到 Linux 内核当中，它并不是防火墙，其主要功能是管理整个 Linux 系统的访问权限控制（access control），可以避免一些可能造成 Linux 操作系统安全问题（Security）的软件的破坏。

最早之前，SELinux 的开发是有鉴于系统经常被一般用户误操作而造成系统数据的安全性问题，因此加上这个模块来防止系统被终端用户不小心滥用系统资源。

SELinux 具有较好的系统防护能力，不过如果不熟悉，那么启用 SELinux 后系统服务可能会因为这个较为严密的安全机制而导致无法提供联网服务，或者无法进行数据存取。

当内核出现错误时，使用 dump 可以将当时的内存内的信息写到文件中，从而帮助内核开发者改进内核。

20.7 时区选择

全世界被细分为 24 个时区，所以要告知系统具体时区，可以根据地区选择，也可以直接用鼠标在地图上选择。

要特别注意的是“UTC”，UTC 与所谓的“日光节约时间”有关。不过一般不需要选择这个，不然的话还可能造成时区被影响，导致系统显示的时间会与本地时间不同。

> Tips：
> 事实上，UTC 与 GMT 时间是一样的，GMT 就是格林威治时间——也是标准的地球时间。以格林威治（英国）所在地为 GMT 0 点，将地球细分为 24 个时区。

20.8 root 密码

Linux 的系统管理员的默认名称为 root，root 的密码最好设定的严格一点，至少 8 个字符以上，并且含有特殊符号。

在操作 Linux 系统时，除非必要，否则不要使用 root 账户，由于 root 权限最大，随时有可能因为误操作导致整个系统故障，因此良好的系统管理习惯是创建一个一般身份账户，当需要额外的 root 权限时才使用身份切换命令成为 root 来管理维护。

另外，所有安装的信息都会被记录在/root/install.log 及/root/anaconda-ks.cfg 这两个文件中。

20.9 关于大硬盘导致无法启动的问题

可能在第一次安装完 Linux 后却发现无法启动的问题，也就是说确实安装好了 Linux distribution，但就是无法顺利启动，只要重新启动程序就会出现类似下面的画面：

```
GRUB> _
```

然后等待输入一些数据，如果发生了这样的问题，那么可能的原因有：

(1)主板 BIOS 太旧，导致识别不到大硬盘；

(2)主板不支持大硬盘。

这时首先需要更新 BIOS，将硬盘的 cylinders、heads、sectors 在 BIOS 内手动设定。更简单的解决方法就是重新安装 Linux 并且在分区时建立一个 100MB 左右的分区，然后将其挂载到/boot。要注意/boot 必须要在整个硬盘的最前面，例如，必须是/dev/hda1 才行。

至于会产生这个问题的原因确实是与 BIOS 支持的硬盘容量有关，处理方法虽然比较麻烦，不过也只能这样做了。

第 21 章

Booting

在计算机中，CMOS 与 BIOS 是特别要注意的两个概念，其中 CMOS 是记录各项硬件参数且嵌入在主板上的只读存储器（ROM），BIOS 则是一个写入到主板中的固件（firmware，就是写入到硬件中的一个软件程序），BIOS 也就是在开机的时候计算机系统会主动执行的第一个程序。

接下来 BIOS 会去分析计算机里面有哪些存储设备，比如 BIOS 会依据用户的设置去搜索硬盘，并且到硬盘里面去读取第一个扇区的 MBR 位置。

MBR 这个仅有 446bytes 的硬盘容量里存放有最基本的引导加载程序，接下来的工作就由 MBR 内的引导加载程序来接着工作。引导加载程序的作用是加载（load）操作系统内核文件，而且由于引导加载程序是操作系统在安装时提供的，所以它会识别硬盘内的文件系统格式，于是就能够读取操作系统内核文件，然后接下来就是操作系统内核文件的工作了。

简单的说，整个开机流程到操作系统之前的过程应该是如下这样的：

1、BIOS：开机主动执行的固件，BIOS 会识别第一个可引导启动的设备。

2、MBR：第一个可引导开机设备的第一个扇区内的主引导分区，包含引导加载程序。

3、引导加载程序（Boot Loader）：可读取内核文件来执行载入操作系统内核的软件。

4、操作系统内核文件：逐步开始操作系统的各项功能。

其中，BIOS 和 MBR 都是硬件本身会支持的功能，Boot Loader 则是操作系统安装在 MBR 的软件，它的主要任务包括如下：

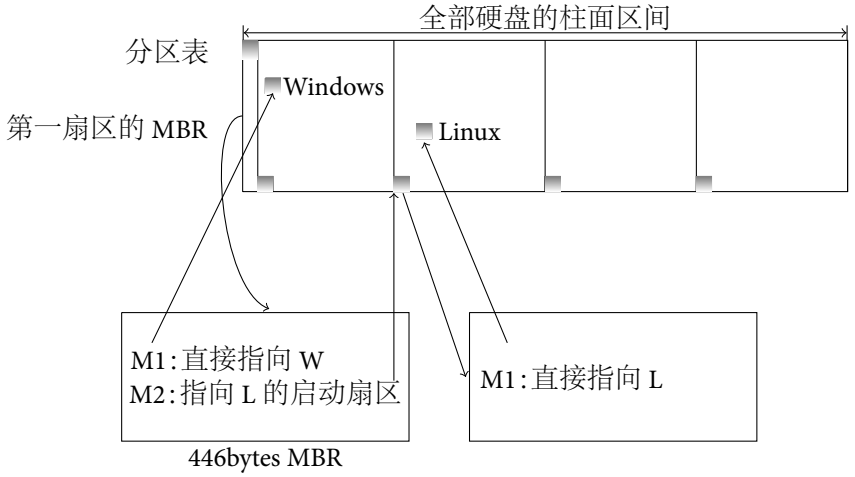
1、选择开机菜单：用户可以根据不同的开机选项来实现多重引导。

2、载入操作系统内核文件：直接指向可引导启动的程序区段来载入操作系统。

3、转交任务给其他 Boot Loader：将引导加载功能转交给其他 Booter Loader。

最后一项说明计算机系统中可以安装两个以上的引导加载程序，而且引导加载程序除了可以安装在 MBR 之外，还可以安装在每个分区的引导扇区（Boot Sector），这个特色才提供了“多重引导”的功能。

举例来说，假设计算机上安装了 Windows 及 Linux 操作系统，而且 MBR 内安装的是可同时识别 Windows 和 Linux 操作系统的引导加载程序，那么整个流程可用下图表示。



在上图中，MBR 的引导加载程序提供两个菜单，菜单一（M1）可以直接加载 Windows 系统的内核文件来开机；菜单二（M2）则是将引导加载工作交给第二个分区的启动扇区（Boot Sector）。当用户在开

机的时候选择菜单二时，整个引导加载工作就会交给第二分区的引导加载程序了。当第二个引导加载程序启动后，该引导加载程序内仅有一个开机菜单，因此就能够使用 Linux 的内核文件来开机，这就是多重引导的工作情况。通过上面陈述可以得出如下结论：

1、每个分区都拥有自己的启动扇区（Boot Sector）；

2、实际可开机的内核文件是可以放置到各分区内的；

3、Boot Loader 只会认识自己的系统分区内的系统内核文件，以及其他 Boot Loader 而已；

4、Boot Loader 可以直接指向或者间接将管理权转交给另一个启动加载程序。

Linux 系统在安装的时候可以选择将引导加载程序安装在 MBR 或个别分区的启动扇区，而且 Linux 的 Boot Loader 可以手动设置系统启动菜单，因此可以在 Linux 的 Boot Loader 中加入 Window 开机选项。而 Windows 在安装的时候，其安装程序会主动覆盖掉 MBR 以及自己所在分区的启动扇区，因此就没有选择的机会，这时可以使用 Linux 的系统恢复光盘来修复 MBR。

> Tips:
> 要注意区分引导加载程序与启动扇区的概念，现代计算机操作系统开机时需要有引导加载程序，而且引导加载程序可以安装在 MBR 及启动扇区等不同地方。

在 Linux 里面默认使用两种引导加载程序，分别是 LILO 与 GRUB，其中 LILO 是比较早期的引导加载程序，LILO 中的硬盘代号设置与 Linux 的硬盘代号相同。较新的 GRUB 最大的功能也最具魅力的地方是具有“动态搜索核心文件”的功能，GRUB 可以让用户在搜索的同时自行编辑启动设置文件。

硬盘分区与配置的好坏会影响到未来主机的使用情况，此外好一点的分区方式会让用户数据具有一定的安全性。除此之外硬盘分区的好坏还可以影响到系统存取数据的效率。

正常情况下的 Linux 主机通常会依照目录与主机的特性来进行硬盘分区。不过，Linux 的硬盘分区比较弹性，而且 Linux 的硬盘分区程序 fdisk 功能也比较强大，但是要说明的是，如果要分区合理，必须要理解基础的硬盘结构。

第 22 章

Initialization

第 23 章

使用 Linux

Linux 在运行的过程中会有很多的程序常驻在内存中，而且 Linux 使用非同步的硬盘/内存数据传输的模式，因此硬盘使用效率比较高，同时 Linux 提供的是多用户多任务的操作环境，由此对于 Linux 系统来说，不正常开关机有可能造成硬盘数据的损坏。

在 Linux 使用过程中，正确的开关机是很重要的，不正常的关机可能会导致整个系统的分区错乱并造成数据的损坏，这也是为什么通常 Linux 主机都会附加一个不断电系统的原因。

事实上，GRUB 的功能很多，其中就包含可以在系统发生错误的时候以额外的参数来强制开机并进行系统的修复等的功能。

此外，也可以以另一个开机管理程序 LILO 来设定 MBR 的开机菜单。不过在默认的情况下，Linux 并不会主动的安装 LILO。

一般来说，在操作 Linux 系统时，除非必要，否则不要使用 root 的权限，这是因为管理员（root）的权限太大了，所以建立一个一般身份用户来操作才是好习惯。举例来说，一般身份用户的帐号用来操作 Linux ，而当主机需要额外的 root 权限时，才使用身份切换命令来切换身份成为 root 来管理和维护。

> Tips：
> 为了让 X Window 的显示效果更好，很多团体开始开发桌面应用的环境，KDE/GNOME 就是其中的代表，这些团体的目标就是开发出类似 Windows 桌面的一整套可以工作的桌面环境，KDE 是构建在 X Window 上面的，可以进行视窗的定位、放大、缩小，同时还提供很多的桌面应用软件，详情可以参考http://www.kde.org/，GNOME 则是另外一个计划。

另外，Linux 是多用户多任务的操作系统，那么每个用户自然应该都会有自己的“工作目录”，这个目录是用户可以完全控制的，所以就称为“用户个人目录”，一般来说，主文件夹都在/home 下。

资源管理器在 GNOME 中其实称为“鹦鹉螺（Nautilus）”，而在 KDE 中则称为“征服家（Konqueror）”。在使用资源管理器查看文件时，文件名开头为小数点“.”的是隐藏文件。

一般来说，用户是可以手动来直接修改 X Window 的设定，不过修改完成之后 X Window 并不会立即载入，必须要重新启动 X 才行。特别注意，不是重新开机，而是重新启动 X。

重新启动 X 最简单的方法就是在 X 的画面中直接按下 [Alt]+ [Ctrl] + [Backspace]，这样就可以直接重新启动 X，也就可以直接读入设定。

另外，如果 X Window 因为不明原因导致有点问题时也可以利用这个方法来重新启动 X。

23.1 X Window 与命令行模式的切换

通常也称命令行模式为终端界面，terminal 或 console。Linux 默认的情况下会提供 6 个 terminal 来让用户登录，切换的方式为使用 [Ctrl] + [Alt] + [F1] [F6] 的组合键，同时系统为了判断，会将 [F1] [F6] 定义为 tty1 tty6 的操作界面环境，也就是说当按下 [crtl] + [Alt] + [F1] 这三个组合键时，就会进入到 tty1 的 terminal 界面中。同样的 [F2] 就是 tty2 ，如果要回到刚刚的 X Window，按下 [Ctrl] + [Alt] + [F7] 就可以。

Tips：

某些 Linux distribution 会使用到 F8 这个终端界面做为桌面终端，所以这部份还不是很统一，无论如何，尝试按按 F7 或者 F8 就可以知道。

- Ctrl + Alt + F1～F6：命令行界面登录 tty1～ tty6 终端；
- Ctrl + Alt + F7 ：图形界面桌面。

也就是说，如果是以命令行界面登录，那么可以有 tty1～ tty6 这 6 个命令行界面的终端可以使用，但是图形界面则没有任何东西。如果以图形界面登录，就可以使用图形界面跟命令行界面，而如果是以命令行界面启动 Linux，tty7 默认是没有东西的，那就可以直接通过命令：

```
[root@linux ~]# startx
```

“理论上”可以启动图形界面。当然前提是“已安装 KDE/GNOME 等桌面系统，X Window 已经设定好，并且 X Server 能够顺利启动”，另外启动 X Window 所必需的服务，例如字体服务器（X Font Server，XFS）必须要先启动。

在 Linux 开机之后，可以进入 X Window 或者是纯命令行界面，那么这两种环境是否可以更改呢？这就涉及到所谓的“Run Level”。

Linux 默认提供了 7 个 Run Level 给用户使用，如果需要 Linux 下次以纯文本环境（run level 3）启动，可以将默认启动的 X Window（run level 5）改为不启动（run level 3），这只要修改/etc/inittab 这个文件的内容就可以。

23.2 以命令行模式登录 Linux

如果使用命令行界面（其实是 run level 3）启动 Linux 主机，那么默认就是登录到 tty1 这个环境中。

```
linux login: root
Password:
[root@linux ~]# _
```

root 就是“系统管理员”，也就是“超级用户，Super User”。在 Linux 主机内，root 帐号代表的是“无穷的权力”，任何事都可以进行，因此使用这个帐号要小心。

注意，在输入密码的时候显示器上面不会显示任何字符，正确登录之后最左边的 root 显示的是“目前用户的帐号”，而之后接的 linux 则是“主机名称”，最右边的 ～ 则指的是“目前所在的目录”，那么那个 # 则是用户常常讲的“提示字符”。

Tips：

～ 符号代表的是“用户的家目录”，它是个“变量”，举例来说，root 的家目录在/root，～ 就代表/root。

Linux 中默认 root 的提示字符为 # ，而一般身份用户的提示字符为 $，登录成功后显示的内容部分其实是来自于/etc/issue 这个文件。

在一般的 Linux 使用情况中，为了“系统与网络安全”的考虑，通常用户都希望不要以 root 身份来登录主机，这是因为系统管理员帐号 root 具有无穷大的权力，例如 root 可以删除任何一个文件或目录，因此一个称职的网络/系统管理人员通常都会具有两个帐号，平时以一般帐号来使用 Linux 主机的资源，有需要用到系统功能修改时才会转换身份成为 root。

退出 Linux 系统可以直接这样做：

```
[root@linux ~]# exit
```

注意，注销、退出系统和结束当前会话并不是关机，只是让当前账号离开系统而已。基本上 Linux 本身已经有相当多的工作在进行，而登录也仅是其中的一个“工作”而已，所以当某一个用户离开时，那么该工作就停止了，但此时 Linux 其他的工作是还是进行的。

23.3 以命令行模式执行命令

其实用户都是通过“程序”来跟系统通信的，比如窗口或命令行都是一组或一个程序在负责用户所想要完成的命令。“命令行模式”就是指在登录 Linux 的时候得到的一个 Shell，Shell 提供用户一些工具，可以通过 Shell 来改变 kernel 的行为。其实整个命令下达的方式很简单，只要记得几个重要的概念就可以了。举例来说，可以这样下达命令：

```
[root@linux ~]# command [-options] parameter1 parameter2 ...
                  命令      选项       参数 (1)    参数 (2)
```

说明：

0、一行命令中第一个输入的绝对是“命令(command)”或“可执行文件”。

1、command 为命令的名称，例如变换路径的命令为 cd 等；

2、中刮号 [] 并不存在于实际的命令中，而加入参数设定时，通常为 -号，例如 -h；

有时候完整参数名称会输入 --符号，例如 --help；

3、parameter1 parameter2... 为依附在 option 后面的参数，或者是 command 的参数；

4、command, -options, parameter1.. 中间以空格来区分，不论空几格 Shell 都视为一个空格；

5、按下 [Enter] 按键后，该命令就立即执行。[Enter] 按键为 <CR> 字符（Command Run），代表著一行命令的开始启动。

6、命令太长的时候，可以使用 \ 符号来跳出 [Enter] 符号，使命令连续到下一行。注意 \ 后要立即接特殊字符才能转义。

其他：

a、Linux 是区分大小写的，比如 cd 与 CD 在 Linux 命令中意义并不同。

注意到上面的说明当中，“第一个被输入的数据绝对是命令或者是可执行的文件”。

例：以 ls 这个“命令”列出“/root”这个目录下的“所有隐藏档与相关的文件属性”，文件的属性的 option 为 -al ，所以有：

```
[root@linux ~]# ls -al /root
[root@linux ~]# ls -al /root
```

上面这两个命令的下达方式是一模一样的执行结果。

Linux 是区分大小写的，在下达命令的时候千万要注意到命令是大写还是小写。

```
[root@linux ~]# date
[root@linux ~]# Date
[root@linux ~]# DATE
```

Linux 是支持多国语系的，若可能的话，显示器的信息是会以该支持语系来输出的，但是终端界面在默认的情况下无法支持以中文编码输出数据，此时需要将支持语系改为英文才能够显示出正确的信息。可以这样做：

注意一下，上面每一行命令都是用等号“=”连接并且等号两边没有空格，是连续输入的。这样一来就能够在“本次的登录”查看英文信息，但如果退出 Linux 后，刚刚下达的命令就没有用了。

23.4 基础命令的操作

23.4.1 显示日期的命令：date

如果想要在命令行界面显示目前的时间，可以直接在命令行模式输入 date：

```
[root@linux ~]# date
```

另外，date 还有其他相关功能，例如：

```
[root@linux ~]# date +%Y/%m/%d
[root@linux ~]# date +%H:%M
```

+%Y/%m/%d 就是 date 的一些参数，这同时也说明在某些特殊情况下，参数前面也会带有正号“+”等。

23.4.2 显示日历的命令：cal

如果想要列出目前这个月份的月历，可以直接使用 cal 命令：

```
[root@linux ~]# cal
```

基本上，cal(calendar)命令可以做的事情还有很多，可以显示整年的月历情况：

```
[root@linux ~]# cal 2012
```

也就是说，基本上 cal 的语法为：

```
[root@linux ~]# cal [[month] year]
```

所以如果想要知道 2012 年 7 月的月历，可以直接使用如下命令：

```
[root@linux ~]# cal 7 2005
```

```
[root@linux ~]# cal 13 2012
```

某些命令有特殊的参数存在，若输入错误的参数则该命令会有错误信息的提示，通过这个提示可以知道命令执行错误之处。

23.4.3 计算器命令：bc

如果想要使用计算器，可以使用 bc 命令。输入 bc 之后，显示出版本信息之后就进入到等待命令的阶段，如下所示：

```
[root@linux ~]# bc
```

事实上，用户是“进入到 bc 这个命令的工作环境”中了，下面用户输入的数据都是在 bc 程序当中在进行运算的动作，所以输入的数据当然就得要符合 bc 的要求。在基本的 bc 计算操作之前，先要理解 bc 可以执行哪些运算：

- + 加法
- - 减法
- * 乘法
- / 除法
- ^ 指数
- % 余数

这里怎么 10/100 会变成 0 呢？原来是 bc 默认仅输出整数，如果要输出小数点下位数，那么就必须要执行 scale=number，那个 number 就是小数点位数，例如：

```
[root@linux ~]# bc
bc 1.06
Copyright 1991-1994, 1997, 1998, 2000 Free Software Foundation, Inc.
This is free software with ABSOLUTELY NO WARRANTY.
For details type `warranty'.
scale=3
1/3
.333
340/2349
.144
quit
```

Tips:

就像执行 bc 会进入 bc 的软件功能一样,那怎么知道目前等待输入的地方是某个软件的功能还是 Shell 的可输入命令的环境下呢?

其实,在进入 Linux 的时候就会出现提示字符了,如果发现在等待输入的地方并非提示字符,那通常就是已经进入到某个软件的功能当中了。

23.5 重要的几个热键 [tab], [ctrl]-c, [ctrl]-d

命令行模式里有很多的功能键,这些按键可以辅助用户进行命令的编写与程序的中断。

23.5.1 [tab] 按键

在各种 UNIX-like 的 Shell 当中,[tab] 按键算是 Linux 的 Bash Shell 最有用的功能之一。

[tab] 按键具有“命令补全”与“文件补齐”的功能,可以避免用户打错命令或文件名称,但是 [tab] 按键在不同的地方输入会有不一样的结果。

举下面的例子来说明,上一小节不是提到 cal 这个命令吗?如果在命令行输入 ca 再按两次 [tab] 按键,看看会出现什么信息?

```
[root@linux ~]# ca[tab][tab] <==[tab] 按键是紧接在 a 字母后面
```

上面的 [tab] 指的是“按下那个 tab 键”,不是要输入 ca[... 的意思。

```
cadaver            callgrind_control capiinit  case
cal                cancel            capinfos  cat
calibrate_ppa      cancel.cups       captoinfo catchsegv
caller             capifax           card
callgrind          capifaxrcvd       cardctl
callgrind_annotate capiinfo          cardmgr
```

所有以 ca 为开头的命令都被显示出来了,那如你输入 ls -al /.bash,然后两个 [tab] 会出现什么呢?

```
[root@linux ~]# ls -al ~/.bash[tab][tab]
.bash_history  .bash_logout   .bash_profile  .bashrc
```

在该目录下所有以.bash 开头的文件名称都会被显示出来了。

用户按 [tab] 按键的地方如果是在 command(第一个输入的数据)后面时,它就代表“命令补全”,如果是接在第二个字以后就会变成“文件补齐”的功能。

- [*tab*] 接在一串命令的第一个字的后面,则为“命令补全”;
- [*tab*] 接在一串命令的第二个字以后时,则为“文件补齐”。

善用 [tab] 按键可以避免很多输入错误。

23.5.2 [ctrl]-c 按键

在 Linux 下如果输入了错误的命令或参数,导致程序无法停止或循环执行,可以使用 [ctrl] 与 c 组合键,可以中断当前程序。

不过应该要注意的是,这个组合键是可以将正在运行中的命令中断,如果正在运行比较重要的命令,就不能使用这个组合按键了。

23.5.3 [ctrl]-d 按键

[ctrl] 与 d 的组合键通常代表“键盘输入结束(End Of File,EOF 或 End Of Input)的意思。另外,也可以用来取代 exit 的输入,例如想要直接离开命令行界面,可以直接按下 [ctrl]-d 就能够直接离开了(相当于输入了 exit)。

23.6 查看错误信息

当输入了错误的命令或得到了错误的结果时，可以通过显示器上显示的错误信息来了解问题出在哪里，那就很容易知道如何处理这个错误信息。举个例子来说，假如想执行 date 却打错成为 DATE 时，这个错误的信息是这样显示的：

```
[root@linux ~]# DATE
-bash: DATE: command not found
```

上面那个 bash: 表示的是用户的 Shell 名称，在构成计算机的“用户、用户界面、核心、硬件”的架构中，Shell 就是用户界面，在 Linux 中默认的用户界面是 bash Shell。

那么上面的例子就说明了，bash 有错误，具体什么错误呢？在这里 bash 告诉用户：

```
DATE: command not found
```

字面上的意思是说“找不到命令”，哪个命令呢？就是 DATE 这个命令，这里的错误信息意思是说系统上面可能没有 DATE 这个命令。

```
[root@linux ~]# cal 13 2012
cal: illegal month value: use 1-12
```

这时 cal 警告用户，illegal month value: use 1-12，意思是说“不合法的月份值，应该使用 1-12 之间的数字”。

23.7 Linux 系统帮助:man page/info page

在命令行模式下,可以直接按下两个 [tab] 按键,看看 Linux 总共有多少命令,结果是至少也有 2000 多个以上的命令。对于这么多的 Linux 命令,用户应该主要还是以理解“在什么情况下,应该要使用哪方面的命令”来使用这些命令。

Linux 开发的软件大多数都是自由软件,这些软件的开发者为了让大家能够了解命令的用法,都会自行制作很多的文件,这些文件也可以直接在线上就能够轻易的被用户查询到,这基本上就是“在线帮助文件”。

23.7.1 man page

在 Linux 中,对于用户不熟悉的命令,只要使用简单的方法去寻找一下说明的内容,马上就清清楚楚的知道该命令的用法了,这就是 man 命令的用途。

man 是 manual(操作说明)的简写,比如查询 date 命令的使用方法,可以使用 man date,于是就会有清楚的说明。

```
[root@linux ~]# LANG="en"
[root@linux ~]# man date
DATE(1)             User Commands             DATE(1)
NAME
       date - print or set the system date and time
SYNOPSIS
       date [OPTION]... [+FORMAT]
       date [-u|--utc|--universal] [MMDDhhmm[[CC]YY][.ss]]
DESCRIPTION
       Display the current time in the given FORMAT,
       or set the system date.
       -d, --date=STRING
              display time described by STRING, not 'now'
       -f, --file=DATEFILE
              like --date once for each line of DATEFILE
       -ITIMESPEC, --iso-8601[=TIMESPEC]
              output  date/time  in ISO 8601 format.
              TIMESPEC='date' for date only,'hours','minutes',
              'or' seconds' for date and time to the indicated
              precision. --iso-8601  without TIMESPEC defaults
              to 'date'.
.................(略)....
AUTHOR
       Written by David MacKenzie.
REPORTING BUGS
       Report bugs to .
COPYRIGHT
       Copyright (c)2004 Free Software Foundation, Inc.
       This is free software; see the source for copying conditions.
       There is NO warranty; not even for MERCHANtabILITY or FITNESS
       FOR A PARTICULAR PURPOSE.
SEE ALSO
       The full documentation for date is maintained as a  Texinfo
       manual.If the info and date programs are properly installed
       at your site, the command
              info coreutils date
       should give you access to the complete manual.
date (coreutils) 5.2.1     May 2005         DATE(1)
```

上述的页称为 man page,可以在里面查询命令的用法与相关的参数说明。

如果仔细一点来看这个 man page 的话，应该会发现几个有趣的东西。

首先，在上个表格的第一行，可以看到的是：“DATE(1)”，其中 (1) 代表的是“一般用户可使用的命令”，这可以帮助用户了解或者是直接查询相关的数据。

常见的几个数字的意义是这样的：

代号	代表内容
1	用户在 Shell 环境中可以执行的命令或可执行文件
2	系统核心可调用的函数与工具等
3	一些常用的函数(function)与函数库(library)，大部分为 C 的函数库(libc)
4	设备文件的说明，通常是在/dev 下的文件
5	配置文件或者是某些文件的格式
6	游戏(games)
7	惯例与协议等，例如 Linux 标准文件系统、网络协议、ASCII 码等的说明内容
8	系统管理员可用的管理命令
9	跟 kernel 有关的文件

使用 man page 在查看某些数据时，就会知道该命令/文件所代表的基本意义是什么了。举例来说，如果下达了 man null 时，会出现的第一行是：“NULL(4)”，对照一下上面的数字意义，原来 null 是一个“配置文件”。

而且 man page 的内容也分成好几个部分来说明，也就是上面 man date 那个表格内以 NAME 作为开始介绍，最后还有 SEE ALSO 来作为结束。

基本上 man page 大致分成下面这几个部分：

代号	内容说明
NAME	简短的命令、数据名称说明
SYNOPSIS	简短的命令执行语法 (syntax) 简介
DESCRIPTION	较为完整的说明
OPTIONS	针对 SYNOPSIS 部分列举的所有可用的参数说明
COMMANDS	当这个程序(软件)在执行时，可以在此程序(软件)中执行的命令
FILES	这个程序或数据所使用或参考或连接到的某些文件
SEE ALSO	可以参考的，跟这个命令或数据有相关的其他说明
EXAMPLE	一些可以参考的范例
BUGS	是否有相关的 bug

有时候除了这些外，还可能会看到 AUTHORS 与 COPYRIGHT 等，不过也有很多时候仅有 NAME 与 DESCRIPTION 等部分。通常在查询某个数据时，一定要查看 NAME，简单看一下这个数据的意思，再详看一下 DESCRIPTION，这个 DESCRIPTION 会提到很多相关的数据与使用时机，从这个地方可以学到很多小细节。接下来主要就是查询关于 OPTIONS 的部分，从而可以知道每个参数的意思，这样就可以执行比较细部的命令内容。最后会再看一下跟这个数据有关的还有哪些信息可以使用的。举例来说，上面的 SEE ALSO 就告知用户还可以利用“info coreutils date”来进一步查阅信息，某些说明内容还会列举有关的文件(FILES 部分)来供用户参考，这些都是很有帮助的。

另外，在 man page 中还可以利用哪些按键来帮忙查阅呢？首先，如果要向下翻页的话，可以按下键盘的空格键，也可以使用 [Page Up] 与 [Page Down] 来翻页。同时，如果知道某些关键字，那么可以在任何时候输入“/word”来主动搜寻关键字，例如在上面的搜寻当中，输入/date 会变成如下这样：

```
DATE(1)          User Commands                DATE(1)
NAME
       date – print or set the system date and time
```

```
SYNOPSIS
       date [OPTION]... [+FORMAT]
       date [-u|--utc|--universal] [MMDDhhmm[[CC]YY][.ss]]
DESCRIPTION
       Display the current time in the given FORMAT,
       or set the system date.
............(中间省略).........
/date
```

按下“/”之后,光标应该就会移动到显示器的最下面一行,并等待输入搜寻的字符串。此时输入 date 后,man page 就会开始搜寻跟 date 有关的字符串,并且移动到该区域。最后如果要离开 man page,直接按下“q”就可以离开了,一些会在 man page 常用的按键整理如下:

按键	进行工作
空格键	向下翻一页
[Page Down]	向下翻一页
[Page Up]	向上翻一页
[Home]	去到第一页
[End]	去到最后一页
/string	向“下”搜寻 string 这个字串
?string	向“上”搜寻 string 这个字串
n, N	利用/或? 来查询字串时,可以用 n 来继续下一个查询(不论是/还是?),可以利用 N 来进行“反向”查询,反过来 N 和 n 可以互换。
q	结束这次的 man page

注意上面的按键是在 man page 的画面当中才能使用的,有趣的是搜索功能,用户可以往下或者是往上搜寻某个字串,例如要在 man page 内搜寻 vbird 这个字串,可以输入/vbird 或者是?vbird,只不过一个是往下而一个是往上来搜寻的。而要“重复搜索”某个字符串时,可以使用 n 或者是 N 即可。

至于这些 man page 的数据的存放位置,不同的 distribution 通常可能有点差异,不过通常是放在/usr/share/man 里,然而用户可以通过修改 man page 搜索路径,可以通过修改/etc/man.config (或 man.conf、manpath.conf)来完成,更多的关于 man 的信息可以使用“man man”来查询。

man 还可以查询特定命令/文件的 man page 说明文件,具体来说,当要使用某些特定的命令或者是修改某些特定的配置文件时,可以使用 man 命令在找到所需要的 man page。下面的例子可以用来找到系统中与“man”有关的说明文件或者更多跟 man 较相关的信息,此时可以执行:

```
[root@linux ~]# man -f man
man             (1)  - format and display the on-line manual pages
man             (7)  - macros to format man pages
man.conf [man]  (5)  - configuration data for man
```

使用 -f 参数可以取得更多的 man 的相关信息,而上面这个表格中也有提示了(数字)的内容,举例来说,第二行的“man (7)”表示有个 man (7) 的说明文件存在,也有个 man (1) 存在。对于上表当中的两个 man,可以使用这样的命令将对应的文件找出来:

```
[root@linux ~]# man 1 man  <== 这里是用 man(1) 的文件数据
[root@linux ~]# man 7 man  <== 这里是用 man(7) 的文件数据
```

将上面两个命令执行之后就可以看到,两个命令输出的结果是不同的,这里 1 和 7 就是分别取出在 man page 里面关于 1 与 7 相关数据的文件。而至于搜索的文件类型是 1 还是 7,就和查询的顺序有关。

查询的顺序是记录在/etc/man.conf 这个配置文件中的,先查询到的说明文件会先被显示出来。一般来说,因为排序的关系通常会先找到数字较小的那个文件,所以 man man 会跟 man 1 man 结果相同。

除此之外,用户还可以利用“关键字”找到更多的说明文件,当我们使用“man -f 命令”时,man 只会找数据中的命令(或文件)的完整名称,有一点不同都不行,而使用“man -f 关键字”命令,则只要含有关键字就会列出来,例如:

```
[root@linux ~]# man -k man
. [builtins]         (1)  - bash built-in commands, see bash(1)
alias [builtins]     (1)  - bash built-in commands, see bash(1)
.........(中间省略)....
xsm                  (1x) - X Session Manager
zshall               (1)  - the Z Shell meta-man page
zshbuiltins          (1)  - zsh built-in commands
zshzle               (1)  - zsh command line editor
```

在系统的说明文件中,只要有 man 这个关键字就会将该 manual 列出来,这里就是利用关键字将说明文件里面只要含有 man 那个字符串(不见得是完整字串)的文件查找出来。

事实上,还有两个命令与 man page 有关,而这两个命令是 man 的简略写法,分别是:

```
[root@linux ~]# whatis  [命令或者是数据]   <== 相当于 man -f [命令或者是数据]
[root@linux ~]# apropos [命令或者是数据]   <== 相当于 man -k [命令或者是数据]
```

而要注意的是,这两个特殊命令要能使用,必须要创建 whatis 数据库才行,这个数据库的创建需要以 root 身份执行如下命令:

```
[root@linux ~]# makewhatis
```

23.7.2 info page

在所有的 UNIX-like 系统中都可以利用 man 来查询命令或者是相关文件的用法。但是在 Linux 里面则又额外提供了一种线上求助的方法,那就是利用 info。

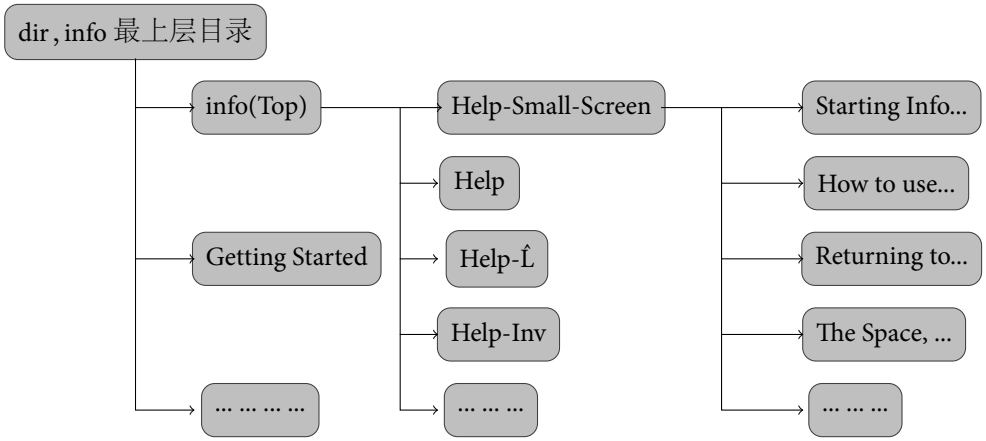
基本上,info 与 man 其实差不多,都是用来查询命令的用法或者是文件的格式,但是与 man 一次就输出所有信息不同,info page 则是将文件数据拆成一个一个的段落,每个段落用自己的页面来撰写,并且在各个页面中还有类似网页的“超链接”导航来跳转到各个不同的页面中,因此 info 文件数据必须要以 info 写成的才会比较完整,每个独立的页面也被称为一个节点(node),所以可以将 info page 看作是命令行模式的网页显示数据。

支持 info 命令的文件是放置在/usr/share/info/这个目录中的。举例来说,info 的说明文件是 info 格式,所以使用 info info 可以得到:

```
[root@linux ~]# info info
File:info.info, Node:Top, Next:Getting Started,Up:(dir)
Info:An Introduction
**********************
The GNU Project distributes most of its on-line manuals
in the "Info format",which you read using an "Info reader".
You are probably using an Info reader to read this now.
   There are two primary Info readers:`info', a stand-
alone program designed just to read Info files,and the
`info' package in GNU Emacs,a general-purpose editor.
At present,only the Emacs reader supports using a mouse.
   ---------省略---------------
   To read about expert-level Info commands, type `n'
twice. This brings you to `Info for Experts',skipping
over the `Getting Started' chapter.
* Menu:
* Getting Started::        Getting started using an Info reader.
* Expert Info::            Info commands for experts.
* Creating an Info File::  How to make your own Info file.
```

```
* Index::                      An index of topics,commands,and variables.
--zz-Info:(info.info.gz)Top,29 lines --All--------
Welcome to Info version 4.8.
Type ? for help,m for menu item.
```

可以发现最后一行显示出目前的 info 这个程序的版本信息,按下 m 按键就可以有更多的命令说明。而第一行则显示目前这个 info page 的文件名以显示数据来源,第一行的 Node 显示当前这个画面是“在第几层”(所属节点位置),因为 info page 将所有有关的数据都进行了连接,因此它可以利用分层的架构来说明每个文件数据,而且还有下一层数据,因此会看到第一行还有 Next 字符串,这表示只要输入“n”(Next)键后就可以跑到下一层,也就是 Getting Started 那个章节,输入“u”(Up)回到上一层,输入“p”(Pre)回到上一个节点。



用户可以将游标移动到该命令行或者 * 上面,按下 Enter 就可以前往该小节,利用 [tab] 键就可以快速的将游标在上表的画面中的 node 间移动,这里 node 就是各个入口点。

举例来说,上个表格当中,按下 n 或者是将游标游动到 Next 这个字上并按下 Enter 就可以前往下个说明了。

info 的说明文件将内容分成多个节点,并且每个节点都有定位与链接。在各链接之间还可以具有类似“超链接”的快速按钮,可以通过 [tab] 键在各个超链接之间移动,也可以使用 u、p、n 字母键在各个分层与相关链接中显示。

不过,info 需要系统支持才可以,如果用户以没有提供支持的 man 命令来查看,info man 的结果与 man man 的结果就一样了,至于 info page 当中可以使用的按键,可以整理成如下这样:

按键	进行工作
空白键	向下翻一页
[Page Down]	向下翻一页
[Page Up]	向上翻一页
[tab]	在 node 之间移动,有 node 的地方,通常会以 * 显示。
[Enter]	当游标在 node 上面时,按下 Enter 可以进入该 node。
b	移动游标到该 info 画面当中的第一个 node 处
e	移动游标到该 info 画面当中的最后一个 node 处
n	前往下一个 info page 处
p	前往上一个 info page 处
u	向上移动一层
s(/)	在 info page 当中进行搜寻
h	显示求助菜单
?	命令一览表
q	结束这次的 info page

23.7.3 其他有用的文件

一般而言，命令或者软件开发者都会将自己的命令或者是软件的说明制作成“在线帮助文件”，但是毕竟不是都需要做成在线帮助文件的，还有相当多的说明需要额外的文件，此时这个所谓的 How-To Documents 就很重要了。

另外，某些软件不只告诉用户“如何做”，还会有一些相关的原理会说明，这些说明文件一般是存放在/usr/share/doc 中的。举例来说，如果想要知道与当前版本的 Fedora 相关的各项信息，可以直接到/usr/share/doc/fedora-release-x 目录查阅。如果想要知道 bash 是什么，则可以到/usr/share/doc/bash-x 目录中查阅。

注意，/usr/share/doc 目录下的数据主要是以软件包（packages）为主的，例如 GCC 的相关信息在/usr/share/doc/gcc-xxx（xxx 表示版本的意思）。

另外，在出现任何问题的时候，除了自己检查之外，可以到搜索引擎或各大对应的与 Linux 相关的网站中查找相应的解决方案之外，下面列出了一些有用的 FAQ 和 How-To 网站供参考。

```
CLDP 中文文件计划:http://www.linux.org.cn/CLDP/
The Linux Documentation Project:http://www.tldp.org/
```

23.8 正确的关机方法（shutdown, reboot, init, halt）

在 Windows（非 NT 主机系统）系统中，由于是单用户假多任务的情况，所以即使关机对于别人应该不会有影响。不过在 Linux 下，由于每个程序（或者说是服务）都是在后台执行的，因此在用户看不到的显示器背后其实可能有相当多人同时在主机上面工作，如果直接按下电源开关来关机，其他人的数据可能就此中断。此外最大的问题是，若不正常关机则可能造成文件系统的损坏（因为来不及将数据回写到文件中，所以有些服务的文件会有问题）。正常情况下要关机时需要注意下面几件事：

1、观察系统的使用状态：如果要看目前有谁在线上，可以执行 `who` 命令，而如果要看网络的连接状态，可以执行 `netstat -a` 命令，而要看后台执行的程序可以执行 `ps -aux` 命令。使用这些命令可以让用户稍微了解主机目前的使用状态，当然也就可以判断是否可以关机了。

2、通知在线用户关机的时间：要关机前总得给在线的用户一些时间来结束他们的工作，所以这个时候可以使用 `shutdown` 的命令来完成这个功能。

3、正确的关机命令使用 `shutdown` 与 `reboot` 命令。

由于 Linux 系统的关机/重启是重大的系统操作，因此只有 root 账号才能够执行 shutdown/reboot 等命令，不过某些 distribution，例如 CentOS 允许在主机的 tty7 使用图形界面登录时使用一般账号来关机或重启，但其他一些 distribution 则在要关机时要求输入 root 密码。

23.8.1 数据同步写入硬盘：sync

在 Linux 系统中，为了加快数据的读取速度，在默认的情况下某些数据将不会直接被写入硬盘而是先暂存在内存当中，这样如果一个数据被用户重复的读写，那么由于它尚未被写入硬盘中，因此可以直接由内存当中读取出来，在速度上要快一些。

不过，这也造成一些困扰，那就是万一当用户重新开机或者是关机亦或者是不正常的断电的情况下，由于数据尚未被写入硬盘当中，所以就会造成数据的更新不正常。

这个时候就需要 sync 命令来进行数据的写入动作，在命令行界面下输入 sync，在内存中尚未被更新的数据就会被写入硬盘中，这个命令在系统关机或重新开机之前是很重要的，最好多执行几次。

sync 命令也只有 root 才可以执行，虽然目前的 shutdown/reboot/halt 等命令均已经在关机前进行了 sync 这个工具的调用，不过仍然可以通过手动执行该命令来保护数据。

> 事实上，sync 也可以被一般账号使用，只不过一般账号用户所更新的硬盘数据仅是其自己的数据，而 root 账号使用 sync 可以更新整个系统中的数据。

```
[root@linux ~]# sync
```

23.8.2 关机命令:shutdown

注意只有 root 才有权力关机,较常使用的关机命令是 shutdown,该命令会通知系统内的各个进程(processes),并且将通知系统中的关闭 run level 内的一些服务。shutdown 可以完成如下功能:

1、可以自由选择关机模式:是要关机、重新开机或进入单人操作模式均可;

2、可以设定关机时间:可以设定成现在立刻关机,也可以设定某一个特定的时间才关机;

3、可以自定义关机信息:在关机之前,可以将自己设定的信息传送给在线用户;

4、可以仅发出警告信息:有时有可能要进行一些测试而不想让其他的用户干扰,或者是明白的告诉用户某段时间要注意一下,这个时候可以使用 shutdown 来通知用户但却不是真的要关机;

4、可以选择是否要使用 fsck 命令检查文件系统。

简单的 shutdown 语法规则为:

```
[root@linux ~]# /sbin/shutdown [-t 秒][-arkhncfF][时间][警告信息]
实例:
[root@linux ~]# /sbin/shutdown -h 10 'I will shutdown after 10 mins'
告诉大家,这部机器会在十分钟后关机,并且会显示在目前登录用户的显示器上。

-t sec  □ -t 后面加秒数,也就是"过几秒后关机"的意思
-k      □ 不要真的关机,只是发送警告信息出去。
-r      □ 在将系统的服务停掉之后就重新开机
-h      □ 将系统的服务停掉后,立即关机。
-n      □ 不经过 init 程序,直接以 shutdown 的功能来关机
-f      □ 关机并开机之后,强制略过 fsck 的硬盘检查
-F      □ 系统重新开机之后,强制进行 fsck 的硬盘检查
-c      □ 取消已经在进行的 shutdown 命令内容。
时间    : 这是一定要加入的参数,指定系统关机的时间。
```

此外需要注意的是,时间参数请务必加入,否则会自动跳到 run-level 1(就是单人维护的登录情况)。

```
[root@linux ~]# shutdown -h now
立即关机,其中 now 相当于时间为 0 的状态
[root@linux ~]# shutdown -h 20:25
系统在今天的 20:25 分会关机。若在 21:25 才执行此命令,则隔天才关机。
[root@linux ~]# shutdown -h +10
系统再过十分钟后自动关机
[root@linux ~]# shutdown -r now
系统立刻重启
[root@linux ~]# shutdown -r +30 'The system will reboot'
再过三十分钟系统会重新开机,并显示后面的信息给所有在线的用户。
[root@linux ~]# shutdown -k now 'This system will reboot'
仅发出警告信件的参数,系统并不会关机。
```

23.8.3 reboot, halt, poweroff

这三个命令差不多,它们调用的函数库都差不多,仅仅是用途上有些不同而已,reboot 其实与 shutdown -r now 几乎相同。不过,建议在关机之前还是将数据同步的命令执行一次再执行关机命令。

```
[root@linux ~]# sync; sync; sync; reboot
```

此外,halt 与 poweroff 也具有相同的功能。

通常在重启的时候,都会执行如下的命令:

```
[root@linux ~]# sync; sync; sync; reboot
```

基本上,在默认的情况下,reboot、helt、poweroff、shutdown 这几个命令都可以完成一样的工作(因为 halt 会先调用 shutdown,而 shutdown 最后会调用 halt),不过 shutdown 可以依据目前已启动的服务来逐次关闭各服务后才关机,halt 却能够在不理会目前系统状况下进行硬件关机的特殊功能。

```
[root@linux ~]# shutdown –h now
[root@linux ~]# poweroff –f
```

23.8.4 切换执行等级:init

系统运作的模式分为命令行界面(run level 3)和图形模式界面(run level 5)。Linux 共有 7 种执行等级,其中:

- run level 0:关机;
- run level 3:纯命令行界面;
- run level 5:含有图形界面模式;
- run level 6:重启。

使用 init 命令切换执行等级,也就是说,如果要关机,除了执行上述的 shutdown -h now 以及 poweroff 之外,也可以使用如下的命令来关机:

```
[root@linux ~]# init 0
```

23.9 开机过程的问题排解

23.9.1 文件系统错误的问题

在开机的过程中最容易遇到的问题就是硬盘可能有坏道或分区错乱(数据损坏)的情况,这种情况虽然不容易发生在稳定的 Linux 系统下,不过由于不当的开关机还是可能会发生的,原因可能有:

最可能发生的原因是因为断电或不正常关机所导致的硬盘扇区错乱,硬盘使用率过高或主机所在环境不良也是一个可能的原因,例如用户开放了 FTP 服务,但是使用的又不是稳定的 SCSI 接口硬盘,仅使用 IDE/SATA 等接口的硬盘,虽然机率真的不高,但还是有可能造成扇区错乱的。另外,如果主机所在环境散热不良,或者是湿度相对较高,也很容易造成硬盘损坏。

解决的方法其实很简单,也可能很困难,这要取决于出错扇区所挂载的目录位置。如果根目录“/”并没有损坏,那就很容易解决,如果根目录已经损坏了,那就比较麻烦。

1、如果根目录没有损坏:假设发生错误的硬盘区块是在/dev/hda7,那么在开机的时候,系统自检信息应该会告诉用户:press root password or ctrl+D:

这时候输入 root 的密码登录系统,进行单用户的维护工作。

输入 fsck /dev/hda7(fsck 为命令,/dev/hda7 为错误的硬盘区块),然后依据实际情况设置相应的执行参数,这时会显示开始整理硬盘的信息,如果有发现任何的错误时,会显示:clear [Y/N] 的询问信息,直接输入 Y 即可。整理完成之后,以 reboot 重新开机。

2、如果根目录损坏了:此时可以将硬盘连接接到另一台 Linux 系统的计算机上,并且不要挂载(mount)该硬盘,然后以 root 的身份执行 fsck /dev/hdb1(/dev/hdb1 指的是硬盘设备名称,要根据实际状况来设定)。

另外,也可以使用 Live CD 启动主机进入 Linux 操作系统,再加载(mount)原本的/,以 fsck /dev/hda1 来修复根目录。

3、如果硬盘整个损毁:如果硬盘整体损毁,就需要尽可能先抢救硬盘内的数据并尽快更换硬盘。如果不愿意更换硬盘,那就只能重新安装 Linux,并且在重新安装的过程中,在格式化硬盘阶段选择

“error check”,只是如此一来,format 会很慢,并且何时会再坏掉也不确定,所以最好还是更换一块新硬盘。

硬盘需要妥善保养,比如主机通电之后不要搬动,避免移动或震动硬盘,尽量降低硬盘的温度,可以加装风扇来冷却硬盘或者可以换装 SCSI/SSD 硬盘。

使用硬盘时要划分不同的分区,Linux 每个目录被读写的频率是不同的,因而 Linux 不同的安装模式就在于硬盘划分的不同,通常会建议用户划分成下列的硬盘分区:

```
/
/boot
/usr
/home
/var
```

这样划分的好处在于,如果/var 是系统默认的一些数据暂存或者是 cache 数据的存储目录,当这部分的硬盘损坏时,由于其他的地方是没问题的,从而数据得以保存,而且在解决起来也比较容易。

23.9.2 忘记 root 密码

如果设定好了 Linux 之后忘记 root 密码,此时只要以单用户维护模式登录即可更改 root 密码。不过目前的多重引导程序主要有 LILO 与 GRUB 两种,这两种模式并不相同,有必要来说明一下。

1、LILO

只要在出现 LILO 选择菜单的时候输入:boot: linux -s

注意,如果是 Red Hat 7.0 以后的版本,会出现图形界面的 LILO,这个时候要按下 [Ctrl] + x 即可进入纯命令行界面的 LILO。

进入单人单机维护模式(即为 run-level 1)后再输入 passwd 命令就可以直接更改 root 的密码了。同时,如果图形界面无法登录的时候,也可以使用该方法来进入单人单机的维护工作,然后再去修改/etc/inittab 更改一下登录的默认模式,这样就可以在下次开机的时候以命令行模式登录。同时要注意,如果在设定启动的名称的时候更改了启动的名称,就必须在 boot: 下面输入类似于下面的命令:

```
boot: Red-Hat-2.4.7linux -s
boot: Red-Hat-2.4.7linux single
```

2、GRUB

GRUB 作为多重引导程序,要进入单人维护模式就比较麻烦一些。在开机的过程当中会有读秒的时刻,此时按下任意按键就会进入选择菜单界面,然后只要选择相应的核心文件并且按下“e”就可以进入编辑界面了。此时看到的画面有点像:

```
root   (hd0,0)
kernel /boot/vmlinuz-2.4.19 ro root=LABEL=/ rhgb quiet
```

此时,需要将游标移动到 kernel 那一行,再按一次“e”进入 kernel 行的编辑界面中,然后在出现的画面当中输入 single:

```
root   (hd0,0)
kernel /boot/vmlinuz-2.4.19 ro root=LABEL=/ rhgb quiet single
```

再按下回车确定之后,按下“b”就可以开机进入单用户维护模式了,在这个模式下可以在 tty1 的地方不需要输入密码即可取得终端的控制权(而且是使用 root 的身份),这时就可以使用下面的命令修改 root 的密码。

```
[root@linux ~]# passwd
```

接下来系统会要求输入两次新的密码,然后再重启计算机。

第 24 章

Shutdown

Bibliography

[1] RAMESH NATARAJAN. 6 stages of linux boot process (startup sequence), 02 2011. URL http://www.thegeekstuff.com/2011/02/linux-boot-process/.

[2] Sheep. Linux 的启动流程留言, 08 2013. URL http://www.ruanyifeng.com/blog/user/sheep.html.

[3] Rui Silva. Disk geometry, 06 2005. URL http://www.msexchange.org/articles-tutorials/exchange-server-2003/planning-architecture/Disk-Geometry.html.

[4] Wikipedia. 全局唯一标识分区表, . URL http://zh.wikipedia.org/zh-cn/GUID%E7%A3%81%E7%A2%9F%E5%88%86%E5%89%B2%E8%A1%A8.

[5] Wikipedia. 主引导记录, . URL http://zh.wikipedia.org/wiki/%E4%B8%BB%E5%BC%95%E5%AF%BC%E8%AE%B0%E5%BD%95.

[6] Wikipedia, .

[7] 紫月冰河. Linux 查看系统和内核版本, 12 2012. URL http://yansublog.sinaapp.com/2012/12/17/linux-%E6%9F%A5%E7%9C%8B%E7%B3%BB%E7%BB%9F%E5%92%8C%E5%86%85%E6%A0%B8%E7%89%88%E6%9C%AC-2/.

[8] 阮一峰. 计算机是如何启动的？, 02 2013. URL http://www.ruanyifeng.com/blog/2013/02/booting.html.

[9] 阮一峰. Linux 的启动流程, 08 2013. URL http://www.ruanyifeng.com/blog/2013/08/linux_boot_process.html.

[10] 阿冬米博客. linux 启动过程中都发生了什么？, 07 2012. URL http://www.adonmi.com/linux/8.html.

Part III

Filesystem

第 25 章

Introduction

In computing, disk file systems are file systems which manage data on permanent storage devices. As magnetic disks are the most common of such devices, most disk file systems are designed to perform well in spite of the seek latencies inherent in such media.
在计算机科学中, 磁盘文件系统是一种在永久储存装置上管理资料的文件系统。这类装置中, 最盛行的是磁盘装置, 以硬盘为主。因此绝大多数的磁盘文件系统都是为了针对如何让磁盘系统运作最佳化而设计的。

A filesystem is the methods and data structures that an operating system uses to keep track of files on a disk or partition; that is, the way the files are organized on the disk. The word is also used to refer to a partition or disk that is used to store the files or the type of the filesystem. Thus, one might say I have two filesystems meaning one has two partitions on which one stores files, or that one is using the extended filesystem, meaning the type of the filesystem.

The difference between a disk or partition and the filesystem it contains is important. A few programs(including, reasonably enough, programs that create filesystems) operate directly on the raw sectors of a disk or partition; if there is an existing file system there it will be destroyed or seriously corrupted. Most programs operate on a filesystem, and therefore won't work on a partition that doesn't contain one (or that contains one of the wrong type).

Before a partition or disk can be used as a filesystem, it needs to be initialized, and the bookkeeping data structures need to be written to the disk. This process is called making a filesystem.

Most UNIX filesystem types have a similar general structure, although the exact details vary quite a bit. The central concepts are superblock, inode, data block, directory block, and indirection block. The superblock contains information about the filesystem as a whole, such as its size (the exact information here depends on the filesystem). An inode contains all information about a file, except its name. The name is stored in the directory, together with the number of the inode. A directory entry consists of a filename and the number of the inode which represents the file. The inode contains the numbers of several data blocks, which are used to store the data in the file. There is space only for a few data block numbers in the inode, however, and if more are needed, more space for pointers to the data blocks is allocated dynamically. These dynamically allocated blocks are indirect blocks; the name indicates that in order to find the data block, one has to find its number in the indirect block first.

Like UNIX, Linux chooses to have a single hierarchical directory structure. Everything starts from the root directory, represented by /, and then expands into sub−directories instead of having so−called 'drives'. In the Windows environment, one may put one's files almost anywhere: on C drive, D drive, E drive etc. Such a file system is called a hierarchical structure and is managed by the programs themselves (program directories), not by the operating system. On the other hand, Linux sorts directories descending from the root directory / according to their importance to the boot process.

If you're wondering why Linux uses the frontslash / instead of the backslash \as in Windows it's because it's simply following the UNIX tradition. Linux, like Unix also chooses to be case sensitive. What this means is that the case, whether in capitals or not, of the characters becomes very important. So this is not the same as THIS. This

feature accounts for a fairly large proportion of problems for new users especially during file transfer operations whether it may be via removable disk media such as floppy disk or over the wire by way of FTP.

文件系统[2]是一种用于向用户提供底层数据访问的机制。通过文件系统，可以将设备中的空间划分为特定大小的块(扇区)，一般每块 512 字节。数据存储在这些块中，大小被修正为占用整数个块。作为通用操作系统重要的组成部分，文件系统软件负责将这些块组织为文件和目录，并记录哪些块被分配给了哪个文件，以及哪些块没有被使用。

文件系统通常使用硬盘和光盘[1]这样的存储设备，并维护文件在设备中的物理位置。但是，实际上文件系统也可能仅仅是一种访问数据的界面而已，实际的数据是通过网络协议(如 NFS[2]、SMB、9P 等)提供的或者内存上，甚至可能根本没有对应的文件(如 proc 文件系统)。

传统上操作系统在内核层面上对文件系统提供支持，但通常内核态的代码难以调试，生产率较低，因此 Linux 从 2.6.14 版本开始通过 FUSE 模块支持在用户空间实现文件系统。

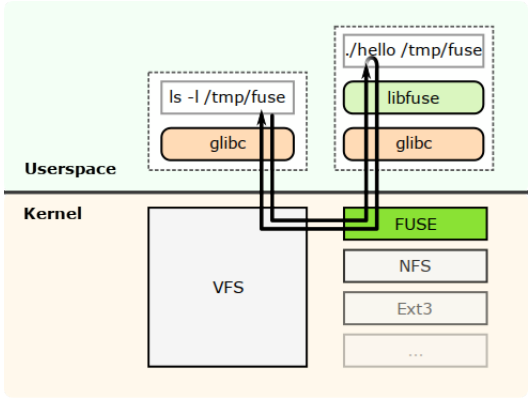


图 25.1: Linux 中 FUSE 的运行机制

FUSE (Filesystem in Userspace) 是操作系统中的概念，特指完全在用户态实现的文件系统。目前 Linux、FreeBSD、NetBSD、OpenSolaris 和 Mac OSX 支持用户空间态文件系统，其中 Linux 是通过内核模块对此进行支持，因此 FUSE 一词有时也特指 Linux 下的用户空间文件系统。另外一些文件系统诸如 ZFS[3]、GlusterFS[4]和 Lustre[5]也使用 FUSE 实现。

另外，Hadoop 提供的分布式文件系统 HDFS 可以通过一系列命令访问，并不一定经过 Linux FUSE。

在用户空间实现文件系统能够大幅提高生产率，简化了为操作系统提供新的文件系统的工作量，特别适用于各种虚拟文件系统和网络文件系统，例如上述的 ZFS 和 glusterfs 都属于网络文件系统。但是在用户态实现文件系统必然会引入额外的内核态/用户态切换带来的开销，对性能会产生一定影响。

不过，文件系统并不一定只在特定存储设备上出现。它是数据的组织者和提供者，至于它的底层，可以是磁盘，也可以是其它动态生成数据的设备(比如网络设备)，例如文件管理方面的一个新概念是一种基于数据库的文件系统的概念。不再(或者不仅仅)使用分层结构管理，文件按照它们的特征(如文件类型、专题、作者或者亚数据)进行区分，因此文件检索就可以按照 SQL 风格甚至自然语言风格进行。

计算机的文件系统是一种存储和组织计算机数据的方法，它使得对其访问和查找变得容易，文件系统使用文件和树形目录的抽象逻辑概念代替了硬盘和光盘等物理设备使用数据块的概念，用户使用文件系统来保存数据不必关心数据实际保存在硬盘(或者光盘)的地址为多少的数据块上，只需要记住这个文件的所属目录和文件名。在写入新数据之前，用户不必关心硬盘上的那个块地址没有被使用，硬盘上的存储空间管理(分配和释放)功能由文件系统自动完成，用户只需要记住数据被写入到了哪个文件中。

[1] ISO 9660 和 UDF 文件系统被用于 CD、DVD 与蓝光光盘。
[2] 网络文件系统(NFS，Network File System)是一种将远程主机上的分区(目录)经网络挂载到本地系统的一种机制。
[3] ZFS 可以认为是 Lustre 的 Linux 版本。
[4] GlusterFS 是用于集群的分布式文件系统，可以扩展到 PB 级。
[5] Sun 的 Lustre 是和 GlusterFS 类似但更早的一个集群文件系统。

严格地说，文件系统是一套实现了数据的存储、分级组织、访问和获取等操作的抽象数据类型（Abstract data type）。

- 文件名
  在文件系统中，文件名是用于定位存储位置。大多数的文件系统对文件名的长度有限制。在一些文件系统中，文件名是大小写不敏感（如“FOO”和“foo”指的是同一个文件）；在另一些文件系统中则大小写敏感。大多现今的文件系统允许文件名包含非常多的 Unicode 字符集的字符。然而在大多数文件系统的界面中，会限制某些特殊字符出现在文件名中。（文件系统可能会用这些特殊字符来表示一个设备、设备类型、目录前缀、或文件类型）然而，这些特殊的字符会允许存在于用双引号内的文件名。方便起见，一般不建议在文件名中包含特殊字符。
- 元数据
  其它文件保存信息常常伴随着文件自身保存在文件系统中。文件长度可能是分配给这个文件的区块数，也可能是这个文件实际的字节数。文件最后修改时间也许记录在文件的时间戳中。有的文件系统还保存文件的创建时间，最后访问时间及属性修改时间。（不过大多数早期的文件系统不记录文件的时间信息）其它信息还包括文件设备类型（包括区块数、字符集、套接口和子目录等），文件所有者的 ID，组 ID，还有访问权限（例如只读、可执行等）。
- 安全访问
  针对基本文件系统操作的安全访问可以通过访问控制列表或 capabilities 实现，只是访问控制列表难以保证安全，这也就是研发中的文件系统倾向于使用 capabilities 的原因。然而目前多数商业性的文件系统仍然使用访问控制列表。

磁盘文件系统是一种设计用来利用数据存储设备来保存计算机文件的文件系统，最常用的数据存储设备是磁盘驱动器，可以直接或者间接地连接到计算机上。例如 FAT、exFAT、NTFS、HFS、HFS+、ext2、ext3、ext4、ODS-5、btrfs 等。另外，有些文件系统是进程文件系统（也有译作日志文件系统）或者追踪文件系统。

闪存文件系统（Flash file system）是一种为了在闪存设备上存储数据而设计的文件系统。闪存设备跟磁盘存储设备，在硬件上有不同的特性，例如：

- 抹除区块（Erasing blocks）：闪存的区块（block）在写入之前，要先做抹除（erase）的动作。抹除区块的时间可能会很长，因此最好利用系统闲置的时间来进行抹除。
- 耗损平均技术（Wear leveling）：闪存的区块有抹写次数的限制，重复抹除、写入同一个单一区块将会造成读取速度变慢，甚至损坏而无法使用，因此闪存设备的驱动程序需要将抹写的区块分散，以延长闪存寿命。用于闪存的文件系统，也需要设计出平均写入各区块的功能。
- 随机存取（Random access）：一般的硬盘，读写数据时，需要旋转磁盘，以找到存放的扇区，因此，一般使用于磁盘的文件系统，会作优化，以避免搜索磁盘的作用。但是闪存可以随机存取，没有查找延迟时间，因此不需要这个优化。

尽管磁盘文件系统也能在闪存上使用，但闪存文件系统是闪存设备的首选[6]，理由如下：

- 擦除区块：闪存的区块在重新写入前必须先进行擦除。擦除区块会占用相当可观的时间。因此，在设备空闲的时候擦除未使用的区块有助于提高速度。
- 随机访问：由于在磁盘上寻址有很大的延迟，磁盘文件系统有针对寻址的优化，以尽量避免寻址，但闪存没有寻址延迟。
- 写入平衡（Wear levelling）：闪存中经常写入的区块往往容易损坏。闪存文件系统的设计可以使数据均匀地写到整个设备。

设计闪存文件系统的基本概念是，当存储数据需要更新时，文件系统将会把新的复本写入一个新的闪存区块，将文件指针重新指向，并在闲置时期将原有的区块抹除，例如 JFFS2 和 YAFFS 等日志文件系统具有闪存文件系统需要的特性，而 exFAT 则是为了避免日志频繁写入而导致闪存寿命衰减的非日志文件系统。

[6]计算机上通行的大部份文件系统都是针对磁盘存储设备设计的，应用到闪存上并不适合。一般的文件系统可以通过闪存转换层（Flash Translation Layer，FTL）写入闪存，但是它的缺点是写入的效率较差。因此设计闪存文件系统仍然是有必要的。

在 1990 年代由微软所研发的闪存文件系统 FFS2（Flash File System 2）被应用在 MS-DOS 上。后来 PCMCIA 组织通过了闪存转换层（Flash Translation Layer，FTL）的规格，允许 Linear Flash 设备能够看起来像是 FAT 磁盘设备，但是仍然保有耗损平均技术的能力。

在 Linux 上实现的闪存转换层被称为 MTD。MTD 是一个硬件的抽象层，能够让闪存设备看起来像是一种区块设备，因此能够将既有的文件系统（如 FAT、Ext、XFS 等）直接应用在闪存上。

25.1 UNIX Filesystem

The Unix file system (UFS; also called the Berkeley Fast File System, the BSD Fast File System or FFS) is a file system used by many Unix and Unix-like operating systems. It is a distant descendant of the original filesystem used by Version 7 Unix.

In Unix and operating systems inspired by it, the file system[4] is considered a central component of the operating system. It was also one of the first parts of the system to be designed and implemented by Ken Thompson in 1969. Like in other operating systems, it provides information storage and interprocess communication, in the sense that the many small programs that traditionally comprise a Unix system can store information in files so that other programs can read these, although pipes complemented it in this role starting with the Third Edition.

The rest of this article uses "Unix" as a generic name to refer to both the original Unix operating system as well as its many workalikes.

The original Unix file system supported three types of files:

- ordinary files
- directories
- "special files", also known as device files

The Berkeley Software Distribution added sockets and symbolic links to this, and modern Unix system may support additional types of files. The filesystem appears as a single rooted tree of directories. Instead of addressing separate volumes such as disk partitions, removable media, and network shares as separate trees (as done in MS-DOS and Windows: each "drive" has a drive letter that denotes the root of its file system tree), such volumes can be "mounted" on a directory, causing the volume's file system tree to appear as that directory in the larger tree. The root of the entire tree is denoted `/`.

In the original Bell Labs Unix, a two-disk setup was customary, where the first disk contained startup programs, while the second contained users' files and programs. This second disk was mounted at the empty directory named `usr` on the first disk, causing the two disks to appear as one filesystem, with the second's disks contents viewable at `/usr`.

Unix directories do not "contain" files. Instead, they contain the names of files paired with references to so-called inodes, which in turn contain both the file and its metadata (owner, permissions, time of last access, etc., but no name). Multiple names in the file system may refer to the same file, a feature known as (hard) linking. (If this feature is taken into account, the file system is a limited type of directed acyclic graph rather than a tree. As originally envisioned ca. 1968-69, the file system would in fact be a more general type of directed graph, but navigating this turned out to be difficult, and the decision was made to disallow multiple links to a single directory, enforcing a tree structure.)

Certain conventions exist for locating particular kinds of files, such as programs, system configuration files and users' home directories. These were first documented in the hier(7) man page since Version 7 Unix; subsequent versions, derivatives and clones typically have a similar man page.

The details of the directory layout have varied over time. Although the file system layout is not part of the Single UNIX Specification, several attempts exist to standardize it, such as the Linux Foundation's Filesystem Hierarchy Standard (FHS).

25.1.1 History

Early versions of Unix filesystems were referred to simply as *FS*. FS only included the boot block, superblock, a clump of inodes, and the data blocks. This worked well for the small disks early Unixes were designed for, but as technology advanced and disks grew larger, moving the head back and forth between the clump of inodes and the data blocks they referred to caused thrashing. Marshall Kirk McKusick, then a Berkeley graduate student, optimized the BSD 4.2's FFS (Fast File System) by inventing cylinder groups, which break the disk up into smaller chunks, with each group having its own inodes and data blocks.

The intent of BSD FFS is to try to localize associated data blocks and metadata in the same cylinder group; and, ideally, all of the contents of a directory (both data and metadata for all the files) in the same or nearby cylinder group, thus reducing fragmentation caused by scattering a directory's contents over a whole disk.

Some of the performance parameters in the superblock included number of tracks and sectors, disk rotation speed, head speed, and alignment of the sectors between tracks. In a fully optimized system, the head could be moved between close tracks to read scattered sectors from alternating tracks while waiting for the platter to spin around.

As disks grew larger and larger, sector level optimization became obsolete (especially with disks that used linear sector numbering and variable sectors per track). With larger disks and larger files, fragmented reads became more of a problem. To combat this, BSD originally increased the filesystem block size from one sector to 1K in 4.0BSD; and, in FFS, increased the filesystem block size from 1K to 8K. This has several effects. The chances of a file's sectors being contiguous is much greater. The amount of overhead to list the file's blocks is reduced, while the number of bytes representable by any given number of blocks is increased.

Larger disk sizes are also possible, since the maximum number of blocks is limited by a fixed bit-width block number. However, with larger block sizes, disks with many small files will waste space, since each file must occupy at least one block. Because of this, BSD added block level fragmentation, also called block suballocation, tail merging or tail packing, where the last partial block of data from several files may be stored in a single "fragment" block instead of multiple mostly empty blocks.

25.1.2 Implementations

Vendors of some proprietary Unix systems, such as SunOS / Solaris, System V Release 4, HP-UX, and Tru64 UNIX, have adopted UFS. Most of them adapted UFS to their own uses, adding proprietary extensions that may not be recognized by other vendors' versions of Unix. Surprisingly, many have continued to use the original block size and data field widths as the original UFS, so some degree of (read) compatibility remains across platforms. Compatibility between implementations as a whole is spotty at best and should be researched before using it across multiple platforms where shared data is a primary intent.

As of Solaris 7, Sun Microsystems included UFS Logging, which brought filesystem journaling to UFS, which is still available in current versions of Solaris. Solaris UFS also has extensions for large files and large disks and other features.

In 4.4BSD and BSD Unix systems derived from it, such as FreeBSD, NetBSD, OpenBSD, and DragonFlyBSD, the implementation of UFS1 and UFS2 is split into two layers: an upper layer that provides the directory structure and supports metadata (permissions, ownership, etc.) in the inode structure, and lower layers that provide data containers implemented as inodes. This was done to support both the traditional FFS and the LFS log-structured file system with shared code for common functions. The upper layer is called "UFS", and the lower layers are called "FFS" and "LFS". In some of those systems, the term "FFS" is used for the combination of the FFS lower layer and the UFS upper layer, and the term "LFS" is used for the combination of the LFS lower layer and the UFS upper layer.

Kirk McKusick implemented block reallocation, a technique that reorders the blocks in the file system just before the writes are done to reduce fragmentation and control file system aging. He also implemented soft updates,

a mechanism that maintains the file system consistency without limiting the performance in the way the traditional sync mode did. This has the side effect of reducing the requirement of file system checking after a crash or power failure. To overcome the remaining issues after a failure, a background fsck utility was introduced.

In UFS2, Kirk McKusick and Poul-Henning Kamp extended the FreeBSD FFS and UFS layers to add 64-bit block pointers (allowing volumes to grow up to 8 zettabytes), variable-sized blocks (similar to extents), extended flag fields, additional 'birthtime' stamps, extended attribute support and POSIX1.e ACLs. UFS2 became the default UFS version starting with FreeBSD 5.0. FreeBSD also introduced soft updates and the ability to make file system snapshots for both UFS1 and UFS2. These have since been ported to NetBSD, but eventually soft updates (called soft dependencies in NetBSD) was removed from NetBSD 6.0 in favor of the less complex file system journaling mechanism called WAPBL (also referred as logging), which was added to FFS in NetBSD 5.0. OpenBSD has supported soft updates since version 2.9 and has had UFS2 (FFS2) support (no ACLs) since version 4.2. Since FreeBSD 7.0, UFS also supports filesystem journaling using the gjournal GEOM provider. FreeBSD 9.0 adds support for lightweight journaling on top of softupdates(SU+J), which greatly reduces the need for background fsck, and uses NFS-style ACLs by default.

FreeBSD, NetBSD, OpenBSD, and DragonFlyBSD also include the Dirhash system, developed by Ian Dowse. This system maintains an in-memory hash table to speed up directory lookups. Dirhash alleviates a number of performance problems associated with large directories in UFS.

Linux includes a UFS implementation for binary compatibility at the read level with other Unixes, but since there is no standard implementation for the vendor extensions to UFS, Linux does not have full support for writing to UFS. The native Linux ext2 filesystem was inspired by UFS1 but does not support fragments and there are no plans to implement softupdates. (In some 4.4BSD-derived systems, the UFS layer can use an ext2 layer as a container layer, just as it can use FFS and LFS.)

NeXTStep, which was BSD-derived, also used a version of UFS. In Apple's Mac OS X, it was available as an alternative to HFS+, their proprietary filesystem. However, as of Mac OS X Leopard, it was no longer possible to install Mac OS X on a UFS-formatted volume. In addition, one cannot upgrade older versions of Mac OS X installed on UFS-formatted volumes to Leopard; upgrading requires reformatting the startup volume. There was a 4GB file limit for disks formatted as UFS in Mac OS X. As of Mac OS X Lion, UFS support was completely dropped.

25.1.3 Composition

The structures of UFS directory contents is table, and its max file size is 2^{73}bytes=8ZB. The max filename length is 255bytes.

A UFS volume's max size is 2^{73}bytes=8ZB, and it is composed of the following parts:

- A few blocks at the beginning of the partition reserved for boot blocks (which must be initialized separately from the filesystem)
- A superblock, containing a magic number identifying this as a UFS filesystem, and some other vital numbers describing this filesystem's geometry and statistics and behavioral tuning parameters
- A collection of cylinder groups. Each cylinder group has the following components:
  - A backup copy of the superblock
  - A cylinder group header, with statistics, free lists, etc., about this cylinder group, similar to those in the superblock
  - A number of inodes, each containing file attributes
  - A number of data blocks

Inodes are numbered sequentially, starting at 0. The first two inodes are reserved for historical reasons, followed by the inode for the root directory, which is always inode 2.

Directory files contain only the list of filenames in the directory and the inode associated with each file. All file metadata is kept in the inode.

Here is a generalized overview of common locations of files on a Unix operating system:

Table 25.1: UNIX Filesystem Hierarchy Standard

Directory	Description
/	The slash / character alone denotes the root of the filesystem tree.
/bin	Stands for ”binaries” and contains certain fundamental utilities, such as ls or cp, which are generally needed by all users.
/boot	Contains all the files that are required for successful booting process.
/dev	Stands for ”devices”. Contains file representations of peripheral devices and pseudo-devices.
/etc	Contains system-wide configuration files and system databases. Originally also contained ”dangerous maintenance utilities” such as init,[9] but these have typically been moved to /sbin or elsewhere.
/home	Contains the home directories for the users. In the original version of Unix, home directories were in /usr instead.[10] Some systems use or have used different locations still: OS X has home directories in /Users, and older versions of BSD put them in /u.
/lib	Contains system libraries, and some critical files such as kernel modules or device drivers.
/lib64	Contains system libraries, and some critical files such as kernel modules or device drivers, on some 64-bit systems.
/media	Default mount point for removable devices, such as USB sticks, media players, etc.
/mnt	Stands for ”mount”. Contains filesystem mount points. These are used, for example, if the system uses multiple hard disks or hard disk partitions. It is also often used for remote (network) filesystems, CD-ROM/DVD drives, and so on.
/opt	Contains locally installed software. Originated in System V, which has a package manager that installs software to this directory (one subdirectory per package).
/proc	procfs virtual filesystem showing information about processes as files.
/root	The home directory for the superuser ”root” - that is, the system administrator. This account's home directory is usually on the initial filesystem, and hence not in /home (which may be a mount point for another filesystem) in case specific maintenance needs to be performed, during which other filesystems are not available. Such a case could occur, for example, if a hard disk drive suffers physical failures and cannot be properly mounted.
/sbin	Stands for ”system (or superuser) binaries” and contains fundamental utilities, such as init, usually needed to start, maintain and recover the system.
/srv	Server data (data for services provided by system).
/sys	In some Linux distributions, contains a sysfs virtual filesystem, containing information related to hardware and the operating system. On BSD systems, commonly a symlink to the kernel sources in /usr/src/sys.
/tmp	A place for temporary files. Many systems clear this directory upon startup; it might have tmpfs mounted atop it, in which case its contents do not survive a reboot, or it might be explicitly cleared by a startup script at boot time.

Directory	Description
/usr	Originally the directory holding user home directories, its use has changed. It now holds executables, libraries, and shared resources that are not system critical, like the X Window System, KDE, Perl, etc. However, on some Unix systems, some user accounts may still have a home directory that is a direct subdirectory of /usr, such as the default in Minix. (on modern systems, these user accounts are often related to server or system use, and not directly used by a person)
/usr/bin	This directory stores all binary programs distributed with the operating system not residing in /bin, /sbin or (rarely) /etc.
/usr/include	Stores the development headers used throughout the system. Header files are mostly used by the #include directive in C programming language, which historically is how the name of this directory was chosen.
/usr/lib	Stores the required libraries and data files for programs stored within /usr or elsewhere.
/usr/libexec	On BSD systems and older distributions of Linux, programs that are meant to be executed by other programs rather than by users directly. E.g., the Sendmail executable may be found in this directory. Not present in the FHS; Linux distributions have typically moved the contents of this directory into /usr/lib, where they also resided in 4.3BSD.
/usr/local	Resembles /usr in structure, but its subdirectories are used for additions not part of the operating system distribution, such as custom programs or files from a BSD Ports collection. Usually has subdirectories such as /usr/local/lib or /usr/local/bin.
/var	A short for ”variable.” A place for files that may change often - especially in size, for example e-mail sent to users on the system, or process-ID lock files.
/var/log	Contains system log files.
/var/mail	The place where all the incoming mails are stored. Users (other than root) can access their own mail only. Often, this directory is a symbolic link to /var/spool/mail.
/var/spool	Spool directory. Contains print jobs, mail spools and other queued tasks.
/var/tmp	A place for temporary files which should be preserved between system reboots.

25.2 Linux Filesystem

Linux supports many different file systems, but common choices for the system disk on a block device include the ext* family (such as ext2, ext3 and ext4), XFS, JFS, ReiserFS and btrfs. For raw flash without a flash translation layer (FTL) or Memory Technology Device (MTD), there is UBIFS, JFFS2, and YAFFS, among others. SquashFS is a common compressed read-only file system.

第 26 章

FHS

The Filesystem Hierarchy Standard (FHS)[1] defines the directory structure and directory contents in Unix and Unix-like operating systems, maintained by the Linux Foundation. The current version is 2.3, announced on 29 January 2004.

When the FHS was created, other UNIX and Unix-like operating systems already had their own standards. Notable examples are these: the hier(7) description of file system layout, which has existed since the release of Version 7 Unix (in 1979); the SunOS filesystem(7) and its successor, the Solaris filesystem(5).

FHS 2.0 is the direct successor for FSSTND 1.2, name of the standard was changed to Filesystem Hierarchy Standard on Oct 26, 1997. From then on, most Linux distributions follow the FHS and declare it their own policy to maintain FHS compliance, and some distributions that generally follow the standard deviate from it in some areas. Common deviations include:

- Modern Linux distributions include a `/sys` directory as a virtual filesystem (sysfs, comparable to `/proc`, which is a procfs), which stores and allows modification of the devices connected to the system, whereas many traditional UNIX and Unix-like operating systems use `/sys` as a symbolic link to the kernel source tree.
- Modern Linux distributions include a `/run` directory as a temporary filesystem (tmpfs) which stores volatile runtime data, and which is being considered for the next version of the FHS. According to the FHS version 2.3, this data should be stored in `/var/run` but this was a problem in some cases because this directory isn't always available at early boot. As a result, these programs have had to resort to such trickery as using `/dev/.udev`, `/dev/.mdadm`, `/dev/.systemd` or `/dev/.mount` directories, even though the device directory isn't intended for such data. Among other advantages, this makes the system easier to use normally with the root filesystem mounted read-only. This is a detailed example from Debian:
  - /dev/.* → /run/*
  - /dev/shm → /run/shm
  - /dev/shm/* → /run/*
  - /etc/* (writeable files) → /run/*
  - /lib/init/rw → /run
  - /var/lock → /run/lock
  - /var/run → /run
  - /tmp → /run/tmp
- Many modern UNIX systems (like FreeBSD via its ports system) install third party packages into `/usr/local` while keeping locally developed code in `/usr`.
- Some Linux distributions no longer differentiate between `/lib` versus `/usr/lib` and have `/lib` symlinked to `/usr/lib`.
- Some Linux distributions no longer differentiate between `/bin` versus `/usr/bin` and `/sbin` versus `/usr/sbin`. They symlink `/bin` to `/usr/bin` and `/sbin` to `/usr/sbin`. And `/usr/sbin` may get symlinked to `/usr/bin`.

In the FHS all files and directories appear under the root directory "`/`", even if they are stored on different physical or virtual devices. Note however that some of these directories may or may not be present on a Unix system depending on whether certain subsystems, such as the X Window System, are installed.

The majority of these directories exist in all UNIX operating systems and are generally used in much the same way; however, the descriptions here are those used specifically for the FHS, and are not considered authoritative for platforms other than Linux.

Table 26.1: Linux Filesystem Hierarchy Standard

Directory	Description
/	Primary hierarchy root and root directory of the entire file system hierarchy.
/bin	Essential command binaries that need to be available in single user mode; for all users, e.g., cat, ls, cp.
/boot	Boot loader files, e.g., kernels, initrd.
/dev	Essential devices, e.g., /dev/null.
/etc	Host-specific system-wide configuration files[1]
/etc/opt	Configuration files for /opt/.
/etc/sgml	Configuration files for SGML.
/etc/X11	Configuration files for the X Window System, version 11.
/etc/xml	Configuration files for XML.
/home	Users' home directories, containing saved files, personal settings, etc.
/lib	Libraries essential for the binaries in /bin/ and /sbin/.
/lib<qual>	Alternate format essential libraries. Such directories are optional, but if they exist, they have some requirements.
/media	Mount points for removable media such as CD-ROMs (appeared in FHS-2.3).
/mnt	Temporarily mounted filesystems.
/opt	Optional application software packages.
/proc	Virtual filesystem providing information about processes and kernel information as files. In Linux, corresponds to a procfs mount.
/root	Home directory for the root user.
/sbin	Essential system binaries, e.g., init, ip, mount.
/srv	Site-specific data which are served by the system.
/tmp	Temporary files (see also /var/tmp). Often not preserved between system reboots.
/usr	Secondary hierarchy for read-only user data; contains the majority of (multi-)user utilities and applications.
/usr/bin	Non-essential command binaries (not needed in single user mode); for all users.
/usr/include	Standard include files.
/usr/lib	Libraries for the binaries in /usr/bin/ and /usr/sbin/.
/usr/lib<qual>	Alternate format libraries (optional).
/usr/local	Tertiary hierarchy for local data, specific to this host. Typically has further subdirectories, e.g., bin/, lib/, share/.
/usr/sbin	Non-essential system binaries, e.g., daemons for various network-services.
/usr/share	Architecture-independent (shared) data.

[1] There has been controversy over the meaning of the name itself. In early versions of the UNIX Implementation Document from Bell labs, `/etc` is referred to as the etcetera directory, as this directory historically held everything that did not belong elsewhere (however, the FHS restricts `/etc` to static configuration files and may not contain binaries). Since the publication of early documentation, the directory name has been re-designated in various ways. Recent interpretations include backronyms such as "Editable Text Configuration" or "Extended Tool Chest".

Directory	Description
/usr/src	Source code, e.g., the kernel source code with its header files.
/usr/X11R6	X Window System, Version 11, Release 6.
/var	Variable files—files whose content is expected to continually change during normal operation of the system—such as logs, spool files, and temporary e-mail files.
/var/cache	Application cache data. Such data are locally generated as a result of time-consuming I/O or calculation. The application must be able to regenerate or restore the data. The cached files can be deleted without loss of data.
/var/lib	State information. Persistent data modified by programs as they run, e.g., databases, packaging system metadata, etc.
/var/lock	Lock files. Files keeping track of resources currently in use.
/var/log	Log files. Various logs.
/var/mail	Users' mailboxes.
/var/run	Information about the running system since last boot, e.g., currently logged-in users and running daemons.
/var/spool	Spool for tasks waiting to be processed, e.g., print queues and outgoing mail queue.
/var/spool/mail	Deprecated location for users' mailboxes.
/var/tmp	Temporary files to be preserved between reboots.

Unix 文件系统(Unix File System, UFS)是一种为许多 UNIX 和类 Unix 操作系统所使用的文件系统,也被称为伯克利快速文件系统(Berkeley Fast File System)或者 BSD 快速文件系统(BSD Fast File System),缩写为 FFS。

FFS 对 Unix System V 文件系统“FS”有所继承, FFS 在 Unix 文件系统(1 或 2)之上并提供目录结构的信息,以及各种磁盘访问的优化。UFS(以及 UFS2)定义了 on-disk 数据规格。

Linux 系统并不是特指某一个操作系统,而是指使用了由 Linus Torvalds(林纳斯·托瓦兹)发明并领导开发的 Linux 内核的所有操作系统。这里, Linux 仅仅指的是该系统的内核,单独的一个 Linux 基本上是无法运行任何程序的。

由不同的团队开发出来的基于 Linux 系统自然有很多地方是无法统一的,而且并且计算机配置与使用方法完全不统一,因此 Linux 基金会发布了 Linux 标准规范 LSB (Linux Standard Base) 用以规范 Linux 的开发,其中就规定了 Linux 的文件系统层次结构标准(Filesystem Hierarchy Standard, 缩写为 FHS)。

文件系统层次结构标准[3] 定义了 Linux 操作系统中的主要目录及目录内容,例如在/(根目录)下的各个主要目录应该存放的主要文件内容,此外还专门定义了/usr 和/var 两个目录及其子目录的结构。在大多数情况下, FHS 是一个传统 BSD 文件系统层次结构的形式化与扩充。

多数 Linux 发行版遵从 FHS 标准并且声明其自身政策以维护 FHS 的要求,例如 Linux 系统采用的是树状存储结构,在 Linux 中所有文件与目录都是由/(根)开始的。然而,包括由自由标准小组成员在内开发的绝大多数发行版,并不完全执行建议的标准。一些 Linux 系统如 GoboLinux 和 Syllable Server 使用了和 FHS 完全不同的文件系统层次组织方法。

现在的 Linux 发行版包含一个`/sys`目录作为虚拟文件系统 (sysfs, 类似于`/proc`, 一个 procfs), 它存储且允许修改连接到系统的设备,然而许多传统 UNIX 和类 Unix 操作系统使用`/sys`作为内核代码树的符号链接。

当 FHS 建立之时,其他的 UNIX 和类 Unix 操作系统已经有了自己的标准,尤其是 hier(7) 文件系统布局描述。自从第七版 Unix(于 1979 年) 发布以来已经存在,或是 SunOS filesystem(7), 和之后的 Solaris filesystem(5)。例如, Mac OS X 使用如 /Library、/Applications 和/Users 等长名与传统 UNIX 目录层次保持一致。

在 FHS 中,所有的文件和目录都出现在根目录“/”下,即使它们存储在不同的物理设备中。但是这些目录中的一些可能或可能不会在 Unix 系统上出现,这取决于系统是否含有某些子系统,例如 X Window 系统的安装与否。

这些目录中的绝大多数都在所有的 UNIX 操作系统中存在,并且一般都以大致类似的方法使用。然而,这里的描述是针对于 FHS 的,并未考虑除了 Linux 平台以外的权威性。

Table 26.2: Linux Filesystem Hierarchy Standard

目录	描述
/	第一层次结构的根, 整个文件系统层次结构的根目录, 所有文件、文件夹的入口。
/bin/	需要在单用户模式可用的必要命令(面向所有用户的可执行文件),例如 cat、ls、cp。
/boot/	存放引导程序文件与内核, 例如 kernel、initrd, 通常是一个单独的分区
/dev/	设备目录, 例如/dev/null、/dev/sda1[2]
/etc/[3]	特定主机, 系统范围内的配置文件。

[2]在 Linux 所有设备也都是以文件的形式出现的,打开/dev/sda1 就是打开了硬盘的第一个分区。

[3]关于这个名称目前有争议。在贝尔实验室关于 UNIX 实现文档的早期版本中, /etc 被称为 etcetera, 这是由于过去此目录中存放所有不属于别处的所有东西(然而 FHS 限制/etc 存放静态配置文件,不能包含二进制文件)。自从早期文档出版以来,目录

目录	描述
/etc/opt/	/opt/的配置文件
/etc/X11/	X Window 系统 (版本 11) 的配置文件
/etc/sgml/	SGML 的配置文件
/etc/xml/	XML 的配置文件
/home/	用户的家目录，包含保存的文件、个人设置等，一般为单独的分区。
/lib/	/bin/和/sbin/中二进制文件必要的库文件。
/media/	以前是挂接外部存储器的，现在是可移除媒体 (如 CD-ROM) 的挂载点。
/mnt/	临时挂载的文件系统或外接设备目录，例如移动硬盘、U 盘的内容在其子目录下存放
/opt/	可选应用软件包。
/proc/	虚拟文件系统，将内核与进程状态归档为文本文件。例如 uptime、network。在 Linux 中，对应 procfs 格式挂载。
/root/	超级用户的家目录
/sbin/	必要的系统二进制文件，同时也是管理员使用的命令，例如 init、ip、mount
/srv/	站点的具体数据，由系统提供
/tmp/	临时文件 (参见/var/tmp)，在系统重启时目录中文件不会被保留
/usr/	用于存储只读用户数据的第二层次，包含绝大多数的 (多) 用户工具和应用程序
/usr/bin/	非必要可执行文件 (在单用户模式中不需要)，面向所有用户
/usr/include/	标准包含文件
/usr/lib/	/usr/bin/和/usr/sbin/中二进制文件的库
/usr/sbin/	非必要的系统二进制文件，例如大量网络服务的守护进程
/usr/share/	体系结构无关(共享)的数据
/usr/src/	源代码，例如内核源代码及其头文件
/usr/X11R6/	X Window R11, Release 6
/usr/local/	本地数据的第三层次，具体到特定主机。通常而言有进一步的子目录，例如 bin/、lib/、share/
/var/	变量文件——在正常运行的系统中其内容不断变化的文件，如日志、脱机文件和临时电子邮件文件，有时是一个单独的分区。
/var/cache/	应用程序缓存数据。这些数据是在本地生成的一个耗时的 I/O 或计算结果，应用程序必须能够再生或恢复数据，缓存的文件可以被删除而不导致数据丢失。
/var/lib/	状态信息。由程序在运行时维护的持久性数据。例如数据库、包装的系统元数据等
/var/lock/	锁文件，一类跟踪当前使用中资源的文件
/var/log/	日志文件，包含大量日志文件
/var/mail/	用户的电子邮箱

名称已被以各种方式重新称呼。最近的解释包括反向缩略语如："可编辑的文本配置"（Editable Text Configuration）或 "扩展工具箱"（Extended Tool Chest）。

目录	描述
/var/run/	自最后一次启动以来运行中的系统的信息，例如当前登录的用户和运行中的守护进程。现已经被/run 代替
/var/spool/	等待处理的任务的脱机文件，例如打印队列和未读的邮件
/var/spool/mail/	用户的邮箱 (不鼓励的存储位置)
/var/tmp/	在系统重启过程中可以保留的临时文件
/run	代替/var/run 目录

开发一套文件系统层次结构标准的进程始于 1993 年 8 月，标准努力重整 Linux 的文件和目录结构。FSSTND (Filesystem Standard)，一个针对 Linux 操作系统的文件系统层次结构标准在 1994 年 2 月 14 日发布。后续的修正版本分别在 1994 年 10 月 9 日和 1995 年 3 月 28 日发布。

在 1996 年初，开发一个更加全面的、不仅解决 Linux，而且解决其他类 Unix 系统目录层次结构问题的 FSSTND 的计划在 BSD 开发社区成员的协助下正式被采纳。因此，计划重点解决在类 Unix 系统上普遍存在的问题。为了适应标准范围的扩充，标准的名称修改为文件系统层次结构标准。

FHS 现在由 Linux 基金会维护，这是一个由主要软件或硬件供应商组成的非营利组织，例如 HP、Red Hat、IBM、和 Dell。当前的 FHS 版本是 2.3，在 2004 年 1 月 29 日公布。

26.1 Linux

目前的操作系统大多数是将数据由硬盘读出来，那么每个操作系统使用的硬盘在 x86 架构上都是一样，但是每种操作系统都有其独特的读取文件的方法，也就是说，每种操作系统对硬盘读取的方法不同，所以就出现了不同的文件系统。

举例来说，Windows 98 默认的文件系统是 FAT(或 FAT16)文件系统，Windows 2000 有 NTFS 文件系统，Linux 的正统文件系统则为 ext2(Linux second extended file system，ext2fs)。

系统能不能读取某个文件系统，与“核心功能”有关，Linux 核心必须要能够认识某种文件系统才能读取该文件系统的数据内容，也就是说，必须要将所想要支持的文件系统编译到核心中才能被支持，因此可以发现 Windows 与 Linux 安装在同一个硬盘的不同 partition 时，Windows 将不能使用 Linux 的硬盘数据，就因为 Windows 的核心不认识 Linux 的文件系统。

目前 Fedora 默认的文件系统为 ext3(Third extended File System)，它是 ext2 的升级版，主要是增加了日志(journaling)的功能，但是 ext3 还是向下支持 ext2 的。

另外，如果需要将原有的 Windows 系统也挂载在 Linux 下，Linux 同时也支持 MS-DOS、VFATFAT、BSD 等的文件系统。Window NT 的 NTFS 文件系统则不见得每一个 Linux distribution 都有支持。

Linux 能够支持的文件系统与核心是否有编译进去有关，所以可以到 Linux 系统的：

```
/lib/modules/`uname -r`/kernel/fs
```

查看，如果有想要的文件系统，那么这个核心就有支持。很多 Linux 所需要的功能都可以在 ext2 上面完成，不过 ext2 缺乏日志管理系统，发生问题时修复过程会比较慢一些，所以最新的 Linux distribution 大多已经默认采用 ext3 或 reiserfs 这种具有日志式管理的文件系统了。

ext3 其实只是多做了一个日志式数据的记录。当要在将数据写入硬盘时，ext2 是直接将数据写入，但是 ext3 则会将这个“要开始写入”的信息写入日志式记录区，然后才开始进行数据的写入。在数据写入完毕后，又将“完成写入动作”的信息写入日志记录区，这有什么好处呢？最大的好处就是数据的完整性与“可恢复性”。

早期的 ext2 文件系统如果发生类似断电后时，文件系统就得要检查文件一致性。这个检查的过程要将整个 partition 内的文件做一个完整的比较，比较耗时间。

如果是 ext3，那么只要通过检查“日志记录区”就可以知道断电时，是否有哪些文件正在进行写入的动作运行，只要检查这些地方即可，这样就能够节省很多文件检查的时间。

还可以引用 Red Hat 公司首席核心开发者 Michael K. Johnson 的话：

“为什么你想要从 ext2 转换到 ext3 呢？有四个主要的理由：可利用性、数据完整性、速度及易于转换”。

可利用性指出，这意味著从系统中止到快速重新复原而不是持续的让 e2fsck 执行长时间的修复。ext3 的日志式条件可以避免数据毁损的可能，它也指出，“除了写入若干数据超过一次时，ext3 往往会较快于 ext2，因为 ext3 的日志使硬盘读取头的移动能更有效的进行”，然而或许决定的因素还是在 Johnson 先生的第四个理由中。

“它是可以轻易的从 ext2 更改到 ext3 来获得一个强而有力的日志式文件系统而不需要重新做格式化”。“那是正确的，为了体验一下 ext3 的好处是不需要去做一种长时间的，冗长乏味的且易于产生错误的备份工运行及重新格式化的动作运行”。

上列数据可在 Whitepaper: Red Hat's New Journaling File System: ext3 (`http://www.redhat.com/support/wpapers/redhat/ext3/`)查看得到，所以使用 ext3 或者是其它的日志式文件系统是有好处的，最大的好处当然是错误问题的排除效率比较高。

26.2 VFS

最初的 UNIX 系统一般都只支持一种单一类型的文件系统，在这种情况下，文件系统的结构会深入到整个系统内核中。Sun 在 1985 年开发的 SunOS 2.0 实现了第一个虚拟文件系统，之后被加入到 UNIX System V 第四版中，它让 UNIX 的系统调用可以适用于本地端的 UFS 以及远程的 NFS。

现在的操作系统大多都在系统内核和文件系统之间提供一个标准的接口，这样不同文件结构之间的数据可以十分方便地交换，因此 Linux 也在系统内核和文件系统之间提供了一种叫做 VFS(Virtual Filesystem Switch)的标准接口。

Linux 操作系统通过 VFS 的核心功能去读取文件系统，也就是说，整个 Linux 认识的文件系统其实都是 VFS 在进行管理，用户并不需要知道每个分区上的文件系统是什么，VFS 会主动的帮用户做好读取的工作。

只要系统管理员一开始就设定好各主要文件系统对应的内核模块后，核心的 VFS 就会主动接管该文件系统的存取工作，用户可以在不知道每个文件系统是什么的情况下就能自由的使用系统中的各种文件系统。

虚拟文件系统又称为虚拟文件切换系统(virtual filesystem switch)，是操作系统的文件系统虚拟层，在其下是实体的文件系统。虚拟文件系统的主要功用在于让上层的软件能够用单一的方式来跟底层不同的文件系统沟通。在操作系统与之下的各种文件系统之间，虚拟文件系统提供了标准的操作接口，让操作系统能够很快的支持新的文件系统。

这样，文件系统的代码就分成了两部分：

- 上层用于处理系统内核的各种表格和数据结构；
- 下层用来实现文件系统本身的函数，并通过 VFS 来调用。

这些函数主要包括：

- 管理缓冲区 (buffer. c)；
- 响应系统调用 fcntl() 和 ioctl()(fcntl.c and ioctl.c)；
- 将管道和文件输入/输出映射到索引节点和缓冲区 (fifo.c, pipe.c)；
- 锁定和不锁定文件和记录 (locks.c)；
- 映射名字到索引节点 (namei.c, open.c)；
- 实现 select( ) 函数 (select . c)；
- 提供各种信息 (stat.c)；
- 挂接和卸载文件系统 (super.c)；

- 调用可执行代码和转存核心 (exec.c);
- 装载各种二进制格式 (bin_fmt*.c);

VFS 接口则由一系列相对高级的操作组成, 这些操作由和文件系统无关的代码调用, 并且由不同的文件系统执行。其中最主要的结构有 inode_operations 和 file_operations。file_system_type 是系统内核中指向真正文件系统的结构。每挂接一次文件系统, 都将使用 file_system_type 组成的数组。file_system_type 组成的数组嵌入到了 fs/filesystems.c 中。相关文件系统的 read_super 函数负责填充 super_block 结构。

26.3 XFS

XFS[6] 是由 Silicon Graphics 为 IRIX 操作系统而开发的高性能的日志文件系统。XFS 的开发始于 1993 年, 在 1994 年被首次部署在 IRIX 5.3 上并作为其默认文件系统。2000 年 5 月, Silicon Graphics 以 GNU 通用公共许可证发布这套系统的源代码, 之后被移植到 Linux 内核上。现在所有的 Linux 发行版上都可以使用 XFS, Red Hat Enterprise Linux 7 将默认使用 XFS 文件系统。

XFS 最初被合并到 Linux 2.4 主线中, 这使得 XFS 几乎可以被用在任何一个 Linux 系统上, 在 Linux 发行版的安装程序中都可以选择 XFS 作为文件系统, 但由于默认的启动管理器 GRUB 中的限制, 只有少数 Linux 发行版允许用户在 /boot 挂载点(引导目录)上使用 XFS 文件系统。

26.3.1 Capacity

XFS 是一个 64 位文件系统, 最大支持 8exbibytes 减 1 字节的单个文件系统, 因此 XFS 特别擅长处理大文件, 同时提供平滑的数据传输, 实际部署时取决于宿主操作系统的最大块限制。对于一个 32 位 Linux 系统, 文件和文件系统的大小会被限制在 16tebibytes。

26.3.2 Journaling

日志文件系统是一种即使在断电或者是操作系统崩溃的情况下保证文件系统一致性的途径。XFS 对文件系统元数据提供了日志支持。当文件系统更新时, 元数据会在实际的磁盘块被更新之前顺序写入日志。XFS 的日志被保存在磁盘块的循环缓冲区上, 不会被正常的文件系统操作影响。XFS 日志大小的上限是 64k 个块和 128MB 中的较大值, 下限取决于已存在的文件系统和目录的块的大小。在外置设备上部署日志会浪费超过最大日志大小的空间。XFS 日志也可以被存在文件系统的数据区(称为内置日志), 或者一个额外的设备上(以减少磁盘操作)。

XFS 的日志保存的是在更高层次上描述已进行的操作的“逻辑”实体。相比之下, “物理”日志存储每次事务中被修改的块。为了保证性能, 日志的更新是异步进行的。当系统崩溃时, 崩溃的一瞬间之前所进行的所有操作可以利用日志中的数据重做, 这使得 XFS 能保持文件系统的一致性。XFS 在挂载文件系统的同时进行恢复, 恢复速度与文件系统的大小无关。对于最近被修改但未完全写入磁盘的数据, XFS 保证在重启时清零所有未被写入的数据块, 以防止任何有可能的、由剩余数据导致的安全隐患(因为虽然从文件系统接口无法访问这些数据, 但不排除裸设备或裸硬件被直接读取的可能性)。

历史上 XFS 上的元数据操作曾比其它文件系统都慢, 表现为在删除大量小文件时性能糟糕。该性能问题是被 Red Hat 的 XFS 开发者 Dave Chinner 在代码中定位到的。使用一个叫“延迟记录”的挂载选项可以成数量级地提升元数据操作的性能。该选项几乎把日志整个存在内存中。Linux 内核主线版本 2.6.35 中作为一个试验性特性引入了这个补丁, 在 2.6.37 中使它成为了一个稳定的特性, 并计划在 2.6.39 中把它作为默认的日志记录方法。早期测试显示在有少量线程的环境中其性能接近 EXT4, 在大量线程的环境下超过了 EXT4。

26.3.3 Allocation Groups

XFS 文件系统内部被分为多个“分配组”,它们是文件系统中的等长线性存储区。每个分配组各自管理自己的 inode 和剩余空间。文件和文件夹可以跨越分配组。这一机制为 XFS 提供了可伸缩性和并行特性——多个线程和进程可以同时在同一个文件系统上执行 I/O 操作。这种由分配组带来的内部分区机制在一个文件系统跨越多个物理设备时特别有用,使得优化对下级存储部件的吞吐量利用率成为可能。

26.3.4 Striped Allocation

在条带化 RAID 阵列上创建 XFS 文件系统时,可以指定一个“条带化数据单元”。这可以保证数据分配、inode 分配、以及内部日志被对齐到该条带单元上,以此最大化吞吐量。

26.3.5 Extent Based Allocation

XFS 文件系统中的文件用到的块由变长 Extent 管理,每一个 Extent 描述了一个或多个连续的块。与那些把文件所有块都单独列出来的文件系统来说,extent 大幅缩短了列表。

有些文件系统用一个或多个面向块的位图管理空间分配——在 XFS 中这种结构被由一对 B+ 树组成的、面向 Extent 的结构替代了;每个文件系统分配组 (AG) 包含这样的一个结构。其中,一个 B+ 树用于索引未被使用的 Extent 的长度,另一个索引这些 Extent 的起始块。这种双索引策略使得文件系统在定位剩余空间中的 Extent 时十分高效。

26.3.6 Variable Block Sizes

块是文件系统中的最小可分配单元。XFS 允许在创建文件系统时指定块的大小,从 512 字节到 64KB,以适应专门的用途。比如,对于有很多小文件的应用,较小的块尺寸可以最大化磁盘利用率;但对于一个主要处理大文件的系统,较大的块尺寸能提供更好的性能。

26.3.7 Delayed Allocation

XFS 在文件分配上使用了惰性计算技术。当一个文件被写入缓存时,XFS 简单地在内存中对该文件保留合适数量的块,而不是立即对数据分配 Extent。实际的块分配仅在这段数据被冲刷到磁盘时才发生。这一机制提高了将这一文件写入一组连续的块中的机会,减少碎片的同时提升了性能。

26.3.8 Sparse Files

XFS 对每个文件提供了一个 64 位的稀疏地址空间,使得大文件中的“洞”(空白数据区)不被实际分配到磁盘上。因为文件系统对每个文件使用一个 Extent 表,文件分配表就可以保持一个较小的体积。对于太大以至于无法存储在 inode 中的分配表,这张表会被移动到 B+ 树中,继续保持对该目标文件在 64 位地址空间中任意位置的数据的高效访问。

26.3.9 Extended Attributes

XFS 通过实现扩展文件属性给文件提供了多个数据流,使文件可以被附加多个名/值对。文件名是一个最大长度为 256 字节的、以 NULL 字符结尾的可打印字符串,其它的关联值则可包含多达 64KB 的二进制数据。这些数据被进一步分入两个名字空间中,root 和 user。保存在 root 名字空间中的扩展属性只能被超级用户修改,user 名字空间中的可以被任何对该文件拥有写权限的用户修改。扩展属性可以被添加到任意一种 XFS inode 上,包括符号链接、设备节点、目录,等等。可以使用 attr 这个命令行程序操作这些扩展属性。xfsdump 和 xfsrestore 工具在进行备份和恢复时会一同操作扩展属性,而其它的大多数备份系统则会忽略扩展属性。

26.3.10 Direct I/O

对于要求高吞吐量的应用，XFS 给用户空间提供了直接的、非缓存 I/O 的实现。数据在应用程序的缓冲区和磁盘间利用 DMA 进行传输，以此提供下级磁盘设备全部的 I/O 带宽。

26.3.11 Guaranteed-rate I/O

XFS 确定速率 I/O 系统给应用程序提供了预留文件系统带宽的 API。XFS 会动态计算下级存储设备能提供的性能，并在给定的时间内预留足够的带宽以满足所要求的性能。此项特性是 XFS 所独有的。确定方式可以是硬性的或软性的，前者提供了更高性能，而后者相对更加可靠。不过只要下级存储设备支持硬性速率确定，XFS 就只允许硬性模式。这一机制最常被用在实时应用中，比如视频流。

26.3.12 DMAPI

XFS 实现了数据管理应用程序接口 (DMAPI) 以支持高阶存储管理 (HSM)。到 2010 年 10 月为止，Linux 上的 XFS 实现已经支持 DMAPI 所要求的的磁盘元数据规范，但有报告称内核支持仍处于不稳定状态。此前 SGI 曾提供了一个包含 DMAPI 钩子的内核源码树，但这个支持未被合并进主代码树。不过现在内核开发者们已经注意到了它并对其做了更新。

26.3.13 Snapshots

XFS 并不直接提供对文件系统快照的支持，因为 XFS 认为快照可在卷管理器中实现。对一个 XFS 文件系统做快照需要调用 xfs_freeze 工具冻结文件系统的 I/O，然后等待卷管理器完成实际的快照创建，再解冻 I/O，继续正常的操作。之后这个快照可以被当作备份，以只读方式挂载。在 IRIX 上发布的 XFS 包含了一个集成的卷管理器，叫 XLV。这个卷管理器无法被移植到 Linux 上，不过 XFS 可以和 Linux 上标准的 LVM 正常工作。

在最近发布的 Linux 内核中，xfs_freeze 的功能被实现在了 VFS 层，当卷管理器的快照功能被唤醒时将自动启动 xfs_freeze。相对于无法挂起，卷管理器也无法对其创建“热”快照的 ext3 文件系统，XFS 的快照功能具有很大优势。幸运的是，现在这种情况已经改观。从 Linux 2.6.29 内核开始，ext3, ext4, gfs2 和 jfs 文件系统也获得了冻结文件系统的特性。

26.3.14 Online Defragmentation

虽然 XFS 基于 Extent 的特征和延迟分配策略显著提高了文件系统对碎片问题的抵抗力，XFS 还是提供了一个文件系统碎片整理工具，xfs_fsr（XFS filesystem reorganizer 的简称）。这个工具可以对一个已被挂载、正在使用中的 XFS 文件系统进行碎片整理。

26.3.15 Online Resizing

XFS 提供了 xfs_growfs 工具，可以在线调整 XFS 文件系统的大小。XFS 文件系统可以向保存当前文件系统的设备上的未分配空间延伸。这个特性常与卷管理功能结合使用，因为后者可以把多个设备合并进一个逻辑卷组，而使用硬盘分区保存 XFS 文件系统时，每个分区需要分别扩容。

与 LVM 相比，XFS 文件系统无法被收缩。到 2010 年 8 月为止，XFS 分区不可以原位收缩，不过有一些方法可以变相处理这个问题。

26.3.16 Backup/Restore

XFS 提供了 xfsdump 和 xfsrestore 工具协助备份 XFS 文件系统中的数据。xfsdump 按 inode 顺序备份一个 XFS 文件系统。与传统的 UNIX 文件系统不同，XFS 不需要在 dump 前被卸载；对使用中的 XFS 文

件系统做 dump 就可以保证镜像的一致性。这与 XFS 对快照的实现不同，XFS 的 dump 和 restore 的过程是可以被中断然后继续的，无须冻结文件系统。xfsdump 甚至提供了高性能的多线程备份操作——它把一次 dump 拆分成多个数据流，每个数据流可以被发往不同的目的地。不过到目前为止，Linux 尚未完成对多数据流 dump 功能的完整移植。

26.3.17 Atomic Disk Quotas

XFS 的磁盘配额在文件系统被初次挂载时启用。这解决了一个在其它大多数文件系统中存在的一个竞争问题：要求先挂载文件系统，但直到调用 quotaon(8) 之前配额不会生效。

26.3.18 Write Barriers

XFS 文件系统默认在挂载时启用“写入屏障”的支持。该特性会一个合适的时间冲刷下级存储设备的写回缓存，特别是在 XFS 做日志写入操作的时候。这个特性的初衷是保证文件系统的一致性，具体实现却因设备而异——不是所有的下级硬件都支持缓存冲刷请求。在带有电池供电缓存的硬件 RAID 控制器提供的逻辑设备上部署 XFS 文件系统时，这项特性可能导致明显的性能退化，因为文件系统的代码无法得知这种缓存是非易失性的。如果该控制器又实现了冲刷请求，数据将被不必要地频繁写入物理磁盘。为了防止这种问题，对于能够在断电或发生其它主机故障时保护缓存中数据的设备，应该以 nobarrier 选项挂载 XFS 文件系统。

26.3.19 Journal Placement

XFS 文件系统创建时默认使用内置日志，把日志和文件系统数据放置在同一个块设备上。由于在所有的文件系统写入发生前都要更新日志中的元数据，内置日志可能导致磁盘竞争。在大多数负载下，这种等级的竞争非常低以至于对性能没有影响。但对于沉重的随机写入负载，比如在忙碌的数据块服务器上，XFS 可能因为这种 I/O 竞争无法获得最佳性能。另一个可能提高这个问题的严重性的因素是，日志写入被要求以同步方式提交——它们必须被完全写入，之后对应实际数据的写入操作才能开始。

如果确实需要最佳的文件系统性能，XFS 提供了一个选项，允许把日志放置在一个分离的物理设备上。这只需要很小的物理空间。分离的设备有自己的 I/O 路径，如果该设备能对同步写入提供低延迟的路径，那么它将给整个文件系统的操作带来显著的性能提升。SSD 或带有写回缓存的 RAID 系统是日志设备的合适候选，它们能满足这种性能要求。不过后者在遭遇断电时可能降低数据的安全性。要启用外部日志，只须以 logdev 选项挂载文件系统，并指定一个合适的日志设备即可。

第 27 章

Extended file system

The extended file system, or ext[5], was implemented in April 1992 as the first file system created specifically for the Linux kernel. It has metadata structure inspired by the traditional Unix File System (UFS) and was designed by Rémy Card to overcome certain limitations of the MINIX file system. It was the first implementation that used the virtual file system (VFS), for which support was added in the Linux kernel in version 0.96c, and it could handle file systems up to 2 gigabytes (GB) in size.

ext was the first in the series of extended file systems. It was immediately superseded by both ext2 and xiafs, which competed for a time, but ext2 won because of its long-term viability: ext2 remedied issues with ext, such as the immutability of inodes and fragmentation.

There are other members in the extended file system family:

- ext2, the second extended file system
- ext3, the third extended file system.
- ext4, the fourth extended file system.

磁盘分区完毕也就是在指定了硬盘分区所在的起始与结束柱面之后，接下来就是需要将分区格式化为“操作系统能识别的文件系统”，不同操作系统所设置的文件属性/权限并不相同，为了存放这些文件所需的数据就需要将分区进行格式化，之后操作系统才能使用这些分区。

每个操作系统可以识别的文件系统并不相同，例如 Windows 操作系统在默认状态下就无法识别 Linux 的文件系统(这里指 Linux 的标准文件系统 ext2)，所以要针对操作系统来格式化分区。

可以说，每一个分区就是一个文件系统。传统的磁盘与文件系统的应用中，理论上一个分区是不可以具有两个文件系统的，每个文件系统都有其独特的支持方式，例如 Linux 的 ext3 就无法被 Windows 系统所读取。不过由于新技术的发展，比如 LVM 与 Software RAID，这些技术可以将一个分区格式化为多个文件系统(例如 LVM)，也能够将多个分区合成一个文件系统(LVM，RAID)，所以目前我们在格式化时已经不再说成针对分区来格式化了，通常是称呼一个可被挂载的数据为一个文件系统而不是一个分区。

不论是哪一种文件系统，数据总是需要存储的，既然硬盘是用来存储数据的，数据就必须写入硬盘，硬盘的最小存储单位是 Sector，不过数据所存储的最小单位并不是 Sector，因为用 Sector 来存储效率太低。一个 Sector 只有 512 Bytes，而磁头是一个一个 Sector 的读取，也就是说，假设文件有 10M Bytes，那么为了读这个文件，磁头必须要进行读取(I/O)20480 次。

为了克服这个效率上的不足，所以就有逻辑区块(Block)的产生了。逻辑区块是在分区后进行文件系统的格式化时所指定的“最小存储单位”，这个最小存储单位当然是建立在 Sector 的大小上(因为 Sector 为硬盘的最小物理存储单位)，所以 Block 的大小为 Sector 的 2 的次方倍数。

完成文件系统格式化后，磁头一次就可以读取一个 block，假设格式化时指定 Block 为 4K Bytes(也就是由连续的 8 个 Sector 所构成一个 block)，那么同样一个 10M Bytes 的文件，磁头要读取的次数则大幅降为 2560 次，这样就大大的增加文件的读取效率。

不过，Block 单位的规划并不是越大越好，因为一个 Block 最多仅能容纳一个文件(这里指 Linux 的 ext2 文件系统)。举例来说，假如 Block 规划为 4K Bytes，而现在有一个文件大小为 0.1K Bytes，这个小文件将占用掉一个 Block 的空间，也就是说该 Block 虽然可以容纳 4K Bytes 的容量，然而由于文件只占用了 0.1K Bytes，所以实际上剩下的 3.9K Bytes 是不能再被使用了，所以在考虑 Block 的规划时，需要同时考虑到：

- 文件读取的效率;
- 文件大小可能造成的硬盘空间浪费。

因此在规划硬盘时根据主机的用途来进行规划较好,比如 BBS 主机由于文章较短,也就是说文件较小,那么 Block 小一点的好;如果主机主要用在存储大容量的文件,考虑到效率当然 Block 理论上规划的大一点会比较妥当。

文件系统的运行与操作系统的文件数据有关。较新的操作系统的文件数据除了文件实际内容外,通常还含有非常多的属性,例如 Linux 操作系统的文件权限(r、w、x)与文件属性(所有者、用户组、时间参数等)。文件系统通常会将者两部分的数据分别存放在不同的块,权限与属性存放到 inode 中,实际数据则存放到 data block 块中。另外,还有一个超级块(super block)会记录整个文件系统的整体信息,包括 inode 与 block 的总量、使用量、剩余量等。

每个 inode 与 block 都有编号,inode、block、super block 的意义可以简略说明如下:

- super block:记录此文件系统的整体信息,包括 inode/block 的总量、使用量、剩余量以及文件系统的格式与相关信息等;
- inode:记录文件的属性,1 个文件占用 1 个 inode,同时记录此文件的数据所在的 block 号码;
- block:实际记录文件的内容,若文件太大,则会占用多个 block。

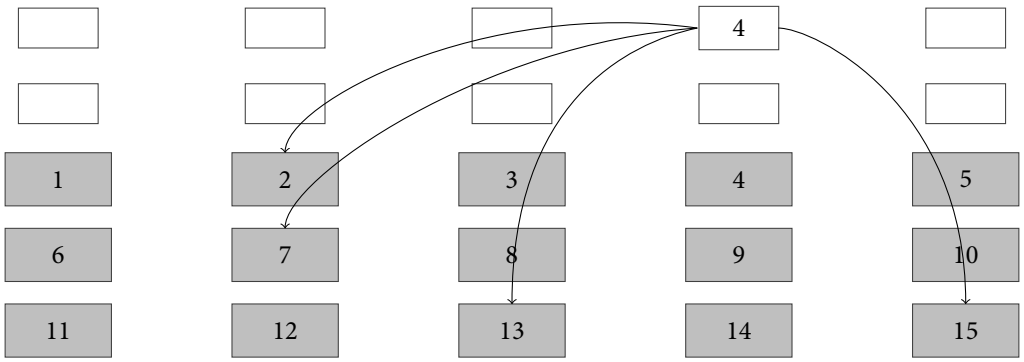
由于每个 inode 与 block 都有编号,而每个文件都会占用一个 inode,而 inode 内存有文件数据放置的 block 号码。找到文件的 inode 自然就会知道这个文件所放置数据的 block 号码,当然也就能够读出该文件的实际数据了,这样读写磁盘时通过 inode 和 block 就能在短时间内读取出全部的数据,提高了读写性能。

27.1 super block

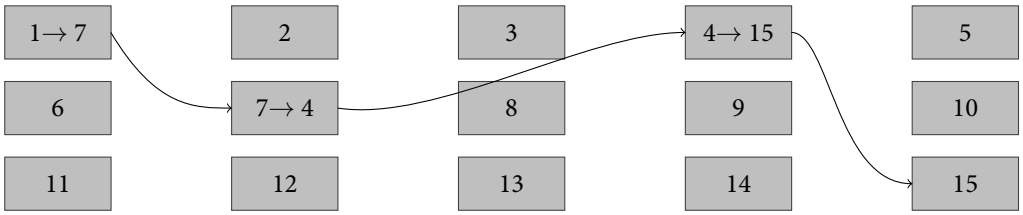
在进行硬盘分区时,每个硬盘分区就是一个文件系统,而每个文件系统开始位置的那个 block 就称为 super block。super block 用于存储文件系统的大小、空的和填满的区块,以及它们各自的总数和其他的信息等,也就是说,当要使用这一个硬盘分区(或者说是文件系统)来进行数据存取的时候,第一个要经过的就是 super block 这个区块,如果 super block 损坏,这个硬盘分区大概也就回天乏术。

27.2 inode, block

下面使用示意图来说明 inode 与 block,文件系统先格式化出 inode 与 block 的块,假设某一个文件的属性与权限数据是放置到 inode 4 号,而这个 inode 记录了文件数据的实际存放点为 2、7、13、15 这 4 个 block 号码,于是操作系统就能够根据这些来排列磁盘的阅读顺序,从而可以很快将 4 个 block 内容读出来,于是数据的读取就如同下图中的箭头所指定的。



这种数据访问的方法称为索引式文件系统(indexed allocation)。而 U 盘等所使用的 FAT 文件格式并没有 inode 存在,因此在 FAT 文件系统中无法将文件的所有 block 在起始时就读取出来,每个 block 号码都记录在前一个 block 中,它的读取方式如下图所示:



在 FAT 文件系统中需要一个接一个地将 block 读出后才会知道下一个 block 的位置。如果同一个文件数据写入的 block 离散度很大，磁盘磁头将无法在磁盘转一圈就读取到所有的数据，此时磁盘就需要多转几圈才能完整地读取到这个文件的内容。

操作系统需要碎片整理的原因就是文件写入的 block 太过于离散了，此时文件读取的性能将会变得很差，通过碎片整理就可以把同一个文件所属的 block 汇合在一起，这样数据的读取会比较容易。

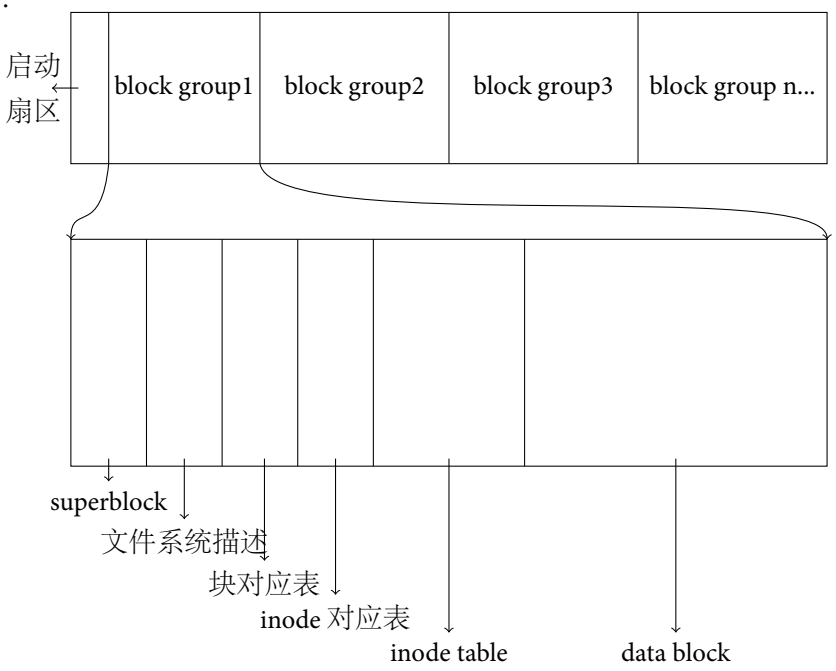
Linux 系统所使用的 ext2/3/4 文件系统是索引式文件系统，基本上不太需要经常进行碎片整理，但是如果文件系统使用太久，或经常删除/编辑/新增文件时就有可能会有文件数据过于离散的问题，就需要进行碎片整理。

27.3 EXT2

这里以 Linux 最标准的 ext2 这个文件系统来作为说明，Linux 系统中每个文件除了文件的内容数据还包括文件的各种权限与属性，例如所属群组、所属使用者、能否执行、文件建立时间、文件特殊属性等。由于 Linux 操作系统是一个多用户多任务的环境，为了要保护每个使用者所拥有数据的隐密性，所以具有多样化的文件属性是难免的。

标准的 ext2 文件系统就是使用 inode 为基础的文件系统，其中会将每个文件的内容分为两个部分来存储，一个是文件的属性，另一个则是文件的内容。为了满足这两个不同的需求，ext2 规划出 inode 与 block 来分别存储文件的属性（放在 inode 当中）与文件的内容（放置在 block area 当中），而且文件系统一开始就将 inode 与 block 规划好了，除非重新格式化（或者利用 resize2fs 等命令更改文件系统大小），否则 inode 与 block 固定后就不再变动。

当要将一个分区格式化为 ext2 时，就必须要指定 inode 与 block 的大小才行，也就是说，当分区被格式化为 ext2 文件系统时，一定会有 inode table 与 block area 这两个区域，因此 ext2 文件系统在格式化时基本上是区分为多个块组（block group）的，每个块组都有独立的 inode/block/superblock 系统，如下图所示：



在整体的规划中，文件系统最前面有一个启动扇区(boot sector)，这个启动扇区可以安装引导加载程序，这是个非常重要的设计，这样用户就能够将不同的引导加载程序安装到其他的文件系统的最前端而不同覆盖整块硬盘唯一的 MBR，这样才能够构造出多重引导的环境。

27.3.1 data block

data block 是用来放置文件内容的地方，在 ext2 文件系统所支持的 block 大小有 1 KB，2 KB 及 4 KB 三种。在格式化文件系统时就已经固定好了，并且每个 block 都有编号以方便 inode 的记录。不过要注意的是，由于 block 的大小的区别会导致该文件系统能够支持的最大磁盘容量与最大单一文件容量不同，由 block 大小而产生的 ext2 文件系统的限制如下表所示：

block 大小	1 KB	2 KB	4 KB
最大单一文件限制	16GB	256GB	2TB
最大文件系统总容量	2TB	8TB	16TB

需要注意的是，虽然 ext2 已经能够支持大于 2GB 以上的单一文件容量，不过某些应用程序还是使用旧的限制，也就是说，这些程序只能够支持 2GB 的文件，这就跟文件系统无关了。除此之外，ext2 文件系统的 block 的基本限制如下：

- 原则上，block 的大小与数量在格式化完成后就不能够再改变了(除非重新格式化)；
- 每个 block 内最多只能放置一个文件的数据；
- 承上，如果文件大于 block 的大小，则一个文件会占用多个 block；
- 承上，如果文件小于 block 的大小，则该 block 的剩余空间就不能够再被使用了。

所以，由于每个 block 仅能容纳一个文件的数据，因此如果要存储的文件都非常小，而此时 block 在格式化时选用的是最大的 4 KB 时可能就会产生一些空间的浪费。

既然大的 block 可能会产生较严重的磁盘容量浪费，但是将 block 定为 1 KB 也不妥。因为如果 block 较小，那么大型文件将会占用数量更多的 block，而 inode 也要记录更多的 block 号码，此时将可能导致文件系统读写性能下降。

在进行文件系统的格式化之前，就要先想好该文件系统预计使用的情况。

27.3.2 inode table

block 是数据存储的最小单位或者说 block 是记录“文件内容数据”的区域，inode 则是记录“该文件的相关属性以及文件内容放置在哪一个 block 之内”的信息。简单的说，inode 除了记录文件的属性外，同时还必须要具有指向(pointer)的功能，也就是指向文件内容放置的区块之中，这样操作系统才可以正确的去取得文件的内容。下面是 inode 记录的文件数据信息(当然不止这些)：

- 该文件的拥有者与群组(owner/group)；
- 该文件的存取模式(read/write/excute)；
- 该文件的类型(type)；
- 该文件创建或状态改变的时间(ctime)、最近一次的读取时间(atime)、最近修改的时间(mtime)；
- 该文件的大小；
- 定义文件特性的标志(flag)，如 SetUID 等；
- 该文件真正内容的指向(pointer)；

利用 ls 查询文件所记录的时间，就是 atime/ctime/mtime 三种时间，这三种时间就是记录在 inode 里面的，默认的显示时间是 mtime。

```
[root@linux ~]# ls -la --time=atime PATH
```

那个 PATH 是所想要查询的文件或目录名称，利用上面的 ls 相关参数就可以取得想要知道的文件相关的三种时间。至于一个 inode 的大小为 128 bytes 这么大，可以使用 dumpe2fs 来查看 inode 的大小。

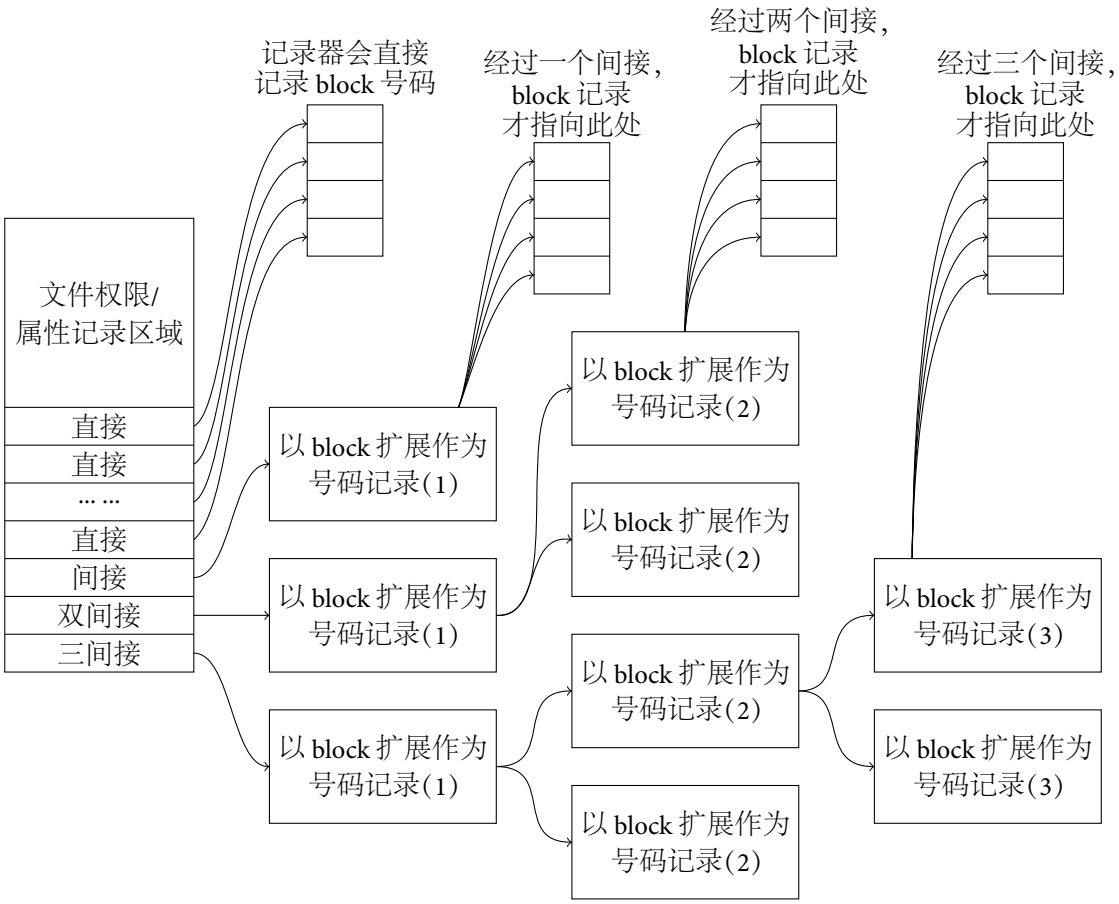
inode 的数量与大小也是在格式化时就已经固定了，除此之外，inode 还有下面的特性：

- 每个 inode 大小均固定为 128 Bytes；
- 每个文件都仅会占用一个 inode 而已；
- 承上，文件系统能够创建的文件数量与 inode 的数量有关；
- 系统读取文件时需要先找到 inode，并分析 inode 所记录的权限与用户是否符合，若符合才能开始实际读取 block 的内容。

下面简要分析 inode/block 与文件大小的关系：

inode 要记录的数据非常多，但其本身只有 128Bytes，而 inode 要记录一个 block 号码要花掉 4Bytes，此时假设某文件有 40 MB 且每个 block 为 4 KB 时，那么就至少要 10 万条 block 号码的记录，这很不符合现实。

现代操作系统采用的是将 inode 记录 block 号码的区域定义为 12 个直接、1 个间接、1 个双间接与 1 个三间接记录区，如下图所示：



图中最左边是 inode 本身(128bytes)，里面有 12 个直接指向 block 号码的对照，这 12 个记录就能够直接取得 block 号码，而间接就是再拿一个 block 来作为记录 block 号码的记录区，如果文件太大时就会使用间接的 block 来记录编号。如果文件持续增大，那么就会使用“双间接”，第一个 block 仅指出下一个记录编号的 block 的位置，实际记录的在第二个 block 中。以此类推，三间接就是利用第三层的 block 来记录编号。

下面以较小的 1KB 的 block 来说明 inode 能够指定的 block 的数目。

(1)12 个直接指向：12×1K=12K；

由于是直接指向，所以总共可以记录 12 条记录。

(2)间接：256×1K=256K；

每条 block 号码的记录会用去 4bytes，因此 1K 的大小能够记录 256 条记录。

(3)双间接：$256\times256\times1K=256^2K$；

第一层 block 会指定 256 个第二层，每个第二层可以指定 256 个号码。

(4)三间接：$256\times256\times256\times1K=256^3K$；

第一层 block 会指定 256 个第二层，每个第二层可以指定 256 个第三层，每个第三层可以指定 256 个号码。

(5)总额：将直接、间接、双间接加总，得到 12+256+256×256+256×256×256(K)=16GB

此时可知，当文件系统将 block 格式化为 1K 大小时，能够容纳的最大文件为 16GB，比较上面的文件系统限制表的结果可看出前后一致。但这个方法不能用在 2K 及 4K 的 block 大小的计算中，因为大于 2K 的 block 将会受到 ext2 文件系统本身的限制，所以计算结果会不太符合。

27.3.3 superblock

superblock 是记录整个文件系统相关信息的地方，没有 superblock 也就没有这个文件系统了，superblock 记录的信息主要有：

- block 与 inode 的总量;
- 未使用与已使用的 inode/block 数量;
- 一个 block 与一个 inode 的大小;
- filesystem 的挂载时间、最近一次写入数据的时间、最近一次检验磁盘(fsck)的时间等文件系统的相关信息;
- 一个 valid bit 数值,若此文件系统已被挂载,则 valid bit 为 0 ,若未被挂载,则 valid bit 为 1。

如果 superblock 损坏,那么文件系统可能就需要花费很多时间来恢复。一般来说,superblock 的大小为 1024bytes,相关的 superblock 信息可以使用 dumpe2fs 命令来查看。

此外,每个 block group 都可能含有 superblock,但是实际上每个文件系统应该仅有 1 个 superblock。事实上除了第一个 block group 内会含有 superblock 之外,后续的 block group 不一定会含有 superblock,若含有 superblock 则该 superblock 主要是作为第一个 block group 内的 superblock 的备份,从而可以用于 superblock 的恢复。

27.3.4 file system discription

file system discription(文件系统描述)区段可以描述每个 block group 的开始与结束的 block 号码,以及说明每个区段(superblock,bitmap,inodemap,data block)分别位于哪一个 block 号码之间,这些信息也可以使用 dumpe2fs 来查看。

27.3.5 block bitmap

在添加文件时就需要用到 block,通过 block bitmap(块对照表)的辅助就可以找到空的 block 来放置文件。同样的,当删除文件时原本被占用的 block 就会被释放出来,此时在 block bitmap 中对应到该 block 号码的标志就得要修改为“未使用”,上述这就是 block bitmap 的功能。

27.3.6 inode bitmap

inode bitmap(inode 对照表)与 block bitmap 功能类似,只是 block bitmap 记录的是使用的与未使用的 block 号码,而 inode bitmap 则是用来记录使用的与未使用的 inode 号码。

文件系统的各个块组的数据都与 block 号码有关, 每个区段与 superblock 的信息都可以使用 dumpe2fs 命令来查询,查询的方法与结果如下:

```
[root@linux ~]# dumpe2fs [-bh] 设备文件名
参数:
-b:列出保留为坏道的部分;
-h:仅列出 superblock 的数据,不会列出其他的区段内容。

[root@linux ~]# dumpe2fs /dev/hda1
File system volume name:           /
File system state:                clean
Errors behavior:                Continue
File system OS type:                Linux
Inode count:                1537088
Block count:                1536207
Free blocks:                735609
Free inodes:                1393089
First block:                0
Block size:                        4096
File system created:                Sat Jun 25 16:21:13 2005
Last mount time:                Sat Jul 16 23:45:04 2005
Last write time:                Sat Jul 16 23:45:04 2005
Last checked:                 Sat Jun 25 16:21:13 2005
First inode:                11
```

```
Inode size:                    128
Journal inode:                   8

Group 0: (Blocks 0-32767)
  Primary superblock at 0, Group descriptors at 1-1
  Reserved GDT blocks at 2-376
  Block bitmap at 377 (+377), Inode bitmap at 378 (+378)
  Inode table at 379-1400 (+379)
  0 free blocks, 32424 free inodes, 11 directories
  Free blocks:
  Free inodes: 281-32704
Group 1: (Blocks 32768-65535)
  Backup superblock at 32768, Group descriptors at 32769-32769
  Reserved GDT blocks at 32770-33144
  Block bitmap at 33145 (+377), Inode bitmap at 33146 (+378)
  Inode table at 33147-34168 (+379)
  18 free blocks, 24394 free inodes, 349 directories
  Free blocks: 37882-37886, 38263-38275
  Free inodes: 38084-38147, 39283-39343, 41135, 41141-65408
# 因为数据很多,这里略去了一些信息,上面是比较精简的显示内容。
# 在 Group 0 之前的都是 Superblock 的内容,记录了 inode/block 的总数,
# 还有其他相关的信息。至于由 Group 0 之后,则是说明各个 bitmap 及 inode table
# 与 block area 等。
```

通过 dumpe2fs 查询到的信息依内容可以区分为两部分，上半部分是 superblock 的内容，而下半部分则是每个 blockgroup 的信息。

通过这些记录可以知道哪些 inode 没有被使用，哪些 block 还可以记录，这样在新增、建立文件与目录时，系统就会根据这些记录来将数据分别写入尚未被使用的 inode 与 block area。

不过，要注意的是，当新增一个文件(目录)时：

- 根据 inode bitmap/block bitmap 的信息，找到尚未被使用的 inode 与 block，进而将文件的属性与数据分别记录进 inode 与 block；
- 将刚刚被利用的 inode 与 block 的号码(number)告知 superblock、inode bitmap、block bitmap 等，让这些 metadata 更新信息。

一般来说，将 inode table 与 block area 称为数据存储区域，而其他的例如 superblock、block bitmap 与 inode bitmap 等记录就被称为 metadata。经由上面两个动作就可以知道一次数据写入硬盘时会有这两个动作。

那么 Linux 系统到底是如何读取一个文件的内容呢？下面分别针对目录与文件来说明：

1、目录：

在 Linux 下的 ext2 文件系统建立一个目录时，ext2 会分配一个 inode 与至少一块 Block 给该目录。其中 inode 记录该目录的相关属性，并指向分配到的那块 Block，而 Block 则是记录在这个目录下的相关连的文件(或目录)的关连性。

2、文件：

当在 Linux 下的 ext2 建立一个一般文件时，ext2 会分配至少一个 inode 与相对于该文件大小的 Block 数量给该文件。例如假设一个 Block 为 4 Kbytes，而要建立一个 100 KBytes 的文件，那么 Linux 将分配一个 inode 与 25 个 Block 来存储该文件。

要注意的是，inode 本身并不记录文件名，而是记录文件的相关属性，至于文件名则是记录在目录所属的 block 区域，那么文件与目录的关系又是如何呢？

就如同上面的目录提到的，文件的相关连接会记录在目录的 block 数据区域，所以当要读取一个文件的内容时，Linux 会先由根目录/取得该文件的上层目录所在 inode，再由该目录所记录的文件关连性(在该目录所属的 block 区域)取得该文件的 inode，最后在经由 inode 内提供的 block 指向取得最终的文件内容。

以 /etc/crontab 这个文件的读取为例，内容数据是这样取得的：

一块 Partition 在 ext2 底下会被格式化为 inode table 与 block area 两个区域，所以在图三里面，我们将 Partition 以长条的方式来示意，会比较容易理解的。

而读取 /etc/crontab 的流程为：

操作系统根据根目录 (/) 的相关数据可取得 /etc 这个目录所在的 inode ，并前往读取 /etc 这个目录的所有相关属性；

根据 /etc 的 inode 的数据，可以取得 /etc 这个目录底下所有文件的关连数据是放置在哪一个 Block 当中，并前往该 block 读取文件的关连性内容；

由上个步骤的 Block 当中，可以知道 crontab 这个文件的 inode 所在地，并前往该 inode ；

由上个步骤的 inode 当中，可以取得 crontab 这个文件的所有属性，并且可前往由 inode 所指向的 Block 区域，顺利的取得 crontab 的文件内容。

整个读取的流程大致上就是这样。

如果想要实作一下以了解整个流程的话，可以这样试做看看：

1. 查看一下根目录所记录的所有文件关连性数据

```
[root@linux ~]# ls -lia /
      2 drwxr-xr-x  24 root root  4096 Jul 16 23:45 .
      2 drwxr-xr-x  24 root root  4096 Jul 16 23:45 ..
 719489 drwxr-xr-x  83 root root 12288 Jul 21 04:02 etc
 523265 drwxr-xr-x  24 root root  4096 Jun 25 20:16 var
# 注意看一下，在上面的 . 与 .. 都是连结到 inode 号码为 2 的那个 inode ，
# 也就是说， / 与其上层目录 .. 都是指向同一个 inode number，两者是相同的。
# 而在根目录所记录的文件关连性 (在 block 内) 得到/etc 的 inode number
# 为 719489 那个 inode number。
```

2. 查看一下 /etc/ 内的文件关连性的数据

```
[root@linux ~]# ls -liad /etc/crontab /etc/.
719489 drwxr-xr-x  83 root root 12288 Jul 21 04:02 /etc/.
723496 -rw-r--r--   1 root root   663 Jul  4 12:03 /etc/crontab
# 此时就能够将/etc/crontab 找到关连性。
```

目录的最大功能就是在提供文件的关连性，在关连性里面，当然最主要的就是“文件名与 inode 的对应数据”。

另外，关于 ext2 文件系统，这里有几点要提醒一下：

1、ext2 与 ext3 文件在建立时(format)就已经设置好固定的 inode 数与 block 数目；

2、格式化 Linux 的 ext2 文件系统，可以使用 mke2fs 这个程序来执行；

3、ext2 允许的 block size 为 1024、2048 及 4096 bytes；

4、一个 Partition(File system) 所能容许的最大文件数，与 inode 的数量有关，因为一个文件至少要占用一个 inode。

5、在目录底下的文件数如果太多而导致一个 Block 无法容纳的下所有的关连性数据时，Linux 会给予该目录多一个 Block 来继续记录关连数据；

6、通常 inode 数量的多少设置为(Partition 的容量)除以(一个 inode 预计想要控制的容量)。举例来说，若 block 规划为 4K bytes，假设一个 inode 会控制两个 block，也就是是假设一个文件大致的容量在 8Kbytes 左右时，假设这个 Partition 容量为 1GBytes，则 inode 数量共有：(1G * 1024M/G * 1024K/M) / (8K) = 131072 个。

而一个 inode 占用 128 bytes 的空间，因此格式化时就会有 131072 个 * 128bytes/个) = 16777216 byes = 16384 Kbytes 的 inode table 。也就是说，这一个 1GB 的 Partition 在还没有存储任何数据前，就已经少了 16MBytes 的容量。

因为一个 inode 只能记录一个文件的属性，所以 inode 数量比 block 多是没有意义的！举上面的例子来说，Block 规划为 4 Kbytes，所以 1GB 大概就有 262144 个 4Kbytes 的 block ，如果一个 block 对应一个 inode 的话，那么当 inode 数量大于 262144 时，多的 inode 将没有任何用处，只是浪费硬盘的空间而已。

另外一层想法，如果文件容量都很大，那么一个文件占用一个 inode 以及数个 block，当然 inode 数量就可以规划的少很多。

当 block 大小越小，而 inode 数量越多，则可利用的空间越多，但是大文件写入的效率较差；这种情况适合文件数量多，但是文件容量小的系统，例如 BBS 或者是新闻组（News Group）这方面服务的系统。

当 Block 大小越大，而 inode 数量越少时，大文件写入的效率较佳，但是可能浪费的硬盘空间较多；这种状况则比较适合文件容量较大的系统。

简单的归纳一下，ext2 有几个特色：

1、Blocks 与 inodes 在一开始格式化时（format）就已经固定了；

2、一个 Partition 能够容纳的文件数与 inode 有关；

3、一般来说，每 4Kbytes 的硬盘空间分配一个 inode；

4、一个 inode 的大小为 128 bytes；

5、Block 为固定大小，目前支持 1024/2048/4096 bytes 等；

6、Block 越大，则消耗的硬盘空间也越多。

关于单一文件：

若 block size=1024，最大容量为 16GB，若 block size=4096，容量最大为 2TB；

关于整个 Partition：

若 block size=1024，则容量达 2TB，若 block size=4096，则容量达 32TB。文件名最长达 255 字符，完整文件名长达 4096 字符。

另外，Partition 的使用效率上，当一个 Partition 规划的很大时，例如 100GB，由于硬盘上面的数据总是来来去去的，所以整个 Partition 上的文件通常无法连续写在一起，而是填入式的将数据填入没有被使用的 block 当中。如果文件写入的 block 真的分的很分散，此时就会有所谓的文件离散的问题发生了。虽然 ext2 在 inode 处已经将该文件所记录的 block number 都记上了，所以数据可以一次性读取，但是如果文件真的太过离散，确实还是会发生读取效率下降的问题，可以将整个 Partition 内的数据全部复制出来，将该 Partition 重新格式化，再将数据复制回去即可解决。

此外，如果 Partition 真的太大了，那么当一个文件分别记录在这个 Partition 的最前面与最后面的 block，此时会造成硬盘的机械臂移动幅度过大，也会造成数据读取效率的下降。因此 Partition 的规划并不是越大越好，而是真的要针对主机用途来进行规划才行。

Linux 最优秀的地方之一，就在于它的多用户多任务的环境。为了让各个用户具有较安全的文件管理机制，因此文件的权限管理就变的很重要了。

Linux 一般将文件可存取访问的身份分为三个类别，分别是 owner/group/other，且各自具有 read/write/execute 等权限，若管理得当将会让 Linux 主机变的较为安全。

27.4 EXT3

27.5 EXT4

第 28 章

HFS+

HFS Plus(Hierarchical File System Plus,或 HFS+)是苹果公司为替代他们的分层文件系统(HFS)而开发的一种文件系统。它被用在 Macintosh 电脑(或者其他运行 Mac OS 的电脑)上。在开发过程中,苹果公司也把这个文件系统的代号命名为“Sequoia”。

HFS+ 也是 iPod 上使用的其中一种格式,同时它也被称为 Mac OS Extended (或误称为“HFS Extended”)。

HFS+ 作为 HFS 的改进版本,支持更大的文件,并用 Unicode 来命名文件或文件夹来代替了 Mac OS Roman 或其他一些字符集。和 HFS 一样,HFS+ 也使用 B 树来存储大部分分卷元数据。

Linux 内核包含了 hfsprogs 模块供系统挂载。HFS+ 的 fsck 与 mkfs 工具程序也被移植到 Linux 工具中。在 Windows 上有商业软件 MacDrive 和 Paragon HFS for Windows 可以让使用者格式化并读写 HFS+ 文件系统,同时从 Mac OS X 10.6 开始 Boot Camp 所随附的驱动程序也可以使 Windows 能够读取 HFS+ 文件系统的分区。

第 29 章

ZFS

ZFS(Zettabyte File System)源自于 Sun Microsystems 为 Solaris 操作系统开发的文件系统。ZFS 是一个具有高存储容量、文件系统与卷管理概念集成、崭新的磁盘逻辑结构的轻量级文件系统,同时也是一个便捷的存储池管理系统。

Lustre 是一种通常使用在大型计算机集群与超级计算机中的平行分布式文件系统,最早由彼得·布拉姆(Peter Braam)创建的集群文件系统公司(Cluster File Systems Inc.)在 1999 年开始研发,于 2003 年采用 GNU GPLv2 开源码授权发布 Lustre 1.0。2007 年,Sun 并购集群文件系统公司。

现在,ZFS 是一个使用通用开发与发布许可证授权的开源项目。Linux 通过用户空间文件系统或原生第三方内核可加载核心模组支持 ZFS。美国劳伦斯利福摩尔国家实验室(LLNL)的 ZFS on Linux 开源计划于 2013 年 3 月发布了一个 ZFS 的 Linux 原生移植。

29.1 Storage Pools

不同于传统文件系统需要驻留于单独设备或者需要一个卷管理系统去使用一个以上的设备,ZFS 创建在虚拟的,被称为“zpools”的存储池之上(存储池最早在 AdvFS 实现,并且加到后来的 Btrfs)。每个存储池由若干虚拟设备(virtual devices,vdevs)组成。这些虚拟设备可以是原始磁盘,也可能是一个 RAID1 镜像设备,或是非标准 RAID 等级的多磁盘组,于是 zpool 上的文件系统可以使用这些虚拟设备的总存储容量。

可以使用磁盘限额以及设置磁盘预留空间来限制存储池中单个文件系统所占用的空间。

29.2 Capacity

ZFS 是一个 128 位的文件系统,这意味着它能存储 1800 亿亿(18.4×10^{18})倍于当前 64 位文件系统的数据。ZFS 的设计如此超前以至于这个极限就当前现实实际可能永远无法遇到。

项目领导 Bonwick 曾说:“要填满一个 128 位的文件系统,将耗尽地球上所有存储设备。除非你拥有煮沸整个海洋的能量,不然你不可能将其填满。(Populating 128-bit file systems would exceed the quantum limits of earth-based storage. You couldn't fill a 128-bit storage pool without boiling the oceans.)”

以下是 ZFS 的一些理论极限:

- 2^{48}——任意文件系统的快照数量(2×10^{14})
- 2^{48}——任何单独文件系统的文件数(2×10^{14})
- 16 exabytes (2^{64} byte)——文件系统最大尺寸
- 16 exabytes (2^{64} byte)——最大单个文件尺寸
- 16 exabytes (2^{64} byte)——最大属性大小
- 128 Zettabytes (2^{78} byte)——最大 zpool 大小
- 2^{56}——单个文件的属性数量(受 ZFS 文件数量的约束,实际为 2^{48})
- 2^{56}——单个目录的文件数(受 ZFS 文件数量的约束,实际为 2^{48})
- 2^{64}——单一 zpool 的设备数
- 2^{64}——系统的 zpools 数量

- 2^{64}——单一 zpool 的文件系统数量

作为对这些数字的感性认识，假设每秒钟创建 1,000 个新文件，达到 ZFS 文件数极限需要大约 9,000 年。

在辩解填满 ZFS 与煮沸海洋的关系时，Bonwick 写到：

> 尽管我们都希望摩尔定律永远延续，但是量子力学给定了任何物理设备上计算速率（computation rate）与信息量的理论极限。举例而言，一个质量为 1 公斤，体积为 1 升的物体，每秒至多在 10^{31} 位信息上进行 10^{51} 次运算。一个完全的 128 位存储池将包含 2^{128} 个块 = 2^{137} 字节 = 2^{140} 位；应此，保存这些数据位至少需要（2^{140} 位）/（10^{31} 位/公斤）= 1360 亿公斤的物质。

29.3 Copy-on-write Transactional Model

ZFS 使用一种写时拷贝事务模型技术。所有文件系统中的块指针都包括 256 位的能在读时被重新校验的关于目标块的校验和。含有活动数据的块从来不被覆盖；而是分配一个新块，并把修改过的数据写在新块上。所有与该块相关的元数据块都被重新读、分配和重写。为了减少该过程的开销，多次读写更新被归纳为一个事件组，并且在必要的时候使用日志来同步写操作。

利用写时拷贝使 ZFS 的快照和事物功能的实现变得更简单和自然，快照功能更灵活。缺点是，COW 使碎片化问题更加严重，对于顺序写生成的大文件，如果以后随机的对其中的一部分进行了更改，那么这个文件在硬盘上的物理地址就变得不再连续，未来的顺序读会变得性能比较差。

29.4 Snapshots and Clones

ZFS 使用写时拷贝技术的一个优势在于，写新数据时，包含旧数据的块被保留着，提供了一个可以被保留的文件系统的快照版本。由于 ZFS 在读写操作中已经存储了所有构建快照的数据，所以快照的创建非常快。而且由于任何文件的修改都是在文件系统和它的快照之间共享的，所以 ZFS 的快照也是空间优化的。

可写快照（“克隆”）也可以被创建。结果就是两个独立的文件系统共享一些列的块。当任何一个克隆版本的文件系统被改变时，新的数据块为了反映这些改变而创建，但是不管有多少克隆版本的存在，未改变的块仍然在其他的克隆版本中共享。

29.5 Dynamic Striping

ZFS 能动态条带化所有设备以最大化吞吐量。当额外的设备被加入到 zpool 中的时候，条带宽度会自动扩展以包含这些设备。这使得存储池中的所有磁盘都被用到，同时负载被平摊到所有的磁盘上。

29.6 Variable Block Sizes

ZFS 使用可变大小的块，最大可至 128KB。现有的代码允许管理员调整最大块大小，这在大块效果不好的时候有用。未来也许能做到自动调整适合工作量的块大小。

ZFS 的可变大小的块与 BtrFS 和 Ext4 的 extent 不同。在 ZFS 中，在一个文件中所有数据块的逻辑长度必须是相同的，不同文件之间的块大小可以不同，因此 ZFS 可以用直接映射（direct map）的方式（同 ufs/ffs/ext2/ext3）来来搜索间接块的数据指针数组（blkptr）。BtrFS 和 Ext4 的 extent 方式在同一个文件中每个数据快的大小都可以不相同，因此需要用 B+ Tree 或者类 B Tree 的方式来组织间接块的数据。

虽然直接映射方式比 B+ Tree 的查找速度快，但是这种方式的缺点也非常明显，如：元数据开销过大、顺序 IO 的大文件性能不好、删除比较慢等等，因此在现代文件系统中映射方式逐渐被 extent 变长块取代。

如果数据压缩(LZJB)被启用，可变块大小需要被用到。如果一个数据块可被压缩至一个更小的数据块，则小的数据块将使用更少的存储和提高吞吐量(代价是增加 CPU 压缩和解压缩的负担)。

29.7 Lightweight Filesystem Creation

在 ZFS 中，存储池中的文件系统的操作相比传统文件系统的卷管理更加便捷。创建 ZFS 文件系统或者改变一个 ZFS 文件系统的大小接近于传统技术中的管理目录而非管理卷。

第 30 章

GFS

Google 文件系统(GFS 或 GoogleFS)是 Google 公司为了满足其需求而开发的基于 Linux 的专有分布式文件系统。尽管 Google 公布了该系统的一些技术细节,但 Google 并没有将该系统的软件部分作为开源软件发布。现在,Hadoop 提供与 Google 文件系统类似的功能。

GFS 专门为 Google 的核心数据即页面搜索的存储进行了优化。数据使用大到若干 G 字节的大文件持续存储,而这些文件极少被删除、覆盖或者减小;通常只是进行添加或读取操作。

GFS 针对 Google 的计算机集群进行了设计和优化,这些节点是由廉价的"常用"计算机组成,这就意味着必须防止单个节点的高损害率和随之带来的数据丢失。

节点分为两类:主节点和 Chunkservers。Chunkservers 存储数据文件,这些单个的文件象常见的文件系统中的簇或者扇区那样被分成固定大小的数据块(这也是名字的由来)。每个数据块有一个唯一的 64 位标签,维护从文件到组成的数据块的逻辑映射。每个数据块在网络上复制一个固定数量的次数,缺省次数是 3 次,对于常用文件如可执行文件的次数要更多。

主服务器通常并不存储实际的大块数据,而是存储与大块数据相关的元数据,这样的数据如映射表格将 64 位标签映射到大块数据位置及其组成的文件、大块数据副本位置、哪个进程正在读写特定的大数据块或者追踪复制大块数据的"快照"(通常在主服务器的激发下,当由于节点失败的时候,一个大数据块的副本数目降到了设定的数目下)。所有这些元数据通过主服务器周期性地接收从每个数据块服务器来的更新("心跳消息")保持最新状态。

操作的允许授权是通过限时的、倒计时"租期"系统来处理的,主服务器授权一个进程在有限的时间段内访问数据块,在这段时间内主服务器不会授权其它任何进程访问数据块。被更改的 chunkserver——总是主要的数据块存储器,然后将更改复制到其它的 chunkserver 上。这些变化直到所有的 chunkserver 确认才存储起来,这样就保证了操作的完整性和原子性。

访问大数据块的程序首先查询主服务器得到所要数据块的位置,如果大数据块没有进行操作(没有重要的租约),主服务器回答大数据块的位置,然后程序就可以直接与 chunkserver 进行联系接收数据(类似于 Kazaa 和它的超级节点)。

第 31 章

GmailFS

Gmail 文件系统 GmailFS 使用一个可加载的 Linux 文件系统，它使用 Gmail 帐号作为存储媒介。

GmailFS 是一个虚拟文件系统，由理查德·琼斯开发，通过用户的 Gmail 电邮帐号来作文件储存。GmailFS 本身是为 Linux 系统而写，并应用了 Linux FUSE 技术，现在 GamilFS 系统也可以移植到 Microsoft Windows 及 Mac OS X 上运行。

GmailFS 的核心是一个以 Python 开发的 libgmail 库，作为 FUSE 与 Gmail 系统沟通的中介，整个 GmailFS 亦是以 Python 来创建。

GmailFS 的速度主要受到用户的互联网连接速度限制，亦视 Gmail 服务器的存取速度而定。理论上，GmailFS 里的文件存放可以是任何大小，但事实上由于受到 Gmail 邮件信箱的大小限制，GmailFS 亦有其上限。文件及目录以邮件及附件的形式储存在 Gmail 里。

第 32 章

用户与用户组

Linux 中的文件都有相当多的属性与权限,其中最重要的可能就是文件的所有者的概念了。

32.1 文件所有者

用户与用户组的功能是 Linux 中相当健全而好用的安全防护手段。由于 Linux 是多用户多任务的系统,可能常常会有多人同时使用主机来进行运算的情形发生,考虑到每个人的隐私以及每个人喜好的运行环境,因此,“文件所有者”的角色是相当重要的。

通过文件的属性可以设定适当的权限,这样就只有具有适当权限的用户才能查看与修改文件。

用户组最简单的功能之一就是在团队开发项目的场景中,举例来说,在当前主机上面有两个团队,第一个团队名称为 testgroup,它的成员有 test1、test2、test3 三个,第二个团队名称为 treatgoup,它的成员为 treat1、treat2、treat3。这两个团队之间是互相有竞争性质的,但是却又要提交同一份报告,然而每组成员又需要同时可以修改自己的团队内任何人所建立的文件且不能让非自己团队的其它人看到自己的文件内容。

这时在 Linux 下就可以通过文件权限设定来限制非自己团队(也就是用户组)的其它人浏览文件内容的权限。而且还可以只能让自己的团队成员可以修改文件,同时如果还有私人的文件,仍然可以设定成让自己的团队成员也看不到这些文件数据。

另外,如果 teacher 这个账号是 testgroup 与 treatgroup 这两个用户组的老师,他想要同时查看二者的进度,因此需要进入这两个用户组的权限,此时就可以设定 teacher 这个账号“同时支持 testgroup 与 treatgroup 这两个用户组”,也就是说,每个账号都可以有多个用户组的支持。

另外也可以使用“家庭”的概念来进行解释。假设有一家人,家里只有三兄弟,分别是王大毛、王二毛与王三毛三个人,而这个家庭是登记在王大毛的名下的,所以“王大毛家有三个人,分别是王大毛、王二毛与王三毛”,而且这三个人都有自己的房间,并且共同拥有一个客厅。由于王家三个人各自拥有自己的房间,所以王二毛虽然可以进入王三毛的房间,但是王二毛不能乱翻王三毛的抽屉。因为抽屉里面可能有王三毛自己私人的东西,这是“私人的空间”。

由于共同拥有客厅,所以王家三兄弟可以在客厅看电视机、看报纸等。反正,只要是在客厅的东西,三兄弟都可以使用,因为大家都是一家人。这里的“王大毛家”就是所谓的“用户组”,至于三兄弟就分别为三个“用户”,而这三个用户是在同一个用户组里面。而三个用户虽然在同一用户组内,但是我们可以设定“权限”,好让某些用户个人的信息不被用户组的拥有者查询以保护用户的“私人的空间”。通过设定用户组共享,则可让大家共同分享文件。

32.2 其它人的概念

比如又有个人叫张小猪,他与王家没有关系。这个时候,除非王家认识张小猪,然后开门让张小猪进来王家,否则张小猪永远没有办法进入王家,更不要说进到王三毛的房间。

不过,如果张小猪通过关系认识了王三毛,并且跟王三毛成为好朋友,那么张小猪就可以通过王三毛进入王家,这个张小猪就是所谓的“其它人,Others”。

在 Linux 里的任何一个文件都具有“User、Group 及 Others”三个权限。此时以王三毛为例,王三毛

这个“文件”的所有者为王三毛,它属于王大毛这个用户组,而张小猪相对于王三毛,则只是一个“其它人(Others)”而已。不过,这里有个特殊的人物要引入,那就是“root”,root 可以到系统内任何想要去的地方。

“用户身份”与该用户所支持的“用户组”概念在 Linux 中很重要,它们可以使多任务 Linux 环境变的更容易管理。

32.2.1 Linux 用户身份与用户组记录的文件

在 Linux 系统默认的情况下,系统上所有的用户的相关信息,都记录在/etc/passwd 这个文件内,而用户的密码则是记录在/etc/shadow 这个文件中。

此外,Linux 所有的用户组名称都记录在/etc/group 内,这三个文件集中了 Linux 系统中账号、密码、用户组信息,不要随便删除这 3 个文件。

第 33 章

Linux 文件权限概念

33.1 Linux 文件属性

ls 是查询 Linux 的文件属性时重要的也是常用的指令，在以 root 的身份登录 Linux 之后，执行 `ls -al` 可以看到：

```
[root@linux ~]# ls -al
total 248
drwxr-x---   9        root  root   4096   Jul 11 14:58   .
drwxr-xr-x  24        root  root   4096   Jul  9 17:25   ..
-rw-------   1        root  root   1491   Jun 25 08:53   anaconda-ks.cfg
-rw-------   1        root  root   13823  Jul 10 23:12   .bash_history
-rw-r--r--   1        root  root   24           Dec  4  2004   .bash_logout
-rw-r--r--   1        root  root   191    Dec  4  2004   .bash_profile
-rw-r--r--   1        root  root   395    Jul  4 11:45   .bashrc
-rw-r--r--   1        root  root   100    Dec  4  2004   .cshrc
drwx------   3        root  root   4096   Jun 25 08:35   .ssh
-rw-r--r--   1        root  root   68495  Jun 25 08:53   install.log
-rw-r--r--   1        root  root   5976   Jun 25 08:53   install.log.syslog
[ 1 ]    [ 2 ][ 3 ] [ 4 ]   [ 5 ]  [ 6 ]          [ 7 ]
```

[属性]　[连接] [所有者] [用户组] [文件容量] [修改日期] [文件名]

ls 与早期的 DOS 指令 dir 功能类似，ls 是“list”的意思，参数“`-al`”则表示列出所有的文件（包含隐藏文件）的详细的权限（Permission）与属性。

第一栏代表这个文件的属性，共有 10 个字符，分别代表 10 个属性：

第一个属性代表这个文件是“目录、文件或连接文件等”：

- 若是 [d] 则是目录；
- 若是 [-] 则是文件；
- 若是 [l] 则表示为连接文件（link file）；
- 若是 [b] 则表示为设备文件里面的可供存储的接口设备；
- 若是 [c] 则表示为设备文件里面的串行端口设备，例如键盘、鼠标等一次性读取设备。

接下来的属性中，3 个为一组，且均为“rwx”的 3 个参数的组合，其中 [r] 代表可读（read）、[w] 代表可写（write）、[x] 代表可执行（execute）。

要注意的是，这 3 个权限的位置不会改变，如果没有权限，就会出现减号（-）。

- 第一组为“文件所有者 (owner) 的权限”；
- 第二组为“同用户组 (usergroup) 的权限”；
- 第三组为“其它非本用户组 (others) 的权限”。

若有一个文件的属性为“-rwxr-xr--”,可简单说明如下:

```
[-][rwx][r-x][r--]
 1  234  567  890
```

1 为:代表这个文件名为目录或文件(上面为文件);

234 为:文件所有者的权限(上面为可读、可写、可执行);

567 为:同用户组用户权限(上面为可读可执行);

890 为:其它用户权限(上面为仅可读)。

上面的属性情况代表一个文件、这个文件的文件所有者可读可写可执行但同用户组的人仅可读与执行,非同用户组的用户仅可读。

除此之外,需要特别留意的是“x”这个标志,若文件名为一个目录,例如上表中的.ssh 这个目录:

```
drwx------  3  root   root   4096   Jun 25 08:35  .ssh
```

可以看到这是一个目录,而且只有 root 可以读写与执行。但是若为下面的样式时,root 的其它人是否可以进入该目录呢?

```
drwxr--r--  3  root   root   4096   Jun 25 08:35  .ssh
```

此时表示非 root 这个账号的其它用户均不可进入.ssh 这个目录,为什么呢?

因为 x 与目录的关系相当的重要,如果在该目录下不能执行任何指令的话,那么自然也就无法进入了,因此特别留意的是如果想要开放某个目录让一些人进来的话,就要该目录的 x 属性开放,因此目录与文件的权限意义并不相同,这是因为目录与文件所记录的数据内容不相同所致。

第二栏表示为连接占用的节点(i-node):

每个文件都会将它的权限与属性记录到文件系统的 i-node 中,由于 Linux 的目录树是使用文件名来记录,因此每个文件名就会连接到一个 i-node,这个跟连接文件(link file)比较有关系。如果是目录的话,那么就与该目录下还有多少目录有关。i-node 属性记录了有多少个不同的文件名连接到相同的一个 i-node 号码。

第三栏表示这个文件(或目录)的“文件所有者”。

第四栏表示文件所属的用户组。

在 Linux 中,ID(如 root 或 test 等账号均是所谓的 ID)就是用户的身份,而且这些账号还可以附属在一个或多个用户组中,例如前面代号为 testgroup 的团体中有三个人,其代号分别是 test1、test2 与 test3,则这三个人为同一用户组,即 testgroup。

那么如果以上图的的文件属性(-rwxrwx---)来看,如果该文件属于 test1 所有,那么 test2、test3 也有读、写、执行的权力,因为它们都属于同一个 group,如果不是属于 test1、test2、test3 的任何一个人,也不属于 testgroup 这个用户组时,那么将不具备读、写、执行这个文件的权限。

第五栏为这个文件的容量大小,默认单位为 Byte。

第六栏为这个文件的创建日期或者是最近的修改日期,分别为月份、日期及时间。如果这个文件被修改的时间距离现在太久了,那么时间部分会仅显示年份而已。使用“ls -l --full-time”可以显示完整的时间格式,包括年、月、日和时间。

特别注意,如果是以中文来安装 Linux 时,那么默认的语系可能会被改为中文。而由于中文无法显示在文字形式的终端上,所以这一栏会成为乱码,此时需要把/etc/sysconfig/i18n 这个文件中的“LC_TIME”修改为 LC_TIME=en_US 再保存后离开,再登录后就可以得到英文字符显示的日期。

至于如何修改该文件,可以以 root 身份用 vi 修改,也可以使用“LANG=en ls -al”之类的语法来显示。

第七栏为这个文件的文件名,如果文件名之前多一个“.”,则代表这个文件为“隐藏文件”,例如上表中的“.bashrc_history”文件名就是隐藏文件,由于有下一个参数为 ls -al,所以连隐藏文件都列出来,如果只输入 ls 则文件名前面是“.”的文件就不会被显示出来。

这 7 个栏位的意义是很重要的,务必清楚的知道各个栏位代表的意义,尤其是第一个栏位的 10 个权限,这是整个 Linux 文件权限的重点之一。

33.1.1 Linux 文件属性的重要性

与 Windows 系统不一样的是，在 Linux 系统（或者说 UNIX like 系统）中，每一个文件都加了很多的属性进来，尤其是用户组的概念。

基本上，这样做的最大的好处是在“数据安全性”上面的。举个简单的例子，在系统中关于系统服务的文件通常只有 root 才能读写或者是执行，例如/etc/shadow 是账号管理的文件，由于该文件记录了系统中的所有账号的数据，因此是很重要的一个配置文件，不能让任何人读取，只有 root 才可以读取，所以该文件的属性就会成为 `-rw-------`。

那么，如果有一个开发团队，希望每个人都可以使用某一些目录下的文件，而非团队的其它人则不予以开放，以上面的例子来说，testgroup 的团队共有三个人，分别是 test1、test2、test3，那么就可以将 test1 的文件属性修改为 `-rwxrwx---`来提供给 testgroup 的团队使用，这可是相当重要的。

在修改 Linux 文件与目录的属性之前，一定要明白什么数据是可变的，什么数据是不能改变的。

33.2 修改文件属性与权限

一个文件的权限大致上就是:用户组、所有者和各种身份的权限等,下面是用于修改文件权限的命令:

- chgrp:改变文件所属用户组
- chown:改变文件所属人
- chmod:改变文件的属性、SUID 等的特性

33.2.1 改变所属用户组:chgrp

改变一个文件的用户组可以直接以 chgrp 来修改,chgrp 就是 change group 的缩写。不过,要改变成为的用户组名称必须是在/etc/group 文件内存在的名称才行,否则就会显示错误。

```
[root@linux ~]# chgrp [-R] dirname/filename ...
参数 
-R:进行递归的持续更改,也就是连同子目录下的所有文件、目录都更改成为这个用户组。
常常用在更改某一目录内所有文件的情况。
范例 
[root@linux ~]# chgrp users install.log
[root@linux ~]# ls -l
-rw-r--r--  1 root users 68495 Jun 25 08:53 install.log
[root@linux ~]# chgrp testing install.log
chgrp: invalid group name 'testing' <== 发生错误信息,找不到这个用户组名。
```

33.2.2 改变文件所有者:chown

改变所有者就是 change owner,这就是 chown 这个指令的用途,要注意的是,用户必须是已经存在系统中的账号,也就是在/etc/passwd 这个文件中有记录的用户名称才可以更改。

chown 还可以直接修改用户组的名称,此外如果要连目录下的所有子目录或文件同时更改文件所有者时,直接加上 -R 的参数就可以,下面是语法与范例:

```
[root@linux ~]# chown [-R] 账号名称 文件或目录
[root@linux ~]# chown [-R] 账号名称: 用户组名称文件或目录
参数 
-R:进行递归的持续更改,也就是连同子目录下的所有文件、目录都更新成为这个用户。
常常用在更改某一目录的情况。
范例 
[root@linux ~]# chown bin install.log
[root@linux ~]# ls -l
-rw-r--r--  1 bin  users 68495 Jun 25 08:53 install.log
[root@linux ~]# chown root:root install.log
[root@linux ~]# ls -l
-rw-r--r--  1 root root 68495 Jun 25 08:53 install.log
```

事实上,chown 也可以有“chown user.group file”的命令,即所有者与用户组间加上小数点“.”也行,只是现在设置账号时也会在账号中加入小数点,这就会造成系统的误判,所以一般建议使用冒号“:”来隔开所有者与用户组。此外 chown 也能单纯地修改所属用户组,例如“`chown .sshd install.log`”就是修改用户组,这就是那个小数点的用途。

需要更改文件的所有者的场景中,最常见的就是在复制文件给其它人时,下面使用最简单的 cp 来说明:

```
[root@linux ~]# cp 来源文件 目的文件
```

假设要将.bashrc 这个文件拷贝成为.bashrc_test 且是要给 bin 这个用户,可以这样做:

```
[root@linux ~]# cp .bashrc .bashrc_test
[root@linux ~]# ls -al .bashrc*
-rw-r--r--  1 root root 395 Jul  4 11:45        .bashrc
-rw-r--r--  1 root root 395 Jul 13 11:31        .bashrc_test
```

但是此时.bashrc_test 还是属于 root 所有,这时即使将文件发给 bin 这个用户了,但是对用户 bin 来说,该文件仍然无法修改的(看属性就知道),所以必须要将这个文件的所有者与用户组进行修改。

33.2.3 改变权限:chmod

文件权限的改变使用的是 chmod 这个指令,但是权限的设定方法有两种,分别可以使用数字或者是符号来进行权限的更改。

1、数字类型改变文件权限

Linux 文件的基本权限就有 9 个,分别是 owner/group/others 三种身份各自的 read/write/excute 权限。

例如,-rwxrwxrwx 这 9 个权限是三个三个一组的,其中可以使用数字来代表各个权限,各权限的对照表如下:

```
r:4
w:2
x:1
```

同一组(owner/group/others)的三个权限(r/w/x)是需要累加的,例如当权限为 -rwxrwx---则是:

```
owner   = rwx   = 4+2+1 = 7
group   = rwx   = 4+2+1 = 7
others  = ---   = 0+0+0 = 0
```

设定权限的更改时,该权限的数字就是 770,更改权限的指令 chmod 的语法是这样的:

```
[root@linux ~]# chmod [-R] xyz 文件或目录
参数:
xyz:就是刚刚提到的数字类型的权限属性,为 rwx 属性数值的相加。
-R:进行递归的持续更改,也就是连同子目录下的所有文件、目录
都更新成为这个用户组之意,常常用在更改某一目录的情况。
```

举例来说,如果要将.bashrc 这个文件所有的属性都打开,那么就执行:

```
[root@linux ~]# ls -al .bashrc
-rw-r--r--  1 root root 395 Jul  4 11:45 .bashrc
[root@linux ~]# chmod 777 .bashrc
[root@linux ~]# ls -al .bashrc
-rwxrwxrwx  1 root root 395 Jul  4 11:45 .bashrc
```

由于一个文件有三组属性,所以可以发现上面 777 为三组,而由于将所有的属性都打开,所以数字都相加,也就是 r+w+x = 4+2+1 = 7。

那如果要将属性变成“-rwxr-xr--”,那就是 [4+2+1][4+0+1][4+0+0]=754,所以需要执行

```
chmod 754 filename
```

指令。

另外,实际应用中最常发生的一个问题就是:以 vi 编辑一个 shell 的文字文件后,它的属性通常是-rw-rw-rw-,也就是 666 的属性,如果要将它变成可执行文件,并且不要让其它人修改该文件的话,那就需要 -rwxr-xr-x 这一个 755 的属性,所以需要这样做:

```
chmod 755 test.sh
```

另外,有些文件不希望被其它人看到,例如 -rwxr-----,那么就执行:

```
chmod 740 filename
```

例:将上述的.bashrc 这个文件的属性改回原来的 -rw-r--r--

```
chmod 644 .bashrc
```

2、符号类型改变文件权限

Linxu 中基本上就是(1)user(2)group(3)others 这三组的 9 个属性,于是就可以通过 u、g、o 来代表这三个组的属性。

此外,a 则代表 all,也就是全部的三组,那么读写的属性就可以写成 r、w、x,也就是可以使用下面的方式:

chmod	u g o	+(加入) -(除去) =(设置)	r w x	文件或目录
chmod	a	+(加入) -(除去) =(设置)	r w x	文件或目录

假如要设定一个文件的属性为“-rwxr-xr-x”,基本上就是:

(1)user(u):具有可读、可写、可执行的权限;

(2)group 与 others(g/o):具有可读与执行的权限。

所以就是:

```
[root@linux ~]# chmod  u=rwx,go=rx  .bashrc
# 注意,u=rwx,go=rx 是用逗号连在一起的,中间并没有任何空白字符。
[root@linux ~]# ls -al .bashrc
-rwxr-xr-x  1 root root 395 Jul  4 11:45 .bashrc
注意,u=rwx,og=rx 这一段文字之间并没有空白字符隔开。
```

如果是“-rwxr-xr--”,可以使用

```
chmod u=rwx,g=rx,o=r filename
```

来设定。

此外,如果不知道原先的文件属性,只想要增加.bashrc 这个文件的每个人均可写入的权限,那么就可以使用:

```
[root@linux ~]# ls -al .bashrc
-rwxr-xr-x  1 root root 395 Jul  4 11:45 .bashrc
[root@linux ~]# chmod  a+w  .bashrc
[root@linux ~]# ls -al .bashrc
-rwxrwxrwx  1 root root 395 Jul  4 11:45 .bashrc
```

而如果是要将属性去掉而不更动其它的属性,例如要去掉所有人的 x 的属性,则:

```
[root@linux ~]# chmod  a-x  .bashrc
[root@linux ~]# ls -al .bashrc
-rw-rw-rw-  1 root root 395 Jul  4 11:45 .bashrc
```

在 + 与 -的状态下,只要是没有指定到的项目,则该属性“不会被变动”,例如上面的例子中,由于仅以 -去掉 x 则其它两个保持当时的值不变,另外利用

```
chmod a+x filename
```

就可以让该程序拥有执行的权限了。

33.3 目录与文件的权限意义

33.3.1 文件的权限意义

文件是实际含有数据的地方，包括一般文本文件、数据库内容文件、二进制可执行文件（binary program）等。

权限对于文件来说，它的意义如下：

（1）r（read）：可读取此文件的实际内容，如读取文本文件的文字内容等。

（2）w（write）：可以编辑、新增或者是修改该文件的内容（但不含删除该文件）；

（3）x（execute）：该文件具有可以被系统执行的权限。

对一个文件具有 w 权限时，可以具有写入、编辑、新增、修改文件的内容的权限，但此时并不具有删除该文件本身的权限。

另外，也必须要更加的注意的是，在 Windows 下一个文件是否具有执行的能力是通过“扩展名”来判断，例如.exe、.bat、.com 等，但是在 Linux 下的文件是否能执行，则是通过是否具有 x 这个属性来决定的，所以跟文件名是没有绝对的关系。

对于文件的 r、w、x 来说，主要都是针对“文件的内容”而言，与文件名的存在与否没有关系，在 Linux 中文件记录的是真实的数据。

33.3.2 目录的权限意义

文件是存放实际数据的所在，目录主要的内容是记录文件名列表，当 r、w、x 针对目录时，简单的说：

（1）r（read contents in directory）：

表示具有读取目录结构清单的权限，所以当具有读取（r）一个目录的权限时就可以查询该目录下的文件名数据，于是就可以利用 ls 这个指令将该目录的内容列表显示出来。

（2）w（modify contents of directory）：这个可写入的权限表示用户将具有更改该目录结构清单的权限，也就是下面这些权限：

- 建立新的文件与目录；
- 删除已经存在的文件与目录（不论该文件的权限是什么）；
- 将已存在的文件或目录进行重命名；
- 移动该目录内的文件、目录位置。

w 则具有相当重要的权限，它可以让用户删除、更新、新建文件或目录，所以说，如果一般身份用户 xx 在/home/xx 这个主文件夹内，无论是谁（包括 root）建立的文件，无论该文件属于谁，无论该文件的属性是什么，xx 这个用户都“有权力将该文件删除”。

（3）x（access directory）：这里的 x 与能否进入该目录有关。

```
[root@linux ~]# cd /tmp
[root@linux tmp]# mkdir testing
[root@linux tmp]# chmod 744 testing
[root@linux tmp]# touch testing/testing
[root@linux tmp]# chmod 600 testing/testing
# mkdir 是在建立目录用的指令，make directory 的缩写。
# 用 root 的身份在/tmp 下建立一个名为 testing 的目录，
# 并且将该目录的权限变为 744，该目录的所有者为 root。
# 另外，touch 可以用来建立一个没有内容的文件，因此 touch testing/testing
# 可以建立一个空的/tmp/testing/testing 文件。
[root@linux tmp]# ls -al
```

```
drwxr--r--   2 root root 4096 Jul 14 01:05 testing
# 目录的权限是 744 ,且所属用户组与用户均是 root,
# 接下来,将 root 的身份切换成为一般身份用户。
# 切换身份成为 dmtsai
[root@linux tmp]# su dmtsai
# su 的指令是用来“切换身份”的一个指令。
# 也就是说,现在当前用户变成 dmtsai,那么 dmtsai 这个人对于
# /tmp/testing 是属于 others 的权限。
[dmtsai@linux tmp] ls -l testing   <== 此时身份为 dmtsai
total 0
# 可以查看里面的信息,因为 dmtsai 具有 r 的权限,不过毕竟权限不够,
# 很多数据竟然是问号 (?) 来的
[dmtsai@linux tmp] cd testing  <== 此时身份为 dmtsai
bash: cd: testing/: Permission denied
# 即使具有 r 的权限,但是没有 x,所以 dmtsai 无法进入/tmp/testing
[dmtsai@linux tmp] exit
[root@linux tmp]# chown dmtsai testing
# 使用 exit 就可以离开 su 的功能。将 testing 目录的所有者设定为
# dmtsai,此时 dmtsai 就成为 owner。
[root@linux tmp]# su dmtsai
[dmtsai@linux tmp] cd testing  <== 此时身份为 dmtsai
[dmtsai@linux testing] ls -l  <== 此时身份为 dmtsai
-rw-------  1 root root 0 Jul 14 01:13 testing
# 再切换身份成为 dmtsai ,此时就能够进入 testing。查看一下内容。
# 发现 testing 这个文件存在,权限是只有 root 才能够存取。
# 那我们测试一下能否删除呢?
[dmtsai@linux testing] rm testing  <== 此时身份为 dmtsai
rm: remove write-protected regular empty file `testing'? y
# 可以删除。
```

能不能进入某一个目录,只与该目录的 x 权限有关,目录只是记录文件名而已,也就是说目录是不能被执行的,目录的“x”代表的是用户能否进入该目录并成为工作目录的用途,而所谓的“工作目录(work directory)”就是当前所在的目录,Linux 中变换目录的命令是“cd(change directory)”。

此外,工作目录对于命令的执行是非常重要的,如果在某目录下不具有 x 的权限,那么就无法切换到该目录下,也就无法执行该目录下的任何命令,即使具有对该目录的 r 权限。

网站服务器中如果只开放目录的 r 权限,将导致任何服务器软件都无法到该目录下读取文件(最多只能看到文件名),访客也总是无法正确查看目录内的文件的内容(显示权限不足),因此要开放目录给任何人浏览时,应该至少也要给予 r 及 x 的权限,但不能赋予 w 权限。

33.4 Linux 文件种类

任何设备在 Linux 下都是文件,甚至连数据通信的接口也有专门的文件负责。

最前面的标志(d 或 -)可以代表目录或文件,那就是不同的文件种类,Linux 的文件种类主要有下面这几种:

1、普通文件(regular file):

就是一般在进行访问的类型的文件,在由 `ls -al` 所显示出来的属性方面,第一个字符为 `[-]`,例如 `[-rwxrwxrwx]`。另外,依照文件的内容,又大略可以分为:

- 纯文本文件(ASCII):这是 UNIX/Linux 系统中最多的一种文件类型,称为纯文本文件是因为内容是我们人类可以直接读到的数据,例如数字、字母等。几乎只要我们可以用来做为设置的文件都属于这一种文件类型。举例来说,可以执行 `cat /.bashrc` 就可以看到该文件的内容,cat 是将一个文件内容读出来的命令。

- 二进制文件(binary):操作系统其实仅认识且可以执行二进制文件,Linux 中的可执行文件(不包括 scripts 和批处理文件)就是这种格式。举例来说,刚刚执行的指令 cat 就是一个二进制文件。

- 数据格式文件(data):有些程序在运行的过程当中会读取某些特定格式的文件,那些特定格式的文件可以被称为数据文件(data file)。举例来说,Linux 在用户登录时都会将登录的数据记录在/var/log/wtmp 那个文件内,该文件就是一个 data file,它能够通过 last 这个指令读出来,但是使用 cat 读取时会读出乱码,因为它是属于一种特殊格式的文件。

2、目录(directory):

第一个属性为 [d] 的文件就是目录,例如 [drwxrwxrwx]。

3、连接文件(link):

类似 Windows 系统下的快捷方式,第一个属性为 [l],例如 [lrwxrwxrwx]。

4、设备与设备文件(device):

与系统外设及存储等相关的一些文件,通常都集中在/dev 这个目录下,通常又分为两种:

(1)块(block)设备文件:就是一些存储数据以提供系统随机访问的接口设备,比如硬盘,现代操作系统可以随机地在硬盘的不同区块读写,例如 1 号硬盘的代码是/dev/hda1 等文件,第一个属性为 [b];

(2)字符(character)设备文件:也就是一些串行端口的接口设备,例如键盘、鼠标等,这些设备的特征就是“一次性读取”,不能够截断输出,这些文件的第一个属性为 [c]。

5、套接字(sockets):

既然被称为数据接口文件,这种类型的文件通常被用于网络上的数据连接。可以启动一个程序来监听用户端的请求,而客户端就可以通过这个 socket 来进行数据的通信了。其第一个属性为 [s],通常会在/var/run 这个目录中看到这种文件类型。

6、管道(FIFO,pipe):

FIFO 也是一种特殊的文件类型,它主要的用途是解决多个程序同时存取一个文件所造成的错误问题。FIFO 是 first-in-first-out 的缩写,第一个属性为 [p]。

那么使用“`ls -al`”命令,就可以简单的经由判断每一个文件的第一个属性来了解这个文件是何种类型。

除了设备文件是系统中很重要的文件,最好不要随意修改之外(通常它也不会让用户修改的),另一个比较有趣的文件就是连接文件,可以将 linux 下的连接文件简单的视为一个文件或目录的快捷方式,而 socket 与 FIFO 文件比较难理解,因为这两个与进程(process)比较有关系,此外也可以通过 man fifo 及 man socket 来查看系统上的说明。

33.5 Linux 文件扩展名

基本上 Linux 的文件是没有所谓的“扩展名”的,一个 Linux 文件能不能被执行与它的第一栏的 10 个属性有关,与文件名根本一点关系也没有。

这与 Windows 的情况不同,在 Windows 下能被执行的文件扩展名通常是.com、.exe、.bat 等,在 Linux 下只要权限当中有 x 的话,例如 [-rwx-r-xr-x] 即代表这个文件可以被执行。

不过,可以被执行跟可以执行成功是不一样的。举例来说,在 root 主文件夹下的 install.log 是一个纯文本文件,如果经由修改权限成为 -rwxrwxrwx,这个文件能够被执行吗?

当然不行,因为它的内容中根本就没有可以执行的数据,所以说这个 x 代表这个文件具有可执行的能力,但是能不能执行成功,当然就得要看该文件的内容。

虽然扩展名没有什么真的帮助,不过由于我们仍然希望可以通过扩展名来了解该文件是什么格式,所以通常还是会以适当的扩展名来表示该文件是什么种类的,下面有数种常用的扩展名:

- *.sh:脚本或批处理文件(scripts),因为批处理文件是使用 shell 写成的,所以扩展名就设成.sh。

- *Z, *.tar, *.tar.gz, *.zip, *.tgz：经过打包的压缩文件。这是因为压缩软件分别为 gunzip、tar 等，使用不同的压缩软件可以取其相关的扩展名。

- *.html、*.php：网页相关文件，分别代表 HTML 语法与 PHP 语法的网页文件。.html 文件可使用 Web 浏览器来直接打开，.php 的文件则可以通过客户端的浏览器发送信息到 Server 端来得到运算后的网页结果。

另外，还有程序语言如 perl 的文件，其扩展名也可能取成.pl 这种文件名。

基本上，Linux 中的文件名只是让用户了解该文件可能的用途而已，真正的执行与否仍然需要权限的规范才行，例如虽然有一个文件为可执行文件，如代理服务器软件 squid，不过如果这个文件的属性被修改成无法执行时，那么它就变成不能执行，这种问题最常发生在文件传送的过程中，例如在网络上下载一个可执行文件，但是偏偏在 Linux 系统中就是无法执行，原因可能就是文件的属性被改变了。不要怀疑，从网络上传送到 Linux 系统的过程中，文件的属性与权限确实是会被改变的。

33.6 Linux 文件限制

当 Linux 系统使用默认的 ext2/ext3 文件系统时，针对文件的文件名长度限制为：

1、单一文件或目录的最大容许文件名为 255 个字符；

2、包含完整路径名称及目录(/)的完整文件名为 4096 个字符。

由于 Linux 在命令行模式下的一些指令操作关系，一般来说，在设置 Linux 下的文件名称时最好可以避免一些特殊字符，例如下面这些：

```
* ? > < ; & ! [ ] | \ ' " ` ( ) { }
```

这些符号在命令行模式下是有特殊意义的。

在 Linux 下单一文件或目录的文件名，加上完整路径时，最长可达 4096 个字符，是相当长的文件名，可以通过 [tab] 按键来确认文件的文件名。

另外，文件名称的开头为小数点“.”时代表这个文件为“隐藏文件”。同时，由于指令执行时常常会使用到 `-option` 之类的参数，所以最好要避免将文件文件名的开头以 -或 + 来命名。

在 Linux 下所有的文件与目录都是由根目录/开始的，/是所有目录与文件的源头，然后再一个一个的分支下来，有点像是树状，因此这种目录配置方式也称目录树(directory tree)。这个目录树主要的特性有：

- 目录树的启始点为根目录(/，root)；

- 每一个目录不止能使用本地端的文件系统，也可以使用网络上的文件系统。举例来说，可以利用 Network File System(NFS)服务器挂载某特定目录等。

- 每一个文件在此目录树中的文件名(包含完整路径)都是独一无二的。

发布目录配置标准 (FHS，Filesystem Hierarchy Standard，http://www.pathname.com/fhs/) 的目的在于希望用户可以了解到已安装软件通常会放置于哪个目录下，并希望独立的软件开发商、操作系统制作者以及想要维护系统的用户都能够遵循。

FHS 的重点在于规范每个特定的目录下应该放置什么样的数据，这样 Linux 操作系统就能够在既有的面貌(目录结构不变)下发展出开发者想要的风格。事实上 FHS 仅是规范出在根目录(/)下各个主要的目录应该是要放置什么样的文件而已，并且 FHS 根据过去的经验一直在持续改版，FHS 依据文件系统使用的频繁与否以及是否允许用户随意改动而将目录定义成为四种互相作用的形态，用下表来说明如下：

	可分享的(shareable)	不可分享的(unshareable)
不变的(static)	/usr(软件) /opt(第三方软件)	/etc(配置文件) /boot(开机与内核文件)
可变动的(variable)	/var/mail(用户邮件信箱) /var/spool/news(新闻组)	/var/run(程序相关) /var/lock(程序相关)

上表中的目录就是一些代表性的目录。

1、可分享的(shareable):

可以分享给其他系统挂载使用的目录,包括执行文件与用户的邮件等数据,是能够分享给网络上其他主机挂载用的目录。

2、不可分享的(unshareable):

本地机器上面运行的设备文件或者是与程序有关的 socket 文件等,因为这些文件仅与本机有关,因此不适合分享。

3、不变的(static):

有些数据是不会经常变动的,不同的 distribution 都一样,例如函数库、文件说明文件、系统管理员所管理的主机服务配置文件等。

4、可变动的(variable):

经常改变的数据,例如登录文件、新闻组等。

FHS 针对目录树架构仅定义出三层目录规范出来,第一层是/下的各个目录应该要放置什么样内容的文件数据,例如/etc 应该要放置设置文件,/bin 与/sbin 则应该要放置可执行文件等。然后则是针对/usr 及/var 这两个目录的子目录来进行定义,例如/var/log 放置系统登录文件,/usr/share 放置共享数据等。

- /(root,根目录):与系统启动有关;
- /usr(UNIX software resource):与软件安装/执行有关;
- /var(variable):与系统运行过程有关。

root 在 Linux 里面的意义有很多种。

- 如果以“账号”的角度来看,所谓的 root 指的是“系统管理员”的身份,
- 如果以“目录”的角度来看,所谓的 root 意即指的是根目录,就是/。

在其它各子目录层级就可以随开发者自行来配置,比如 Fedora 的网络设置数据放在/etc/sysconfig/network-script/目录下,但是 SuSE Server 9 则是放置在/etc/sysconfig/network/目录下,这二者目录名称可是不同的。

特别注意下面这两个特殊的目录:

- . 代表当前的目录,也可以使用./ 来表示;
- .. 代表上一层目录,也可以使用../来代表。

这个. 与.. 目录概念是很重要的,常常会看到 cd .. 或./command 之类的命令执行方式,就是代表上一层与目前所在目录的运行状态。

此外,针对“文件名”与“完整文件名(由/开始写起的文件名)”的字符限制大小为:

- 单一文件或目录的最大允许文件名为 255 个字符;
- 包含完整路径名称及目录(/)的完整文件名为 4096 个字符。

/var/log/下有个文件名为 message,这个 message 文件的最大的文件名可达 255 个字符,var 与 log 这两个上层目录最长也分别可达 255 个字符,但总的来说,由/var/log/messages 这样完整文件名最长则可达 4096 个字符。

33.7 Linux 目录配置的内容

FHS 定义出两层目录内的规范,那么如果来到根目录查看目录数据,会看到:

```
[root@linux ~]# ls -l /
drwxr-xr-x   2 root root  4096 Jul 14 05:22        bin
drwxr-xr-x   3 root root  4096 Jul  9 05:18        boot
drwxr-xr-x   9 root root  4880 Jul 11 00:45        dev
drwxr-xr-x   6 root root  4096 Jun 29 01:06        disk1
drwxr-xr-x   3 root root  4096 Jun 25 08:53        disk2
drwxr-xr-x  83 root root 12288 Jul 14 05:23        etc
drwxr-xr-x   6 root root  4096 May 30 20:07        home
drwxr-xr-x  10 root root  4096 Jul 14 05:23        lib
drwx------   2 root root 16384 Jun 25 16:21        lost+found
drwxr-xr-x   3 root root  4096 Jun 25 19:34        media
drwxr-xr-x   2 root root  4096 Apr 25 23:54        misc
drwxr-xr-x   2 root root  4096 May 23 12:28        mnt
drwxr-xr-x   2 root root  4096 May 23 12:28        opt
dr-xr-xr-x  59 root root     0 Jul 10 01:25        proc
drwx------   9 root root  4096 Jul 13 11:31        root
drwxr-xr-x   2 root root  4096 Jul 14 05:22        sbin
drwxr-xr-x   2 root root  4096 Jun 25 08:23        selinux
drwxr-xr-x   2 root root  4096 May 23 12:28        srv
drwxr-xr-x  10 root root     0 Jul 10 01:25        sys
drwxr-xr-x  10 root root  4096 Jun 25 20:24        system
drwxrwxrwt  10 root root  4096 Jul 14 05:23        tmp
drwxr-xr-x  14 root root  4096 Jun 25 08:27        usr
drwxr-xr-x  24 root root  4096 Jun 25 20:16        var
```

从属性的角度来看,上面的文件名每个都是“目录名称”,较为特殊的是 root,由于 root 这个目录是管理员 root 的主文件夹,这个主文件夹很重要,所以一定要设定成较为严密的 700(rwx------)这个属性。

Linux 中每个目录都是依附在/这个根目录下的,所以在安装的时候一定要有一个/对应的 partition 才能安装的原因即在于此,这也就是树状目录。

33.7.1 根目录(/)的意义与内容

根据 FHS 定义出来的根目录(/)内应该要有下面这些子目录,如下表所示:

目录	应放置的文件内容
/bin	系统有很多放置可执行文件的目录,但/bin 放置的是在单用户维护模式下还能被运行的可执行文件,它们可被 root 与一般账号使用,包括 cat、chmod、chown、date、mv、mkdir、cp、bash 等
/boot	放置开机会用到的文件,包括内核文件、开机菜单与开机所需配置文件等。如果使用的是 grub 来引导加载,则还会有/boot/grub 这个目录。
/dev	Linux 系统上的任何设备与接口设备都会以文件的形式存在于/dev,访问该目录下的某个文件就等于访问某个设备。在此目录下的文件会多出两个属性,分别是 major device number 与 minor device number,系统核心就是通过这两个 number 来判断设备的。比较重要的文件有/dev/null、/dev/zero、/dev/tty*、/dev/ttyS*、/dev/lp*、/dev/hd*、/dev/sd* 等。
/etc	系统主要的配置文件几乎都放置在/etc,例如用户的账号密码文件、各种服务的起始文件等。/etc 下的文件属性是可以让一般用户查看的,但只有 root 有权力修改,并且在此目录下的文件几乎都是文本文件。比较重要的文件有/etc/inittab,/etc/init.d/,/etc/modprobe.d,/etc/X11,/etc/fstab,/etc/sysconfig/等。
/home	系统默认的用户主文件夹(home directory)。在新增一般用户账号时默认的用户主文件夹都会在这里创建。
/lib	放置的仅是在开机时会用到的函数库,以及在/bin 或/sbin 中的命令会调用的函数库,比较重要的是/lib/modules 这个目录内会存放 kernel 相关的模块(驱动程序)。
/media	放置的是可以删除的设备,常见的文件有/media/floppy、/media/cdrom。
/mnt	用于挂载其他额外的设备,在早期 Linux 中/mnt 与/media 的用途相同。
/opt	用于放置第三方软件,在早期 Linux 中则是放置在/usr/local 中。
/root	系统管理员(root)的主文件夹。进入单用户维护模式时而仅挂载根目录时,/root 与根目录(/)放置在同一个分区。
/sbin	放置的是开机过程中所需要的文件,包括开机、修复、还原系统所需要的命令,例如 fdisk、fsck、ifconfig、init、mke2fs、mkswap 等。与/bin 不太一样的地方是这几个目录是给 root 等系统管理用的,但是本目录下的可执行文件还是可以让一般用户用来“查看”但不能修改。至于某些服务器软件程序,一般放置在/usr/sbin 中,本机安装的软件所产生的系统执行文件(system binary)则放置在/usr/local/sbin 中。
/srv	放置的是一些服务启动之后所需要取用的数据。举例来说,WWW 服务器需要的网页数据就可以放置在/srv/www。
/tmp	这是让一般用户或者是正在执行的程序暂时放置文件的地方。/tmp 是任何人都能够存取的,所以需要定期进行清理。当然重要数据不可存放在该目录,FHS 建议开机时应将/tmp 下的数据都删除。

事实上，FHS 针对根目录所定义的标准就仅有上面列举的数据，不过现在实际上在 Linux 中还有几个重要的目录，如下表所示：

目录	应放置的文件内容
/ lost +found	该目录是使用标准的 ext2/ext3 文件系统格式时才会产生的一个目录，目的在于当文件系统发生错误时将一些丢失的片段放置于此目录下，通常这个目录会自动出现在某个 partition 最顶层的目录下，例如加装一个硬盘于/disk 中，那在这个目录下就会自动产生一个这样的目录/disk/lost+found。
/proc	该目录本身是一个“虚拟文件系统（virtual filesystem）”，它放置的数据都是在内存当中，例如系统核心（kernel）、进程信息（processes）、周边设备的状态及网络状态等。因为这个目录下的数据都是在内存当中，所以本身不占任何硬盘空间，包括一些比较重要的文件，例如/proc/cpuinfo、/proc/dma、/proc/interrupts、/proc/ioports 等。
/sys	该目录其实与/proc 非常类似，也是一个虚拟文件系统，主要也是记录与内河相关的信息，包括目前已加载的内核模块与内核检测到的硬件设备信息等，/sys 同样不占硬盘容量。

根目录（/）是整个 Linux 系统最重要的一个目录，所有的目录都是由根目录衍生出来的，而且根目录也与开机、还原、系统修复等操作有关。由于系统开机时需要特定的启动加载程序、内核文件、开机所需程序、函数库等文件数据，若系统出现错误时，根目录也必须要包含有能够修复文件系统的程序。

FHS 希望根目录不要放在非常大的分区内，因为越大的分区会放入越多的数据，这会导致根目录所在分区可能有较多发生错误的机会。

FHS 建议根目录（/）所在分区应该越小越好，并且应用程序所安装的软件最好不要与根目录放在同一个分区内，保持根目录越小越好，这样不但保证性能，而且根目录的文件系统也较不容易发生问题。

除了/bin 之外，/usr/local/bin、/usr/bin 也是放置“用户可执行的 binary file 的目录”。举例来说，ls、mv、rm、mkdir、rmdir、gzip、tar、cat、cp、mount 等指令都放在这个目录中。

一般建议在根目录下只接目录，不要直接有文件在/下。根目录与开机有关，开机过程中仅有根目录会被加载，其他分区则是在开机完成之后才会陆续进行挂载的，因此根目录下与开机过程有关的目录不能放到与根目录不同的分区中，这些目录分别是：

- /etc：设置文件；
- /bin：重要执行文件；
- /dev：开机所需要的设备文件；
- /lib：开机所需要的可执行文件相关的函数库及内核所需的模块；
- /sbin：重要的系统执行文件。

/etc、/bin、/dev、/lib、/sbin 这五个目录都应该要与根目录在一起，不可独立成为某个 partition。

另外，/etc 下重要的目录有：

（1）/etc/init.d/：所有服务的默认启动脚本都是放在这里的，例如要启动或者关闭 iptables，可以运行下面的命令：

- /etc/init.d/iptables start
- /etc/init.d/iptables stop

（2）/etc/xinetd.d/：这就是 super daemon 管理的各项服务的配置文件目录。

（3）/etc/X11：与 X Window 有关的配置文件目录。

33.7.2 /usr 的意义与内容

根据 FHS 的基本定义，/usr 里面放置的数据属于可分享的（shareable）与不可变动的（static），而且/usr 还可以挂载网络文件系统。

/usr 事实上是“UNIX Software Resource”的缩写，在/usr 目录下包含系统的主要程序、图形介面所需要的文件、额外的函数库、本地所自行安装的软件以及共享的目录与文件等，都可以在这个目录当中发现。事实上，它有点像是 Windows 操作系统当中的“Program files”与“Windows”这两个目录的结合。

FHS 建议所有软件开发者应该将它们的数据合理地分别放置到这个目录下的子目录，而不要自行新建软件自己独立的目录。一般来说，/usr 下的重要子目录建议有如下这些：

/usr/bin	绝大部分的用户可使用的命令都放在这里。/usr/bin 与/bin 的不同之处在于后者与开机过程有关。
/usr/sbin	非系统正常运行所需要的系统命令，最常见的就是某些网络服务器软件的服务命令（daemon）。
/usr/include	C/C++ 等程序语言的头文件（header）与包含文件（include）放置处，当以 tarball 方式安装某些数据时会使用到里头的许多包含文件。
/usr/lib	应用软件的函数库、目标文件（object file）以及执行文件或脚本的放置目录，某些软件会提供一些特殊的命令来进行服务器的设置，这些命令也不会经常被系统管理员使用，那就会被放置到该目录下。如果使用的是 x86_64 系统，就可能会生成/usr/lib64 目录。
/usr/local	本地自行安装的软件默认放置的目录，目前也适用于/opt 目录。
/usr/share	共享文件放置的目录，例如下面/usr/share/doc 与/usr/share/man 这两个目录，其中前者放置一些系统说明文件的地方，例如安装了 grub 后可以在这里查到 grub 的说明文件。/usr/share/man 是 manpage 的文件文件目录，也就是使用 man 的时候会去查询的路径。
/usr/src	存放 Linux 系统相关代码的目录，例如/usr/src/linux 为核心源代码。
/usr/X11R6	系统内的 X Window System 重要数据所放置的目录。

在安装完 Linux 之后，基本上所有的配备都有了，但是软件总是可以升级的，例如要升级 proxy 服务，通常软件默认的安装地方就是在/usr/local（local 是“本地”的意思）。为了与系统原先的执行文件有区别，因此升级后的执行文件通常放在/usr/local/bin，为便于管理通常都会将后来才安装上去的软件放置在/usr/local。

另外，之所以命名为 X11R6，是因为最后的 X 版本为第 11 版，且该版的第 6 次发布的意思。

33.7.3 /var 的意义与内容

/var 这个目录也很重要，也是 FHS 规范的第二层目录内容，如果/usr 是安装时会占用较大硬盘空间的目录，那么/var 就是在系统运行后才会逐渐占用硬盘的目录，/var 目录主要针对常态性变动的文件，包括缓存（cache）、登录文件（log file）以及某些软件运行所产生的文件，包括程序文件（lock file、run file）等。此外，某些软件执行过程中会写入的数据库文件，例如 MySQL 数据库也都写入在这个目录中。

/var 下的重要目录如下表所示：

/var/cache	程序文件在运行过程当中的一些暂存文件。
/var/lib	程序本身执行的过程中需要使用到的数据文件放置的目录,在此目录下各自的软件应该要有各自的目录,比如 MySQL 的数据库文件放置到/var/lib/mysql,而 rpm 的数据库则放到/var/lib/rpm 目录下。
/var/log	登录文件放置的目录,其中很重要的包括/var/log/messages、/var/log/wtmp(记录登录者的信息)等。
/var/lock	某些设备或文件一次只能被一个应用程序所使用,如果同时有两个程序使用该设备,就可能报错,因此就要将该设备上锁(lock)以确保该设备只会给单一软件使用。举例来说当刻录机正在刻录光盘时就会把刻录机上锁,只有当刻录机刻录完毕后其他用户就得要等到刻录机设备被解除锁定(也就是说前面的用户使用结束)才能继续使用。
/var/run	某些程序或者是服务启动后,会将它们的 PID 放置在这个目录下。
/var/spool	通常放置队列数据,所谓的“队列”就是排队等待其他程序使用的数据,这些数据被使用后通常会被删除。举例来说,主机收到新电子邮件后,就会放到/var/spool/mail 当中,若信件暂时发不出去就会放置到/var/spool/mqueue 目录下,如果是用户工作排程数据(crontab)则放置在/var/spool/cron 目录中。
/var/mail	放置个人电子邮件的目录,不过这个目录也被放置到/var/spool/mail 中,通常这两个目录是互为连接文件。

33.8 一般主机分区与目录的配置情况

通常一般的大型主机都不会将所有的数据放置在一个目录中(就是只有一个“/”根目录),这有几个目的:

1、安全性考量:

系统通常是在/usr/中,而个人数据则可能放置在/home 当中,至于一些开机数据则放置在/etc 中。如果将所有的数据放在一起,当系统不小心被破坏,则所有的数据也都不见了,这对于一些大型企业来说很危险,因此需要将数据分别放置于不同的目录中会比较保险。

2、便利性:

需要升级系统,Linux 并非需要重新 format 安装,有些数据例如/home 里面的数据为个人用户的数据,似乎与系统无关,所以如果将这些数据分别放置于不同的磁盘,则要升级或者进行一些系统更动时将比较有弹性。或许可以将系统做成这样的 partition 分布:

- /
- /boot
- /usr
- /home
- /var

这是比较常见的目录分布情况,其中:

- /根目录可以分配约 1 GB 以内;
- /boot 大概在 50MB,因为开机文件并不大;
- /var 就至少需要 1GB 以上,因为 mail、proxy 默认的存储区都在这个目录中,除非要将一些设定改变。
- /home 与/usr 通常是最大的,因为所安装的数据都是在/usr/中,而用户数据则放置在/home 中,因此通常大家都会建议将所剩下的磁盘空间平均分配给这两个目录。

不过也不一定,/usr 大概分配 10G 就可以,其它的可以都给/home,也可以保留一些剩余空间来运

行为以后的安装与设定。根据 FHS 的定义，最好能够将/var 独立出来，这样可以提高系统数据的安全性。

对于 Linux 系统来说，只要根目录没有问题，那么就可以进入系统恢复模式进行相关的数据救援工作，不过不同主机的环境与功能用途都不相同，自然其磁盘的分配就会不太一样。

Tips：
/selinux 目录的内容数据也是在内存中的信息，同样不会占用任何的硬盘容量，/selinux 是 Secure Enhance Linux（SELinux）的执行目录，而 SELinux 是 Linux 内核的重要外挂功能之一，它可以用来执行具体权限的管理，主要针对程序（尤其是网络程序）的访问权限来限制。

第 34 章

Linux 文件与目录管理

34.1 目录(Directory)与路径(Path)

Linux 里面的目录是呈“树状”的。也就是有分支的。假设需要在任意一个目录下切换到根目录的 etc 下,那么就应该要使用 cd /etc 命令,这也就是所谓的“绝对路径”,它是从根目录连续接下来的,而如果使用 cd etc 命令,那表示要切换到“当前这个目录下的 etc 目录中”。

相对路径的好处在于方便,但是对于文件的正确性来说,绝对路径的正确度比较好。一般来说,如果是在写程序(shell scripts)时,务必使用绝对路径,虽然绝对路径的写法比较麻烦,但是可以肯定这个写法绝对不会有问题,而如果使用相对路径则可能由于运行的工作环境不同导致一些问题,这个问题在进程调度(at、cron)与例行性命令中尤其重要。

切换目录的指令是 cd,下面这些就是 Linux 中比较特殊的目录:

```
.          代表此层目录,也可以写成./;
..         代表上一层目录,也可有写成../;
-          代表前一个工作目录;
~          代表“目前用户身份”所在的主文件夹;
~account   代表 account 这个使用者的主文件夹;
```

在所有目录下有两个目录是一定会存在的,那就是.与..,它们分别代表当前层与上层目录。在 Linux 文件属性与目录配置中提到根目录(/)是所有目录的最顶层,而且/也有..,可以使用 ls -al /来查看,这时也会注意到根目录本身(.)与其上一层(..)属性完全一样,原来根目录的顶层(..)与它自己(.)是同一个目录。

除了要注意 FHS 目录配置外,在文件名部分也要特别注意,根据文件名写法的不同也可将所谓的路径(path)定义为绝对路径(absolute)与相对路径(relative)。

- 绝对路径:由根目录(/)开始写起的文件名或目录名称,例如/home/dmtsai/.bashrc;
- 相对路径:相对于目前路径的文件名写法,例如./home/dmtsai 或../../home/dmtsai/等。

开头不是/就属于相对路径的写法,而且必须要明白相对路径是以“当前所在路径的相对位置”来表示的。举例来说,目前在/home 这个目录下,如果想要进入/var/log 这个目录,就可以使用如下的命令:

```
cd /var/log(absolute)
cd ../var/log(relative)
```

因为在/home 下,所以要回到上一层(../)之后才能继续往/var 移动。

34.2 目录的相关操作

下面是常见的处理目录的指令:

- cd:切换目录
- pwd:显示目前的目录
- mkdir:建立一个新的目录
- rmdir:删除一个空的目录

34.2.1 cd(切换目录)

用户 dmtsai 的主是/home/dmtsai, 而 root 的主目录则是/root, 假设以 root 身份登录 Linux 系统, 那么下面就是这几个特殊的目录的意义:

```
[root@linux ~]# cd [相对路径或绝对路径]
# 最重要的就是目录的绝对路径与相对路径,还有一些特殊目录的符号。
[root@linux ~]# cd ~dmtsai
# 代表去到 dmtsai 这个用户的主文件夹,也就是/home/dmtsai
[root@linux dmtsai]# cd ~
# 表示回到自己的主文件夹,也就是/root 这个目录
[root@linux ~]# cd
# 没有加上任何路径,也还是代表回到自己主文件夹的意思。
[root@linux ~]# cd ..
# 表示去到目前的上层目录,也就是/root 的上层目录的意思;
[root@linux /]# cd -
# 表示回到刚刚的那个目录,也就是/root。
[root@linux ~]# cd /var/spool/mail
# 这个就是绝对路径的写法,直接指定要去的完整路径名称。
[root@linux mail]# cd ../mqueue
# 这个是相对路径的写法,由/var/spool/mail 去到/var/spool/mqueue 就这样写。
```

cd 是 Change Directory 的缩写, 这是用来切换工作目录的指令。以 root 身份登录 Linux 系统后会在 root 的主文件夹也就是/root 下。使用相对路径时必须要确认目前的路径才能正确的转到想要去的目录。例如上面最后一个例子, 必须要确认是在/var/spool/mail 中, 并知道在/var/spool 中存在 mqueue 目录, 这样才能使用 cd ../mqueue 去到正确的目录, 否则就要直接输入 cd /var/spool/mqueue。

其实提示字符 [root@linux ~]# 中就已经指出目前目录了, 刚登录时会到自己的主文件夹, 而主文件夹还有一个代码, 那就是~ 符号, 从上面的例子可以发现, 使用 cd ~ 可以回到个人的主文件夹中。

另外, 针对 cd 的使用方法, 如果仅输入 cd 代表的就是 cd ~ 的意思, 也就是会回到自己的主文件夹, 而 cd –则代表回到前一个工作目录。

34.2.2 pwd(显示目前所在的目录)

```
[root@linux ~]# pwd [-P]
参数:
-P  :显示出确实的路径,而非使用连接(link)路径。
范例:
[root@linux ~]# pwd
/root   <== 显示出目录
[root@linux ~]# cd /var/mail
[root@linux mail]# pwd
/var/mail
[root@linux mail]# pwd -P
/var/spool/mail
[root@linux mail]# ls -l /var/mail
lrwxrwxrwx  1 root root 10 Jun 25 08:25 /var/mail -> spool/mail
# 看到这里应该知道,/var/mail 是连接文件,连接到/var/spool/mail。
# 所以,加上 pwd -P 的参数后会不以连接文件的数据显示,而是显示正确的完整路径。
```

pwd 是 Print Working Directory 的缩写, 也就是显示目前所在目录的指令, 例如在上面最后的目录是/var/mail 这个目录, 但是提示字符仅显示 mail, 如果想要知道目前所在的目录可以输入 pwd。

此外, 由于很多的套件所使用的目录名称都相同, 例如/usr/local/etc 还有/etc, 但是通常 Linux 仅列出最后面那一个目录, 这时就可以使用 pwd 来知道当前所在目录。

-P 的参数可以让用户取得正确的目录名称而不是以连接文件的路径来显示。如果是 Fedora，那么/var/mail 是/var/spool/mail 的连接文件，所以通过到/var/mail 执行 pwd -P 就能知道这个参数的意义。

34.2.3 mkdir(建立新目录)

```
[root@linux ~]# mkdir [-mp] 目录名称
参数:
-m ： 设定文件的权限，使用-m 参数可以直接设定，不需要看默认权限(umask)。
-p ： 帮助用户直接将所需要的目录递归地建立起来。
范例:
[root@linux ~]# cd /tmp
[root@linux tmp]# mkdir test    <== 建立一个名为 test 的新目录
[root@linux tmp]# mkdir test1/test2/test3/test4
mkdir: cannot create directory `test1/test2/test3/test4':
No such file or directory  <== 无法直接建立此目录。
[root@linux tmp]# mkdir -p test1/test2/test3/test4
# 加了这个-p 参数，可以自行帮用户建立多层目录。
[root@linux tmp]# mkdir -m 711 test2
[root@linux tmp]# ls -l
drwxr-xr-x  3 root  root 4096 Jul 18 12:50 test
drwxr-xr-x  3 root  root 4096 Jul 18 12:53 test1
drwx--x--x  2 root  root 4096 Jul 18 12:54 test2
# 仔细看上面的权限部分，如果没有加上-m 来强制设定属性，系统会使用默认属性。
# 而默认属性则由 umask 决定的。
```

如果想要建立新的目录，那么就使用 mkdir（make directory）命令，不过在默认的情况下用户所需要的目录得一层一层的建立才可以，假如用户要建立一个目录为/home/bird/testing/test1，那么首先必须要有/home 然后是/home/bird，再然后是/home/bird/testing 都必须要存在，才可以建立/home/bird/testing/test1 这个目录。若没有/home/bird/testing 就无法建立 test1 目录。

不过，使用 -p 参数就通过执行如下命令：

```
mkdir -p /home/bird/testing/test1
```

则系统会自动的将/home，/home/bird，/home/bird/testing 依序的建立起目录，并且如果该目录本来就已经存在时，系统也不会显示错误信息。

另外，有个地方必须要先有概念，那就是“默认权限”。可以利用 -m 来强制给予一个新的目录相关的属性，例如上面可以利用 -m 711 来强制给予新的目录 drwx--x--x 的属性。不过，如果没有附加 -m 属性时，那么默认的新建目录属性就跟 umask 有关。

34.2.4 rmdir(删除“空”的目录)

```
[root@linux ~]# rmdir [-p] 目录名称
参数:
-p ： 连同上层“空的”目录也一起删除
范例:
[root@linux tmp]# ls -l
drwxr-xr-x  3 root  root 4096 Jul 18 12:50 test
drwxr-xr-x  3 root  root 4096 Jul 18 12:53 test1
drwx--x--x  2 root  root 4096 Jul 18 12:54 test2
[root@linux tmp]# rmdir test
[root@linux tmp]# rmdir test1
rmdir: `test1': Directory not empty
[root@linux tmp]# rmdir -p test1/test2/test3/test4
[root@linux tmp]# ls -l
drwx--x--x  2 root  root 4096 Jul 18 12:54 test2
```

```
# 利用-p 参数就可以将 test1/test2/test3/test4 直接删除。
# 不过要注意的是,rmdir 仅能"删除空的目录"。
```

如果想要删除旧有的目录时,可以使用 rmdir 命令,不过要注意目录需要一层一层的删除才行,而且被删除的目录里面不能还有其它的目录或文件,这也是所谓的空的目录(empty directory)的原因,如果要将所有目录下的内容都删除,就必须使用 `rm -rf test`。

34.2.5 关于执行文件路径的变量:$PATH

用户在执行命令时,系统会依照 PATH 变量的设定去每个 PATH 定义的路径下搜索执行文件,先搜索到的指令先被执行。通过执行 `echo $PATH` 可以查看有哪些目录被定义出来了,echo 有"显示、打显示出"的意思,而 PATH 前面加的 $ 表示后面接的是变量,所以就会显示出目前的 PATH。

PATH 对于执行文件来说是个很重要的"变量",它主要是用来规范指令搜索的目录,而每个目录是有顺序的,这些目录中间以冒号":"来分隔。

现在若将 ls 移动到/root 下(通过 `mv /bin/ls /root`),然后用户本身也在/root 下(通过 `cd /root`),但是当执行 ls 时却会报错,这就是因为 PATH 这个变量中没有/root 这个目录,而现在又将 ls 移动到了/root,于是系统就找不到可执行文件了,因此就会告诉用户 `command not found`。要解决这个问题有两个方法,可以直接将/root 路径加入 PATH 中,执行下面的命令:来增加 PATH 搜索目录,或者使用完整文件名执行指令,也就是直接使用相对或绝对路径来执行,例如:

```
[root@linux ~]# /root/ls
[root@linux ~]# ./ls
```

这时因为在同一个目录中,而我们又知道在同一个目录中的目录符号为".",因此,就以上面的`./ls`来执行也可以。

如果有两个 ls 文件在不同的目录中,例如/usr/local/bin/ls 下与/bin/ls,那么当执行 ls 时,哪个 ls 会被执行?这时可以找出 PATH 里面哪个目录先被查询,则那个目录下的文件就会被先执行。

为了安全起见,不建议将"."加入 PATH 的查询目录中,并且:

- 不同身份用户默认的 PATH 不同,默认能够随意执行的命令也不同;
- PATH 是可以修改的,所以一般用户还是可以通过修改 PATH 来执行某些位于/sbin 或/usr/sbin 下的命令来查询;
- 使用绝对路径或相对路径直接指定某个命令的文件名来执行,会比查询 PATH 要正确;
- 命令应该要放置到正确的目录下,执行才会比较方便;
- 当前目录(.)最好不要放到 PATH 当中。

34.3 文件与目录管理

文件与目录的管理包括"显示属性"、"拷贝"、"删除文件"及"移动文件或目录"等,在执行程序时系统默认有一个搜索的路径顺序,如果有两个以上相同文件名的执行文件分别在不同的路径时,就需要特别留意。

34.3.1 文件与目录的查看:ls

```
[root@linux ~]# ls [-aAdfFhilRS] 目录名称
[root@linux ~]# ls [--color=never,auto,always] 目录名称
[root@linux ~]# ls [--full-time] 目录名称
参数:
-a  :全部的文件,连同隐藏文件(开头为. 的文件)一起列出来。
-A  :全部的文件,连同隐藏文件,但不包括. 与.. 这两个目录,一起列出来。
-d  :仅列出目录本身,而不是列出目录内的文件数据
-f  :直接列出结果,而不进行排序(ls 默认会以文件名排序)。
```

```
-F   :根据文件、目录等信息,给予附加数据结构,例如 □
       *:代表可执行文件; /:代表目录; =:代表 socket 文件;|:代表 FIFO 文件;
-h   :将文件容量以较易读的方式(例如 GB、KB 等)列出来;
-i   :列出 inode 位置,而非列出文件属性;
-l   :长数据串列出,包含文件的属性等等数据;
-n   :列出 UID 与 GID 而非使用者与用户组的名称。
-r   :将排序结果反向输出,例如原本文件名由小到大,反向则为由大到小;
-R   :连同子目录内容一起列出来;
-S   :以文件容量大小排序!
-t   :依时间排序
--color=never   :不要依据文件特性给予颜色显示;
--color=always  :显示颜色
--color=auto    :让系统自行依据设定来判断是否给予颜色
--full-time     :以完整时间模式(包含年、月、日、时、分)输出
--time=atime,ctime :输出 access 时间或改变权限属性时间(ctime)而非内容变更时间(modification time)
```

在 Linux 系统中,通过 ls 指令可以知道文件或者是目录的相关信息。不过 Linux 的文件所记录的信息实在是太多了,ls 不需要全部都列出来,所以当只是执行 ls 时,默认显示的只有非隐藏文件的文件名、以文件名进行排序及文件名代表的颜色显示等。比如说执行 `ls /etc` 之后,只有经过排序的文件名以及以蓝色显示目录及白色显示一般文件。

那如果还想要加入其它的显示信息时可以加入上面提到的参数,举例来说,一直用到的有 `-l` 这个长串显示数据内容以及将隐藏文件也一起显示出来的 `-a` 参数等。

范例一:将主文件夹下的所有文件列出来(含属性与隐藏文件)

```
[root@linux ~]# ls -al ~
total 252
drwxr-x---   9 root root  4096 Jul 16 23:40 .
drwxr-xr-x  24 root root  4096 Jul 16 23:45 ..
-rw-------   1 root root  1491 Jun 25 08:53 anaconda-ks.cfg
-rw-------   1 root root 12543 Jul 18 01:23 .bash_history
-rw-r--r--   1 root root    24 Dec  4  2004 .bash_logout
-rw-r--r--   1 root root   191 Dec  4  2004 .bash_profile
-rw-r--r--   1 root root   395 Jul  4 11:45 .bashrc
-rw-r--r--   1 root root 68495 Jun 25 08:53 install.log
-rw-r--r--   1 root root  5976 Jun 25 08:53 install.log.syslog
drwx------   2 root root  4096 Jul  4 16:03 .ssh
-rw-------   1 root root 12613 Jul 16 23:40 .viminfo
# 此时会看到以. 开头的几个文件,以及目录文件./../.ssh 等,
# 不过,目录文件都是以深蓝色显示。
```

范例二:承上题,不显示颜色,但在文件名末尾显示出该文件名代表的类型(type)

```
[root@linux ~]# ls -alF --color=never  ~
total 252
drwxr-x---   9 root root  4096 Jul 16 23:40 ./
drwxr-xr-x  24 root root  4096 Jul 16 23:45 ../
-rw-------   1 root root  1491 Jun 25 08:53 anaconda-ks.cfg
-rw-------   1 root root 12543 Jul 18 01:23 .bash_history
-rw-r--r--   1 root root    24 Dec  4  2004 .bash_logout
-rw-r--r--   1 root root   191 Dec  4  2004 .bash_profile
-rw-r--r--   1 root root   395 Jul  4 11:45 .bashrc
-rw-r--r--   1 root root 68495 Jun 25 08:53 install.log
-rw-r--r--   1 root root  5976 Jun 25 08:53 install.log.syslog
drwx------   2 root root  4096 Jul  4 16:03 .ssh/
-rw-------   1 root root 12613 Jul 16 23:40 .viminfo
# 注意看显示结果的第一行,./代表的是“目前目录下”的意思。
```

范例三:完整的呈现文件的修改时间 *(modification time)

```
[root@linux ~]# ls -al --full-time  ~
total 252
```

```
drwxr-x---  9 root root  4096 2005-07-16 23:40:13.000000000 +0800 .
drwxr-xr-x 24 root root  4096 2005-07-16 23:45:05.000000000 +0800 ..
-rw-------  1 root root 12543 2005-07-18 01:23:33.000000000 +0800 .bash_history
-rw-r--r--  1 root root    24 2004-12-04 05:44:13.000000000 +0800 .bash_logout
-rw-r--r--  1 root root   191 2004-12-04 05:44:13.000000000 +0800 .bash_profile
-rw-r--r--  1 root root   395 2005-07-04 11:45:16.000000000 +0800 .bashrc
-rw-r--r--  1 root root 68495 2005-06-25 08:53:34.000000000 +0800 install.log
drwx------  2 root root  4096 2005-07-04 16:03:24.000000000 +0800 .ssh
-rw-------  1 root root 12613 2005-07-16 23:40:13.000000000 +0800 .viminfo
# 上面的“时间”栏位变成了较为完整的格式。
# 一般来说,ls -al 仅列出目前短格式的时间,有时不会列出年份,
# 通过--full-time 可以查看到比较正确的完整时间格式。
```

其实 ls 的用法还有很多,包括查看文件所在 i-node 的 `ls -i` 参数以及用来进行文件排序的 `-S` 参数,还有用来查看不同时间的动作的 `--time=atime` 等参数。而这些参数的存在都是因为 Linux 文件系统记录了很多有用的信息的缘故,Linux 的文件系统中与权限、属性有关的数据放在 i-node 里面。

无论如何,ls 最常被使用到的功能还是 `-l` 参数,为此很多 distribution 在默认的情况中,已经将 ll 设定成为 `ls -l` 了,其实这个功能是 bash shell 的 alias 功能,也就是说输入 `ll` 就等于是输入 `ls -l`。

34.3.2 复制、移动与删除:cp, mv, rm

要复制文件或目录,可以使用 cp(copy)指令,不过 cp 指令除了用于单纯的复制,还可以建立连接文件,对比两文件的新旧而予以更新以及复制整个目录等。移动目录与文件则使用 mv(move),mv 指令也可以直接用来进行重命名(rename)等操作。

1、cp(复制文件或目录)

```
[root@linux ~]# cp [-adfilprsu] 来源文件(source) 目的文件(destination)
[root@linux ~]# cp [options] source1 source2 source3 .... directory
参数:
-a  :相当于-pdr 的意思;
-d  :若来源文件为连接文件的属性(link file),则复制连接文件属性而非文件本身;
-f  :为强制(force)的意思,若有重复或其它疑问时,不会询问使用者,而强制复制;
-i  :若目的文件(destination)已经存在时,在覆盖时会先询问是否真的执行。
-l  :进行硬式连结(hard link)的连接文件建立,而非复制文件本身;
-p  :连同文件的属性一起复制过去,而非使用默认属性;
-r  :递归持续复制,用于目录的复制行为;
-s  :复制成为符号连接文件(symbolic link),也就是“快捷方式”文件;
-u  :若 destination 比 source 旧才更新 destination。
最后需要注意的,如果来源文件有两个以上,则最后一个目的文件一定要是“目录”才行!
范例:
范例一:将主文件夹下的.bashrc 复制到/tmp 下,并更名为 bashrc
[root@linux ~]# cd /tmp
[root@linux tmp]# cp ~/.bashrc bashrc
[root@linux tmp]# cp -i ~/.bashrc bashrc
cp: overwrite `basrhc'? n
# 重复作两次动作,由于/tmp 下已经存在 bashrc 了,加上-i 参数,
# 则在覆盖前会询问使用者是否确定,可以按下 n 或者 y。
# 但是,反过来说,如果不想要询问时,则加上-f 参数来强制直接覆盖。

范例二:将/var/log/wtmp 复制到/tmp 下
[root@linux tmp]# cp /var/log/wtmp . <== 想要复制到目前的目录,最后的. 不要忘
[root@linux tmp]# ls -l /var/log/wtmp wtmp
-rw-rw-r--  1 root utmp 71808 Jul 18 12:46 /var/log/wtmp
-rw-r--r--  1 root root 71808 Jul 18 21:58 wtmp
# 在不加任何参数的情况下,文件的所属者会改变,权限也跟着改变,连文件建立的时间也不一样了。
# 如果想要将文件的所有特性都一起复制过来,可以加上-a。
```

```
[root@linux tmp]# cp -a /var/log/wtmp wtmp_2
[root@linux tmp]# ls -l /var/log/wtmp wtmp_2
-rw-rw-r--  1 root utmp 71808 Jul 18 12:46 /var/log/wtmp
-rw-rw-r--  1 root utmp 71808 Jul 18 12:46 wtmp_2
# 整个数据特性完全一模一样,这就是-a 的特性。

范例三:复制/etc/这个目录下的所有内容到/tmp 下
[root@linux tmp]# cp /etc/ /tmp
cp: omitting directory `/etc'   <== 如果是目录,不能直接复制,要加上-r 的参数
[root@linux tmp]# cp -r /etc/ /tmp
# 再次强调,-r 是可以复制目录,但是文件与目录的权限会被改变。
# 所以,也可以利用 cp -a /etc /tmp 来执行指令。

范例四:将范例一复制的 bashrc 建立一个连接文件(symbolic link)
[root@linux tmp]# ls -l bashrc
-rw-r--r--  1 root root 395 Jul 18 22:08 bashrc
[root@linux tmp]# cp -s bashrc bashrc_slink
[root@linux tmp]# cp -l bashrc bashrc_hlink
[root@linux tmp]# ls -l bashrc*
-rw-r--r--  2 root root 395 Jul 18 22:08 bashrc
-rw-r--r--  2 root root 395 Jul 18 22:08 bashrc_hlink
lrwxrwxrwx  1 root root   6 Jul 18 22:31 bashrc_slink -> bashrc
# bashrc_slink 是由-s 的参数造成的,建立的是一个"快捷方式",
# 所以会看到在文件的最右边显示这个文件是"连接"到哪里去的。
# 建立了 bash_hlink 之后,bashrc 与 bashrc_hlink 所有的参数都一样,
# 只是,第二栏的 link 数改变成为 2,而不是原本的 1。

范例五:若 ~/.bashrc 比/tmp/bashrc 新才复制过来
[root@linux tmp]# cp -u ~/.bashrc /tmp/bashrc
# -u 的特性是在目标文件与来源文件有差异时才会复制,所以常被用于"备份"的工作中。

范例六:将范例四产生的 bashrc_slink 复制成为 bashrc_slink_2
[root@linux tmp]# cp bashrc_slink bashrc_slink_2
[root@linux tmp]# ls -l bashrc_slink*
lrwxrwxrwx  1 root root   6 Jul 18 22:31 bashrc_slink -> bashrc
-rw-r--r--  1 root root 395 Jul 18 22:48 bashrc_slink_2
# 原本复制的是连接文件,但是却将连接文件的实际文件复制过来了
# 也就是说,如果没有加上任何参数时,复制的是原始文件,而非连接文件的属性。
# 若要复制连接文件的属性,就得要使用-d 或者-a 参数。

范例七:将主文件夹的.bashrc 及.bash_history 复制到/tmp 下
[root@linux tmp]# cp ~/.bashrc ~/.bash_history /tmp
# 可以将多个数据一次复制到同一个目录去。
```

不同身份的用户执行 cp 命令会有不同的结果产生,尤其是使用 -a、-p 参数时,一般来说,如果去复制别人的数据(当然必须要有 read 权限才行)时,总是希望复制到的数据最后是我们自己的,所以在默认的条件中,cp 的来源文件与目的文件的权限是不同的,目的文件的所有者通常会是指令操作者本身。举例来说,上面的范例二中,由于用户是 root 的身份,因此复制过来的文件所有者与群组就改变成为 root 所有。

由于具有这个特性,因此在进行备份的时候,某些需要特别注意的特殊权限文件,例如密码文件(/etc/shadow)以及一些配置文件就不能直接以 cp 来复制,而必须要加上 -a 或者是 -p 等可以完整复制文件权限的参数。另外,如果想要复制文件给其它的使用者,也必须要注意到文件的权限(包含读、写、执行以及文件所有者等),否则其它人还是无法针对给予的文件进行修改的动作。

上面的范例当中,第四个范例使用 -l 及 -s 都会建立所谓的连接文件(link file),但是这两种连接文件确有不一样的表现形式,其中 -l 就是所谓的 hard link,-s 则是 symbolic link,

在使用 cp 命令进行复制时,必须要清楚的注意以下事项:

- 是否需要完整的保留来源文件的信息?
- 来源文件是否为连接文件(symbolic link file)?
- 来源文件是否为特殊的文件,例如 FIFO,socket?
- 来源文件是否为目录?

2、rm(移除文件或目录)

```
[root@linux ~]# rm [-fir] 文件或目录
参数:
-f  :就是 force 的意思,强制移除;
-i  :互动模式,在删除前会询问使用者是否执行;
-r  :递归删除,最常用在目录的删除。
范例:
范例一:建立一文件后予以删除
[root@linux ~]# cd /tmp
[root@linux tmp]# cp ~/.bashrc bashrc
[root@linux tmp]# rm -i bashrc
rm: remove regular file `bashrc'? y
# 如果加上-i 的参数就会主动询问,如果不要询问,就加-f 参数。

范例二:删除一个不为空的目录
[root@linux tmp]# mkdir test
[root@linux tmp]# cp ~/.bashrc test/ <== 将文件复制到此目录去就不是空的目录了
[root@linux tmp]# rmdir test
rmdir: `test': Directory not empty <== 删不掉,因为这不是空的目录。
[root@linux tmp]# rm -rf test

范例三:删除一个带有 - 开头的文件
[root@linux tmp]# ls *aa*
-rw-r--r--  1 root  root      0 Aug 22 10:52 -aaa-
[root@linux tmp]# rm -aaa-
rm: invalid option -- a
Try `rm --help' for more information.  <== 因为 "-" 是参数
[root@linux tmp]# rm ./-aaa-
```

移除文件或目录的指令 rm(remove)相当于 dos 下的 del 指令,这里要注意的是,通常在 Linux 系统下,为了怕文件被误删,所以很多 distributions 都已经默认有 -i 参数,-i 是指每个文件被删掉之前都会让使用者确认一次,以预防误删文件。如果要连目录下的东西都一起删掉的话,例如子目录里面还有子目录时,那就要使用 -rf 这个参数了。不过使用 rm -rf 指令之前,该目录或文件“肯定”会被 root 删除,系统不会再次询问是否要删除,如果确定要删除干净,那可以使用 rm -rf 来循环删除。

另外,文件名最好不要使用“-”开头,因为“-”后面接的是参数,因此单纯的使用 rm -aaa-系统的指令就会误判,如果使用后面会谈到的正规表示法也还是会出问题的。所以只能用避过首位字符是“-”的方法。就是加上当前目录“./”即可,如果 man rm 的话,其实还有一种方法,那就是 rm -- -aaa-也可以。

3、mv (移动文件与目录,或更名)

```
[root@linux ~]# mv [-fiu] source destination
[root@linux ~]# mv [options] source1 source2 source3 .... directory
参数:
-f  :force,强制的意思,强制直接移动而不询问;
-i  :若目标文件(destination)已经存在时,就会询问是否覆盖。
-u  :若目标文件已经存在且 source 比较新,才会更新(update)。
范例:
范例一:复制一文件,建立一目录,将文件移动到目录中
[root@linux ~]# cd /tmp
[root@linux tmp]# cp ~/.bashrc bashrc
[root@linux tmp]# mkdir mvtest
```

```
[root@linux tmp]# mv bashrc mvtest
# 将某个文件移动到某个目录去,就是这样做。

范例二:将刚刚的目录名称更名为 mvtest2
[root@linux tmp]# mv mvtest mvtest2
# 其实在 Linux 下还有 rename 命令用于重命名。

范例三:再建立两个文件,再全部移动到 /tmp/mvtest2 当中
[root@linux tmp]# cp ~/.bashrc bashrc1
[root@linux tmp]# cp ~/.bashrc bashrc2
[root@linux tmp]# mv bashrc1 bashrc2 mvtest2
# 注意到这边,如果有多个来源文件或目录,则最后一个目标档一定是"目录"。
# 意思是说,将所有的数据移动到该目录的意思。
```

mv(move)命令中也可以使用 -u(update)来测试新旧文件,看看是否需要移动。另外一个用途就是"变更文件名",可以使用 mv 来变更一个文件的文件名。Linux 另外提供 rename 命令用来更改大量文件的文件名,可以利用 man rename 来查看一下。

34.3.3 取得路径的文件名称与目录名称

介绍完整文件名(包含目录名称与文件名称)中提到,完整文件名最长可以到达 4096 个字符,那么怎么知道那个是文件名?那个是目录名呢?其实,取得文件名或者是目录名称,一般的用途应该是在写程序的时候用来判断的。下面简单的以几个范例来介绍 basename 与 dirname 命令。

```
[root@linux ~]# basename /etc/sysconfig/network
network  <== 很简单,就是取得最后的文件名。
[root@linux ~]# dirname /etc/sysconfig/network
/etc/sysconfig  <== 取得的变成目录名。
```

34.4 文件内容查看

如果要查看一个文件的内容,最常使用的显示文件内容的指令是 cat、more 及 less。此外,如果要查看一个大型的文件,但是我们只需要后端的几行字而已,可以使用 tail,此外使用 tac 这个指令也可以完成。

- cat:由第一行开始显示文件内容;
- tac:从最后一行开始显示;
- nl:显示的时候输出行号;
- more:一页一页的显示文件内容;
- less:less 与 more 类似,但是比 more 更好的是,less 可以往前翻页;
- head:只看头几行;
- tail:只看尾巴几行;
- od:以二进位的方式读取文件内容。

34.4.1 直接查看文件内容

直接查看一个文件的内容可以使用 cat/tac/nl 这几个命令。

1、cat(concatenate)

cat 是 Concatenate(连续)的简写,主要的功能是将一个文件的内容连续的显示出在显示器上,例如上面的例子中们将/etc/issue 显示出来。如果加上 -n,则每一行前面还会加上行号。当文件内容的行数超过 40 行以上,仅使用 cat 根本来不及看,所以配合 more 或者 less 来执行会比较好。

此外,如果是一般的 DOS 文件时,就需要特别留意一些奇怪的符号了,例如断行与 [tab] 等,要显示出来就得加入 -A 之类的参数。

2、tac(反向列示)

```
[root@linux ~]# tac /etc/issue
Kernel \r on an \m
Fedora Core release 4 (Stentz)
# 与刚刚上面的范例一比较,是由最后一行先显示。
```

tac 刚好是将 cat 反写过来,所以 tac 的功能也跟 cat 相反,cat 是由“第一行到最后一行连续显示在显示器上”,而 tac 则是“由最后一行到第一行反向在显示器上显示出来”。

3、nl(添加行号打印)

```
[root@linux ~]# nl [-bnw] 文件
参数:
-b  :指定行号指定的方式,主要有两种 □
      -b a :表示不论是否为空行,也同样列出行号;
      -b t :如果有空行,空的那一行不要列出行号;
-n  :列出行号表示的方法,主要有三种 □
      -n ln :行号在显示器的最左方显示;
      -n rn :行号在自己栏位的最右方显示,且不加 0;
      -n rz :行号在自己栏位的最右方显示,且加 0;
-w  :行号栏位的占用的位数。
范例:
范例一:列出/etc/issue 的内容
[root@linux ~]# nl /etc/issue
     1  Fedora Core release 4 (Stentz)
     2  Kernel \r on an \m

# 注意看,这个文件其实有三行,第三行为空白(没有任何字符),
# 因为它是空白行,所以 nl 不会加上行号,如果确定要加上行号,可以这样做:
[root@linux ~]# nl -b a /etc/issue
     1  Fedora Core release 4 (Stentz)
     2  Kernel \r on an \m
     3
# 如果要让行号前面自动补上 0,可以这样
[root@linux ~]# nl -b a -n rz /etc/issue
000001  Fedora Core release 4 (Stentz)
000002  Kernel \r on an \m
000003
# 自动在自己栏位的地方补上 0,默认栏位是六位数,如果想要改成 3 位数,可以这样
[root@linux ~]# nl -b a -n rz -w 3 /etc/issue
001     Fedora Core release 4 (Stentz)
002     Kernel \r on an \m
003
# 变成仅有 3 位数
```

nl 可以将输出的文件内容自动加上行号,其结果与 cat -n 不太一样,nl 可以将行号做比较多的显示设计,包括位数与是否自动补齐 0 等。

34.4.2 可翻页查看

nl、cat、tac 等都是一次性的将数据显示到显示器上面,如果要一页一页翻页查看,可以使用 more 与 less 命令。

1、more(翻页查看)

```
[root@linux ~]# more /etc/man.config
# Generated automatically from man.conf.in by the
# configure script.
# man.conf from man-1.5p
```

```
#
....... 中间省略.......
--More--(28%) <== 重点在这一行。
```

仔细的看上面的范例,如果 more 后面接的文件长度大于显示器输出的行数时,就会出现类似上面的图示。

重点在最后一行,最后一行会显示出目前显示的百分比,而且还可以在最后一行输入一些有用的指令。在 more 这个程序的运作过程中,有几个快捷键如下:

- 空格键(space):代表向下翻一页;
- Enter :代表向下翻“一行”;
- /字符串:代表在这个显示的内容中向下搜索“字符串”;
- :f :立刻显示出文件名以及目前显示的行数;
- q :代表立刻离开 more,不再显示该文件内容。

比较有用的是搜索字符串的功能,举例来说,使用 `more /etc/man.config` 来测试该文件,若想要在该文件内搜索 MANPATH 这个字符串,可以这样做:

```
[root@linux ~]# more /etc/man.config
#
# Generated automatically from man.conf.in by the
# configure script.
# man.conf from man-1.5p
#
....... 中间省略.......
/MANPATH   <== 输入/之后,光标就会自动跑到最底下一行等待输入。
```

如同上面的说明,输入/之后光标就会跑到最下面一行并等待输入,输入了字符串之后,more 就会开始向下搜索该字符串,而重复搜索同一个字符串,可以直接按下 n。最后不想要看了,就按下 q 即可离开 more。

2、less(翻页查看)

```
[root@linux ~]# less /etc/man.config
# Generated automatically from man.conf.in by the
# configure script.
#
# man.conf from man-1.5p
...... 中间省略........
:   <== 这里可以等待输入指令。
```

less 的用法比起 more 又更加弹性,在使用 more 命令时用户并没有办法向前面翻,只能往后面看,但若使用了 less 就可以使用 [pageup]、[pagedown] 等按键的功能来往前往后翻看文件。

除此之外,在 less 里可以拥有更多的“搜索”功能,不仅可以向下搜索,也可以向上搜索,基本上可以输入的指令有:

- 空格键:向下翻动一页;

pagedown :向下翻动一页;

pageup :向上翻动一页;

- /字符串:向下搜索“字符串”的功能;
- ? 字符串:向上搜索“字符串”的功能;
- n :重复前一个搜索(与/或? 有关);
- N :反向的重复前一个搜索(与/或? 有关);
- q :离开 less。

使用 less 查看文件内容还可以进行搜索,其实 less 还有很多的功能,而且 less 的使用界面和环境与 man page 很类似,并且实际上 man 这个命令就是调用 less 来显示说明文件的内容的。

34.4.3 数据选取

可以将输出的数据作一个最简单的选取,那就是取出前面(head)与取出后面(tail)的文字,不过要注意 head 与 tail 都是以“行”为单位来进行数据选取的。

1、head(取出前面几行)

```
[root@linux ~]# head [-n number] 文件
参数:
-n   :后面接数字,代表显示几行的意思
范例:
[root@linux ~]# head /etc/man.config
# 默认的情况中,显示前面十行。若要显示前 20 行,就要这样:
[root@linux ~]# head -n 20 /etc/man.config
```

head 的英文意思就是“头”,那么 head 的用法自然就是显示出一个文件的前几行,若没有加上 -n 这个参数时,默认只显示 10 行,若只要一行,那就使用 `head -n 1 filename` 即可。

2、tail(取出后面几行)

```
[root@linux ~]# tail [-n number] 文件
参数:
-n   :后面接数字,代表显示几行的意思
范例:
[root@linux ~]# tail /etc/man.config
# 默认的情况中,显示最后的 10 行,若要显示最后的 20 行,就得要这样:
[root@linux ~]# tail -n 20 /etc/man.config
```

tail 的用法跟 head 的用法类似,只是显示的是后面几行,tail 默认也是显示 10 行,若要显示非 10 行,就加 `-n number` 参数。

假如想要显示~/.bashrc 的第 11 到第 20 行,那么可以先取前 20 行,再取后 10 行,所以结果就是

```
head -n 20 ~/.bashrc | tail -n 10
```

这样就可以得到第 11 到第 20 行之间的内容,这里涉及到管道命令。

34.4.4 非纯文本文件:od

如果想要查看非文本文件,举例来说,例如/usr/bin/passwd 这个执行文件的内容时,由于可执行文件通常是 binary file,使用上面提到的指令来读取其内容时确实会产生类似乱码的数据,这时可以利用 od 这个指令来读取。

```
[root@linux ~]# od [-t TYPE] 文件
参数:
-t   :后面可以接各种“类型(TYPE)”的输出,例如:
       a        :利用默认的字符来输出;
       c        :使用 ASCII 字符来输出
       d[size]  :利用十进位(decimal)来输出数据,每个整数占用 size bytes;
       f[size]  :利用浮点数值(floating)来输出数据,每个数占用 size bytes;
       o[size]  :利用八进位(octal)输出数据,每个整数占用 size bytes;
       x[size]  :利用十六进位(hexadecimal)输出数据,每个整数占用 size bytes;
范例:
[root@linux ~]# od -t c /usr/bin/passwd
0000000 177   E   L   F 001 001 001  \0  \0  \0  \0  \0  \0  \0  \0  \0
0000020 002  \0 003  \0 001  \0  \0  \0 260 225 004  \b   4  \0  \0  \0
0000040 020   E  \0  \0  \0  \0  \0  \0   4  \0      \0  \a  \0   (  \0
0000060 035  \0 034  \0 006  \0  \0  \0   4  \0  \0  \0   4 200 004  \b
0000100   4 200 004  \b 340  \0  \0  \0 340  \0  \0  \0 005  \0  \0  \0
..... 中间省略.....
```

利用这个指令,可以将 data file 或者是 binary file 的内容数据读出来,虽然读出的数值默认是使用非文本形式,也就是是 16 进位的数值来显示的,不过还是可以通过 `-t c` 的参数来将数据内的字符以 ASCII 类型的字符来显示,虽然对于一般使用者来说这个指令的用处可能不大,但是对于工程师来说这个指令可以将 binary file 的内容作一个大致的输出,它们可以看出其中的含义。

34.4.5 修改文件时间与创建新文件:touch

每个文件在 linux 下都会记录三个主要的时间参数,分别是:

- modification time(mtime):当该文件的"内容数据"变更时,就会更新这个时间,内容数据指的是文件的内容,而不是文件的属性。
- status time(ctime):当该文件的"状态(status)"改变时,就会更新这个时间,比如权限与属性被更改了,都会更新这个时间啊。
- access time(atime):当"该文件的内容被取用"时,就会更新这个读取时间(access),比如用 cat 去读取 ~/.bashrc,就会更新 atime。

先来看一看您自己的/etc/man.config 的时间参数:

```
[root@linux ~]# ls -l /etc/man.config
-rw-r--r--  1 root root 4506 Apr  8 19:11 /etc/man.config
[root@linux ~]# ls -l --time=atime /etc/man.config
-rw-r--r--  1 root root 4506 Jul 19 17:53 /etc/man.config
[root@linux ~]# ls -l --time=ctime /etc/man.config
-rw-r--r--  1 root root 4506 Jun 25 08:28 /etc/man.config
```

在默认的情况下,ls 显示出来的是该文件的 mtime,也就是这个文件的内容上次被更改的时间。

文件的时间是很重要的,如果文件的时间误判可能会造成某些程序无法顺利的运行,那该如何让该文件的时间变成"现在"的时刻呢?使用"touch"命令即可。

```
[root@linux ~]# touch [-acdmt] 文件
参数:
-a  :仅修改 access time;
-c  :仅修改时间,而不建立文件;
-d  :后面可以接日期,也可以使用--date="日期或时间"
-m  :仅修改 mtime;
-t  :后面可以接时间,格式为 [YYMMDDhhmm]

范例一:新建一个空的文件
[root@linux ~]# cd /tmp
[root@linux tmp]# touch testtouch
[root@linux tmp]# ls -l testtouch
-rw-r--r--  1 root root    0 Jul 19 20:49 testtouch
# 注意这个文件的大小是 0。默认状态下如果 touch 后面有接文件,
# 则该文件的三个时间(atime/ctime/mtime)都会更新为目前的时间。
# 若该文件不存在,则会主动的建立一个新的空的文件。

范例二:将 ~/.bashrc 复制成为 bashrc,假设复制完全的属性,检查其日期
[root@linux tmp]# cp ~/.bashrc bashrc
[root@linux tmp]# ll bashrc; ll --time=atime bashrc; ll --time=ctime bashrc
-rwxr-xr-x  1 root root 395 Jul  4 11:45 bashrc  <== 这是 mtime
-rwxr-xr-x  1 root root 395 Jul 19 20:44 bashrc  <== 这是 atime
-rwxr-xr-x  1 root root 395 Jul 19 20:53 bashrc  <== 这是 ctime
# 在这个案例当中,使用了; 这个指令分隔符号。
# 此外,ll 是 ls -l 的命令别名,这个也会在 Bash shell 中再次提及,
# 目前可以简单的想成,ll 就是 ls -l 的简写即可,至于; 则是同时执行两个指令,
# 且让两个指令"按顺序"执行的意思。上面的结果当中可以看到该文件变更的日期
# Jul 4 11:45,但是 atime 与 ctime 不一样
范例三:修改案例二的 bashrc 文件,将日期调整为两天前
```

```
[root@linux tmp]# touch -d "2 days ago" bashrc
[root@linux tmp]# ll bashrc; ll --time=atime bashrc; ll --time=ctime bashrc
-rwxr-xr-x  1 root root 395 Jul 17 21:02 bashrc
-rwxr-xr-x  1 root root 395 Jul 17 21:02 bashrc
-rwxr-xr-x  1 root root 395 Jul 19 21:02 bashrc
# 跟上个范例比较看看,本来是 19 日的变成了 17 日了(atime/mtime)。
# 不过 ctime 并没有跟着改变。
范例四:将上个范例的 bashrc 日期改为 2005/07/15 2:02
[root@linux tmp]# touch -t 0507150202 bashrc
[root@linux tmp]# ll bashrc; ll --time=atime bashrc; ll --time=ctime bashrc
-rwxr-xr-x  1 root root 395 Jul 15 02:02 bashrc
-rwxr-xr-x  1 root root 395 Jul 15 02:02 bashrc
-rwxr-xr-x  1 root root 395 Jul 19 21:05 bashrc
# 注意看看,日期在 atime 与 mtime 都改变了,但是 ctime 则是记录目前的时间。
```

通过 touch 命令可以轻易的修订文件的日期与时间,并且也可以建立一个空的文件。不过要注意的是,复制一个文件时即使复制所有的属性,也没有办法复制 ctime 这个属性。ctime 可以记录这个文件最近的状态(status)被改变的时间。无论如何还是要说明的是,平时看的文件属性中比较重要的还是属于那个 mtime。

无论如何,touch 命令最常被使用的情况是:

- 建立一个空的文件;
- 将某个文件日期修订为目前(mtime 与 atime)。

34.5 文件与目录的默认权限与隐藏权限

在 Linux 中每个文件都有若干个属性,包括(r,w,x)等基本属性以及是否为目录(d)与文件(-)或者是连接文件(l)等的属性。

除了基本 r, w, x 权限外, 在 Linux 的 ext2/3 文件系统下还可以设定其它的系统隐藏权限, 使用 chattr 来设定并以 lsattr 来查看,其中最重要的属性就是可以设定其不可修改的特性,连文件的所有者(owner)都不能进行修改,这个属性可是相当重要的,尤其是在安全机制上面(security)。

34.5.1 文件默认权限:umask

当建立一个新的文件或目录时,其默认属性与 umask 有关。基本上 umask 就是指定“目前使用者在建立文件或目录时候的权限默认值”,那么如何得知或设定 umask 呢? umask 的指定条件可以以下面的的方式来指定:

```
[root@linux ~]# umask
0022
[root@linux ~]# umask -S
u=rwx,g=rx,o=rx
```

查看的方式有两种,一种可以直接执行 umask,就可以看到数字型态的权限设定分数,一种则是加入 -S(Symbolic)参数,就会以符号类型的方式来显示出权限。

奇怪的是,怎么 umask 会有四组数字? 第一组是特殊权限用的,所以先看后面三组即可。

在默认权限的属性上,目录与文件是不一样的。由于用户不希望文件具有可执行的权力,默认情况下文件是没有可执行(x)权限的。

1、若使用者建立为“文件”则默认“没有可执行(x)权限”,也就是只有 rw 这两个项目,也就是最大为 666,默认属性如下:

```
-rw-rw-rw-
```

2、若使用者建立为“目录”,则由于 x 与是否可以进入此目录有关,因此默认为所有权限均开放,也就是为 777,默认属性如下:

```
drwxrwxrwx
```

umask 指定的是“该默认值需要减掉的权限”,因为 r、w、x 分别是 4、2、1,所以当要去掉可写入的权限,就是输入 2,而如果要去掉读取的权限,也就是 4,那么要去掉读与写的权限,也就是 6,而要去掉执行与写入的权限,也就是 3。

如果以上面的例子来说明,因为 umask 为 022,所以 user 并没有被去掉属性,不过 group 与 others 的属性被去掉了 2(也就是 w 属性),那么由于当使用者:

- 建立文件时:(-rw-rw-rw-)-(——w-w-)==> -rw-r--r--
- 建立目录时:(drwxrwxrwx)-(d——w-w-)==> drwxr-xr-x

```
[root@linux ~]# umask
0022
[root@linux ~]# touch test1
[root@linux ~]# mkdir test2
[root@linux ~]# ll
-rw-r--r--  1 root root    0 Jul 20 00:36 test1
drwxr-xr-x  2 root root 4096 Jul 20 00:36 test2
```

假如想要让与使用者同群组的人也可以存取文件,也就是说,假如 dmtsai 是 users 这个群组的人,而 dmtsai 文件希望让 users 同群组的人也可以存取,这也是常常被用在团队开发计划时会考虑到的权限问题。

在这样的情况下,umask 自然不能取消 group 的 w 权限,也就是说希望文件应该是 `-rw-rw-r--`,所以 umask 应该是要 002 才行(仅去掉 others 的 w 权限),设定 umask 时直接在 umask 后面输入 002 就可以。

```
[root@linux ~]# umask 002
[root@linux ~]# touch test3
[root@linux ~]# mkdir test4
[root@linux ~]# ll
-rw-rw-r--  1 root root    0 Jul 20 00:41 test3
drwxrwxr-x  2 root root 4096 Jul 20 00:41 test4
```

这个 umask 对于文件与目录的默认权限是很有关系的,这个概念可以用在任何服务器上面,尤其是未来在架设文件服务器(file server)时,比如 SAMBA Server 或 FTP Server 时,umask 都是很重要的观念,这牵涉到用户是否能够将文件进一步利用的问题。

例题四:假设 umask 为 003,这种 umask 情况下建立的文件与目录权限是什么?

在默认的情况中,root 的 umask 会去掉比较多的属性,root 的 umask 默认是 022,这是基于安全的考虑,至于一般身份使用者,通常它们的 umask 为 002,也就是保留同群组的写入权力。其实关于默认 umask 的设定可以参考/etc/bashrc 这个文件的内容,不过不建议修改该文件,可以参考 bash shell 提到的环境参数配置文件(~/.bashrc)的说明。

34.5.2 文件隐藏属性:chattr,lsattr

文件的隐藏属性确实对于系统有很大的帮助的,尤其是在系统安全(Security)上。

1、chattr(设定文件隐藏属性)

```
[root@linux ~]# chattr [+-=][ASacdistu] 文件或目录名称
参数:
+   :增加某一个特殊参数,其它原本存在参数则不动。
-   :移除某一个特殊参数,其它原本存在参数则不动。
=   :设定一定,且仅有后面接的参数
```

```
A  :当设定了 A 这个属性时,这个文件(或目录)的存取时间 atime(access)
     将不可被修改,可避免例如笔记本电脑容易有磁盘 I/O 错误的情况发生。
S  :这个功能有点类似 sync 的功能,就是会将数据同步写入磁盘当中,可以
     有效的避免数据丢失。
a  :当设定 a 之后,这个文件将只能增加数据而不能删除,只有 root 才能设定这个属性。
c  :这个属性设定之后,将会自动的将此文件"压缩",在读取的时候将会自动
     解压缩,但是在存储的时候,将会先进行压缩后再存储(对于大文件似乎有用)。
d  :当 dump(备份)程序被执行的时候,设定 d 属性将可使该文件(或目录)
     不具有 dump 功能。
i  :i 可以让一个文件"不能被删除、改名、设定连结也无法写入或新增数据",
     对于系统安全性有相当大的助益。
j  :当使用 ext3 这个文件系统格式时,设定 j 属性将会使文件在写入时先记录
     在 journal 中,但是当 filesystem 设定参数为 data=journalled 时,由于
     已经设定了日志了,所以这个属性无效。
s  :当文件设定了 s 参数时,它将会被完全的移除出这个硬盘空间。
u  :与 s 相反的,当使用 u 来设定文件时,则数据内容其实还存在磁盘中,
     可以使用来 undeletion。

这个属性设定上比较常见的是 a 与 i 的设定值,很多设定值必须要 root 才能够设定。
范例:
[root@linux ~]# cd /tmp
[root@linux tmp]# touch attrtest
[root@linux tmp]# chattr +i attrtest
[root@linux tmp]# rm attrtest
rm: remove write-protected regular empty file `attrtest'? y
rm: cannot remove `attrtest': Operation not permitted
# 连 root 也没有办法将这个文件删除。
[root@linux tmp]# chattr -i attrtest
```

这个指令是重要的,尤其是在系统的安全性上。这些属性是隐藏的,需要以 lsattr 才能看到,其中最重要的当属 +i 这个属性了,因为它可以让一个文件无法被更改,对于需要强烈的系统安全的人来说是相当的重要的,还有相当多的属性是需要 root 才能设定。

此外,如果是 log file 这种登录文件就更需要 +a 这个可以增加但是不能修改旧有的数据与删除的参数。

2、lsattr (显示文件隐藏属性)

```
[root@linux ~]# lsattr [-aR] 文件或目录
参数:
-a :将隐藏文件的属性也秀出来;
-R :连同子目录的数据也一并列出来。
范例:
[root@linux tmp]# chattr +aij attrtest
[root@linux tmp]# lsattr
----ia---j--- ./attrtest
```

使用 chattr 设定后,可以利用 lsattr 来查看隐藏的属性。不过这两个指令在使用上必须要特别小心,否则会造成很大的困扰。

34.6 文件特殊权限:SUID/SGID/Sticky Bit

查看/tmp 和/usr/bin/passwd 的权限时,会看到如下的结果:

```
[root@linux ~]# ls -ld /tmp ; ls -l /usr/bin/passwd
drwxrwxrwt  5 root root 4096  Jul 20 10:00 /tmp
-r-s--x--x  1 root root 18840 Mar  7 18:06 /usr/bin/passwd
```

34.6.1 Set UID

Linux 中的 s 与 t 的权限与系统的账号及系统的进程较为相关,它们可以让一般用户在执行某些程序时能暂时的具有该程序所有者的权限。举例来说,所有帐号的密码都记录在/etc/shadow 中,而/etc/shadow 这个文件的权限是“----------”,而且它的所有者是 root,在这个权限中仅有 root 可以“强制”存储,其它人是连看都没有办法看的。

当 s 这个标志出现在文件所有者的 x 权限上时,例如/usr/bin/passwd 这个文件的权限状态“-rwsr-xr-x”,此时就被称为 Set UID,简称为 SUID 的特殊权限。由上面的定义中知道,当 vbird 这个用户执行/usr/bin/passwd 时,它就会“暂时”的得到文件所有者 root 的权限,这里 UID 代表的是 User 的 ID,而 User 代表的则是这个程序(/usr/bin/passwd)的所有者(root)。

- SUID 仅可用在“二进制文件(binary file)”上,SUID 因为是程序在执行的过程中拥有文件所有者的权限,因此它仅可用于 binary file,不能够用在脚本(shell script),这是因为 shell script 只是将很多的 binary 执行文件调用进来执行,所以 SUID 的权限部分还是得要看 shell script 调用进来的程序的设定而不是 shell script 本身。当然 SUID 对于目录也是无效的。
- 执行者对于该程序需要具有 x 的可执行权限。
- s 权限仅在执行该程序的过程(run-time)中有效。
- 执行者将具有该程序所有者(owner)的权限。

在 Linux 系统中,所有账号的密码都记录在/etc/shadow 这个文件里,这个文件的权限为“---------- 1 root root”,那么 vbird 这个一般用户账号是无法访问这个文件的,但是每个用户都是可以自己修改自己的密码的,而这就是 s 权限的用途,也就是说 vbird 这个一般身份使用者可以修改/etc/shadow 这个文件内的密码。

通过上述说明,可以知道:

- vbird 对于/usr/bin/passwd 这个程序来说是具有 x 权限的,表示 vbird 能执行 passwd;
- passwd 的拥有者是 root 这个账号;
- vbird 执行 passwd 的过程中会“暂时”获得 root 的权限;
- /etc/shadow 可以被 vbird 所执行的 passwd 所修改。

而如果 vbird 执行“cat /etc/shadow”时,仍然是不能读取/etc/shadow 的,可以用下面的示意图来说明如下:

34.6.2 Set GID

进一步来说,如果 s 的权限是在 group 时就是 Set GID,简称为 SGID。与 SUID 不同的是,SGID 可以针对文件或目录来设置。

1、若针对文件,SGID 有如下的功能:

- SGID 对二进制程序有用;
- 程序执行者对于该程序来说,需具备 x 的权限;
- 如果 SGID 是设定在 binary file 上,则不论使用者是谁,在执行该程序的时候,它的有效用户组(effective group)将会获得该程序的用户组(group id)的支持。

举例来说,先查看具有 SGID 权限的文件:

```
[root@linux ~]# ls -l /usr/bin/locate
-rwx--s--x  1  root  slocate    23856  Mar 15 2007   /usr/bin/locate
```

而/usr/bin/locate 这个程序可以去查询/var/lib/mlocate/mlocate.db 这个文件的内容,mlocate.db 的权限如下:

```
[root@linux ~]# ll /var/lib/mlocate/mlocate.db
-rw-r-----  1  root  slocate  3175776  Sep 28 04:02  /var/lib/mlocate.db
```

与 SGID 非常类似，若使用 vbird 这个账号去执行 locate 时，vbird 将会取得 slocate 用户组的支持，于是就能够读取 mlocate.db。

2、若针对目录，SGID 有如下的功能：

- 如果 SGID 是设定在 A 目录上，则在该 A 目录内所建立的文件或目录的 group 将会是这个 A 目录的 group。
- 用户若对于此目录具有 r 与 x 的权限时，该用户能够进入此目录；
- 用户在此目录下的有效用户组(effective group)将会变成该目录的用户组；
- 若用户在此目录下具有 w 的权限(可以新建文件)，则用户所创建的新文件的用户组与此目录的用户组相同。

一般来说，SGID 应该是比较多用在特定的多人团队的项目开发上，因为会涉及到用户组权限的问题。

34.6.3 Sticky Bit

Sticky Bit(SBIT)目前只针对目录有效，对于文件已经没有效果了。SBIT 对于目录的作用是：

- 当用户对于此目录具有 w、x 权限，就具有写入的权限。
- 在具有 SBit 的目录下，使用者若在该目录下具有 w 及 x 权限，则当使用者在该目录下建立文件或目录时，只有文件所有者与 root 才有权力删除。
- 换句话说，当甲这个用户在 A 目录下是拥有 group 或者是 other 的项目，并且拥有 w 的权限，这表示“甲使用者对该目录内任何人建立的目录或文件均可进行删除/更名/搬移等动作。”不过，如果将 A 目录加上了 Sticky Bit 的权限项目时，则甲只能够针对自己建立的文件或目录进行删除/更名/移动等动作，而无法删除别人的文件。

举例来说，/tmp 本身的权限是 `drwxrwxrwt`，在这样的权限下任何人都可以在/tmp 内新增、修改文件，但仅有该文件/目录建立者与 root 能够删除自己的目录或文件，这个特性也是挺重要的，可以这样做个简单的测试：

(1)以 root 登录系统，并且进入/tmp 中；

(2)执行 `touch test`，并且更改 test 权限成为 777；

(3)以一般使用者登录，并进入/tmp；

(4)尝试删除 test 这个文件。

34.6.4 SUID/SGID/SBIT 权限设定

数字型态个更改权限方式为“三个数字”的组合，那么如果在这三个数字之前再加上一个数字，那最前的面数字就代表这几个属性。(注：通常我们使用 `chmod xyz filename` 的方式来设定 filename 的属性时，则是假设没有 SUID、SGID 及 Sticky Bit)

- 4 为 SUID；
- 2 为 SGID；
- 1 为 Sticky Bit。

假设要将一个文件属性改为 `-rwsr-xr-x` 时，由于 s 在用户权限中，所以是 SUID，因此在原先的 755 之前还要加上 4 ，也就是使用：

```
chmod 4755 filename
```

来设定。

此外还有大 S 与大 T 的产生，参考底下的范例，注意底下的范例只是练习，所以使用同一个文件来设定，首先必须了解 SUID 不是用在目录上，而 SBIT 不是用在文件上。

```
[root@linux ~]# cd /tmp
[root@linux tmp]# touch test
```

```
[root@linux tmp]# chmod 4755 test; ls -l test
-rwsr-xr-x  1 root root 0 Jul 20 11:27 test
[root@linux tmp]# chmod 6755 test; ls -l test
-rwsr-sr-x  1 root root 0 Jul 20 11:27 test
[root@linux tmp]# chmod 1755 test; ls -l test
-rwxr-xr-t  1 root root 0 Jul 20 11:27 test
[root@linux tmp]# chmod 7666 test; ls -l test
-rwSrwSrwT  1 root root 0 Jul 20 11:27 test
# 这个例子就要特别小心,怎么会出现大写的 S 与 T? 不都是小写的吗?
# 因为 s 与 t 都是取代 x 这个参数的,但是有没有发现我们执行的是 7666,
# 也就是说,user、group 以及 others 都没有 x 这个可执行的标志(因为 666),
# 所以这个 S、T 代表的就是"空的"。
# SUID 是表示"该文件在执行的时候具有文件所有者的权限",但是文件
# 所有者都无法执行了,哪里来的权限给其它人使用,当然就是空的。
```

34.7 查看文件类型:file

如果想要知道某个文件的基本数据,例如是属于 ASCII 还是 data 文件,或者是 binary,且其中有没有使用到动态函数库(share library)等的信息,可以利用 file 指令来查看,举例来说:

```
[root@linux ~]# file ~/.bashrc
/root/.bashrc: ASCII text  <== ASCII 纯文本文档
[root@linux ~]# file /usr/bin/passwd
/usr/bin/passwd: setuid ELF 32-bit LSB executable, Intel 80386, version 1
(SYSV), for GNU/Linux 2.2.5, dynamically linked (uses shared libs), stripped
[root@linux ~]# file /var/lib/slocate/slocate.db
/var/lib/slocate/slocate.db: data  <== data 文件
```

通过 file 指令可以简单的判断文件的格式。

34.8 命令与文件的查询

我们经常需要知道哪个文件放在哪里才能对该文件进行一些修改或维护等操作,虽然文件名通常不变,但是不同的 Linux distribution 放置的目录可能不同,就需要使用 Linux 的搜索系统来将文件的完整文件名找出来,通常 find 不很常用,除了速度慢之外也很消耗硬盘。通常我们都是先使用 whereis 或者是 locate 来检查,如果真的找不到了才以 find 来搜索,whereis 与 locate 是利用数据库来搜索数据,所以相当的快速,而且并没有实际的搜索硬盘,比较省时间。

34.8.1 which(查找"执行文件")

```
[root@linux ~]# which [-a] command
参数:
-a :将所有可以找到的指令均列出,而不止第一个被找到的指令名称
范例:
[root@linux ~]# which passwd
/usr/bin/passwd
[root@linux ~]# which traceroute -a
/usr/sbin/traceroute
/bin/traceroute
```

这个指令是根据"PATH"这个环境变量所规范的路径去搜索"执行文件"的文件名,所以重点是找出"执行文件"且 which 后面接的是"完整文件名"。若加上 -a 参数,则可以列出所有的可以找到的同名执行文件而非仅显示第一个而已。

34.8.2 whereis(查找特定文件)

```
[root@linux ~]# whereis [-bmsu] 文件或目录名
参数:
-b   :只找 binary 的文件;
-m   :只找在说明文件 manual 路径下的文件;
-s   :只找 source 来源文件;
-u   :没有说明文件的文件。
范例:
[root@linux ~]# whereis passwd
passwd: /usr/bin/passwd /etc/passwd /etc/passwd.OLD
/usr/share/man/man1/passwd.1.gz /usr/share/man/man5/passwd.5.gz
# 任何与 passwd 有关的文件名都会被列出来。
[root@linux ~]# whereis -b passwd
passwd: /usr/bin/passwd /etc/passwd /etc/passwd.OLD
[root@linux ~]# whereis -m passwd
passwd: /usr/share/man/man1/passwd.1.gz /usr/share/man/man5/passwd.5.gz
```

find 是很强大的搜索指令,但 find 是直接搜索硬盘,所以时间消耗很大。此时 whereis 就相当的好用了,而且 whereis 可以加入参数来寻找相关的数据,例如如果是要找可执行文件(binary),那么加上 -b 就可以。例如上面的范例针对 passwd 程序来说明,如果不加任何参数的话,就将所有的数据列出来。

Linux 系统会将系统内的所有文件都记录在一个数据库文件里面,当使用 whereis 或者是 locate 时,都会以此数据库文件的内容为准,因此有的时候还会发现使用这两个执行文件时会找到已经被删除的文件,而且也找不到最新的刚刚建立的文件,这就是因为这两个指令是由数据库当中的结果去搜索文件的所在位置。

基本上 Linux 每天会针对主机上所有文件的位置进行搜索数据库的更新,执行 updatedb 命令会去读取/etc/updatedb.conf 这个配置文件的设置,然后再去硬盘里面进行查找文件名的操作并更新/var/lib/mlocate 内的数据库文件,可以在/etc/cron.daily/slocate.cron 这个文件找到相关的机制,也可以直接使用/usr/bin/updatedb 来更新数据库文件。

34.8.3 locate

```
[root@linux ~]# locate filename
[root@linux ~]# locate passwd
/lib/security/pam_passwdqc.so
/lib/security/pam_unix_passwd.so
/usr/lib/kde3/kded_kpasswdserver.so
/usr/lib/kde3/kded_kpasswdserver.la
....... 中间省略.......
```

locate 的使用更简单,直接在后面输入“文件的部分名称”后就能够得到结果。继续上面的例子,输入 locate passwd,那么在完整文件名(包含路径名称)中只要有 passwd 就会被显示出来,如果忘记某个文件的完整文件名时,这也是个很方便好用的指令。

但是 locate 还是有使用上的限制,之所以使用 locate 查找数据的时候特别快是因为 locate 查找的数据是由“已建立的数据库/var/lib/slocate/”里面的数据所搜索到的,所以不用直接在去硬盘当中存取数据。

locate 也有一定的限制,因为 locate 是通过数据库来搜索,而数据库的建立默认是在每天执行一次(每个 distribution 都不同),所以对于新建立起来的文件而在数据库更新之前搜索该文件的话,locate 会“找不到该文件”。

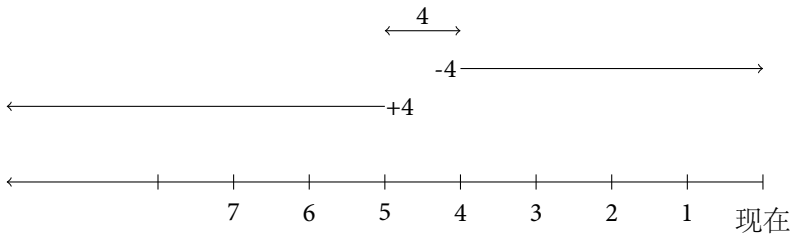
使用 locate 时,可以自己选择需要建立文件数据库的目录,可以在/etc/updatedb.conf 文件内设定,建议使用默认值即可。不过在/etc/updatedb.conf 里面可以把 DAILY_UPDATE=no 改成 DAILY_UPDATE=yes,当然也可以自行手动执行 updatedb 即可。

34.8.4 find

```
[root@linux ~]# find [PATH] [option] [action]
1、与时间有关的参数:
   -atime n :n 为数字,意义为在 n 天之前的“一天之内”被 access 过的文件;
   -ctime n :n 为数字,意义为在 n 天之前的“一天之内”被 change 过状态的文件;
   -mtime n :n 为数字,意义为在 n 天之前的“一天之内”被 modification 过的文件;
   -newer file :file 为一个存在的文件,意思是说只要文件比 file 还要新就会被列出来。
范例一:将过去系统上面 24 小时内有更动过内容(mtime)的文件列出
[root@linux ~]# find / -mtime 0
# 那个 0 是重点,0 代表目前的时间,所以从现在开始到 24 小时前,
# 有变动过内容的文件都会被列出来。那如果是三天前的 24 小时内?
# find / -mtime 3,意思是说今天之前的 3*24 ~ 4*24 小时之间
# 有变动过的文件都被列出的意思,同时-atime 与-ctime 的用法相同。
范例二:查找/etc 下的文件,如果文件日期比/etc/passwd 新就列出
[root@linux ~]# find /etc -newer /etc/passwd
# -newer 用在分辨两个文件之间的新旧关系是很有用。
```

如果想找出一天内被改动过的文件名,可以使用上述的做法,即执行 `find / -mtime 0`。

但如果想要找出“4 天内被改动过的文件名”,就要使用 `fine / -mtime -4`,如果想要找出“4 天前的那一天”,就需要用 `find / -mtime 4`,这里有没有加上“+、-”差别很大,可以用如下的图示来说明:



图中最右边为当前的时间,越往左边则代表越早之前的时间,由图中可以看出:

- +4 代表大于等于 5 天前的文件名:ex>find / -mtime +4;
- −4 代表小于等于 4 天内的文件名:ex>find / -mtime -4;
- 4 代表 4~5 这一天的文件名:ex>find / -mtime 4。

```
2、与使用者或群组名称有关的参数:
   -uid n :n 为数字,这个数字是使用者的帐号 ID,也就是 UID,UID 是记录在
             /etc/passwd 里面与帐号名称对应的数字。
   -gid n :n 为数字,这个数字是群组名称的 ID,也就是 GID,GID 记录在/etc/group。
   -user name :name 为使用者帐号名称,例如 dmtsai;
   -group name:name 为群组名称,例如 users;
   -nouser    :查找文件的所有者不存在于/etc/passwd 的人。
   -nogroup   :查找文件的拥有群组不存在于/etc/group 的文件,当用户自行安装
               软件时,很可能该软件的属性当中并没有文件所有者,在这个时候
               就可以使用-nouser 与-nogroup 搜索。
范例三:搜索/home 下属于 dmtsai 的文件
[root@linux ~]# find /home -user dmtsai
# 当要找出任何一个使用者在系统当中的所有文件时,
# 就可以利用这个指令将属于某个使用者的所有文件都找出来。
范例四:搜索系统中不属于任何人的文件
[root@linux ~]# find / -nouser
# 通过这个指令,可以轻易的就找出那些不太正常的文件。如果有找到不属于系统任何人的文件时,不要太紧张,
# 那有时候是正常的,尤其是曾经以源代码自行编译软件时。
```

如果系统管理员需要知道某个用户在系统下创建了什么,可以使用上述的参数。对于 `-nouser` 或 `-nogroup` 的参数,除了用户自行从网络上下载文件时会发生外,如果系统中某个账号删除了,但是该账号在系统内已经创建过多个文件时就有可能发生找不到文件的用户的情况,此时就可以使用 `-nouser` 来找出这些文件。

3、与文件权限及名称有关的参数:
-name filename:搜索文件名称为 filename 的文件;
-size [+-]SIZE:搜索比 SIZE 还要大(+)或小(-)的文件。这个 SIZE 的规格有:
c:代表 byte,k:代表 1024bytes。所以要找比 50KB 还要大的文件,
就是"-size +50k"
-type TYPE:搜索文件的类型为 TYPE 的,类型主要有:
一般正规文件(f);
装置文件(b,c);
目录(d);
连接文件(l);
socket(s)及 FIFO(p)等属性。
-perm mode:搜索文件属性"刚好等于"mode 的文件,这个 mode 为类似 chmod
的属性值,举例来说,-rwsr-xr-x 的属性为 4755。
-perm -mode :搜索文件属性"必须要全部囊括 mode 的属性"的文件,举例来说,
要搜索-rwxr--r--,也就是 0744 的文件,使用-perm -0744,
当一个文件的属性为-rwsr-xr-x,也就是 4755 时,也会被列出来,
因为-rwsr-xr-x 的属性已经囊括了-rwxr--r--的属性了。
-perm +mode :搜索文件属性"包含任一 mode 的属性"的文件,举例来说,我们搜索
-rwxr-xr-x ,也就是 -perm +755 时,但一个文件属性为 -rw-------
也会被列出来,因为它有 -rw.... 的属性存在!
4、额外可进行的动作:
-exec command:command 为其它指令,-exec 后面可再接额外的指令来处理搜索到的结果。
-print :将结果打印到显示器上,这个动作是默认动作。
范例五:找出文件名为 passwd 这个文件

```
[root@linux ~]# find / -name passwd
# 利用这个-name 可以搜索文件名
```

范例六:搜索文件属性为 f(一般文件)的文件

```
[root@linux ~]# find /home -type f
# -type 属性也很有帮助,尤其是要找出那些怪异的文件,
# 例如 socket 与 FIFO 文件,可以用 find /var -type p 或-type s。
```

范例七:搜索文件当中含有 SGID/SUID/SBIT 的属性

```
[root@linux ~]# find / -perm +7000
# 所谓的 7000 就是 ---s--s--t,那么只要含有 s 或 t 的就列出,
# 所以当然要使用 +7000,使用-7000 表示要含有---s--s--t 的所有三个权限,
```

范例八:将上个范例找到的文件使用 ls -l 列出来

```
[root@linux ~]# find / -perm +7000 -exec ls -l  \;
# 注意到,-exec 后面的 ls -l 就是额外的指令,
# 而代表的是"由 find 找到的内容"的意思,所以-exec ls -l
# 就是将前面找到的那些文件以 ls -l 列出长的数据。
# \; 则是表示-exec 的指令到此为止的意思,也就是说整个指令其实只有在
# -exec(里面就是指令执行)\;
# 也就是说,-exec 最后一定要以 \; 结束才行。
```

范例九:找出系统中大于 1MB 的文件

```
[root@linux ~]# find / -size +1000k
```

上述范例中的 -perm 参数的重点在于找出特殊权限的文件,因为 SUID 与 SGID 都可以设置在二进制程序上,假设想要将/bin、/sbin 这两个目录下主要具有 SUID 或 SGID 的文件就列出来,就可以执行下面的命令:

```
[root@linux ~]# find /bin /sbin -perm +6000
```

因为 SUID 是 4,SGID 是 2,因此可用 6000 来处理这个权限。

注意,find 后面可以接多个目录来进行查找,另外 find 本来就会查找子目录,这个也要特别注意。而 find 的特殊功能是可以进行额外的动作(action),比如:

```
[root@linux ~]# find / -perm +7000 -exec ls -l \;
```

该范例中特殊的地方有“”与“\;”以及“-exec”关键字,分别介绍如下:

- 代表的是“由 find 找到的内容”,find 的结果会放到位置中;
- -exec 一直到“\;”是关键字,代表 find 额外命令的开始(-exec)到结束(\;),在这中间的就是 find 命令内的额外命令。在本例中就是“ls -l ”。
- ; 在 bash 环境中是有特殊意义的,因此利用反斜杠来转义。

因此如果要查找的文件是具有特殊属性的,例如 SUID、SGID、文件所有者、文件大小等,这些条件 locate 是无法达到,那么使用 find 是一个不错的主意,它可以根据不同的参数来给予文件的搜索功能,例如要查找一个文件名为 httpd.conf 的文件,知道它应该是在/etc 下,那么就可以使用 `find /etc -name httpd.conf`。

另外,find 还可以利用通配符来查找文件,如果记得有一个文件文件名包含了 httpd,但是不知道全名,就可以用万用字符,如上以:`find /etc -name '*httpd*'` 就可将文件名含有 httpd 的文件都列出来,不过由于 find 在查找数据的时后相当的消耗硬盘,所以一般情况下不使用 find,而是用 whereis 与 locate 来代替。但不管怎么说,由于 find 可以指定查找的目录(连同子目录),而且可以利用额外的参数,因此在查找寻特殊的文件属性以及特殊的文件权限(SUID/SGID 等)时 find 是相当有用的工具程序之一。

34.9 权限与命令的关系

权限对于用户账号来说是非常重要的,它可以限制用户能不能读取/新建/删除/修改文件或目录,下面分别说明命令在什么样的权限下才能运行:

1、让用户能进入某目录成为“可工作目录”的基本权限。

- 可使用的命令:例如 cd 等切换工作目录的命令;
- 目录所需权限:用户对这个目录至少需要具有 x 的权限;
- 额外需求:如果用户想要在这个目录内利用 ls 查阅文件名,则用户对此目录还需要 r 的权限。

2、让用户在某个目录内读取一个文件的基本权限。

- 可使用的命令:例如 cat、more、less 等;
- 目录所需权限:用户对这个目录至少需要具有 x 权限;
- 文件所需权限:用户对文件至少需要具有 r 的权限。

3、让用户可以修改一个文件的基本权限。

- 可使用的命令:例如 vi 等;
- 目录所需权限:用户在该文件所在的目录至少要有 x 权限;
- 文件所需权限:用户对该文件至少要有 r、w 权限。

4、让用户可以创建一个文件的基本权限。

- 目录所需权限:用户在该目录要具有 w、x 的权限,重点在 w。

5、用户进入某目录并执行该目录下的某个命令的基本权限。

- 目录所需权限:用户在该目录至少要有 x 的权限。
- 文件所需权限:用户在该文件至少需要有 x 的权限。

第 35 章

文件系统操作

35.1 查看

35.2 cat

35.3 ls

35.4 mv

35.5 cp

35.6 rm

35.7 modify

35.8 tar

35.9 backup

35.10 dump

35.11 fdisk

35.12 parted

35.13 mount

35.14 umount

第 36 章

Linux 文件与文件系统的压缩与打包

文件的压缩与打包

在 Linux 下有相当多的压缩指令可以运作，这些压缩指令可以让我们更方便从网络上面下载大型的文件。此外，我们知道在 Linux 下的扩展名是没有什么很特殊的意义的，不过针对这些压缩指令所做出来的压缩文件，为了方便记忆还是会有一些特殊的命名方式。

1. 压缩文件的用途与技术
2. Linux 系统常见的压缩指令
   2.1 compress
   2.2 gzip, zcat
   2.3 bzip2, bzcat
   2.4 tar
   2.5 dd
   2.6 cpio

压缩文件的用途与技术

大型的文件通过所谓的文件压缩技术之后，可以将硬盘使用量降低并达到减低文件容量的效果。此外，有的压缩程序还可以进行容量限制，使一个大型文件可以分割成为数个小型文件。

目前的计算机系统中都是使用所谓的 bytes 单位来计量的，不过事实上，计算机最小的计量单位应该是 bits，此外 1 byte = 8 bits。

假设一个 byte 可以看成下面的模样：

Tips：

由于 1 byte = 8 bits，所以每个 byte 当中会有 8 个空格，而每个空格可以是 0 或 1。而由于我们记录数字是 1，考虑计算机所谓的二进位，1 会在最右边占据 1 个 bit，而其他的 7 个 bits 将会自动的被填上 0。

其实在这样的例子中，那 7 个 bits 应该是“空的”才对。

不过，为了要满足目前我们的操作系统数据的存取，所以就会将该数据转为 byte 的型态来记录了。而一些聪明的计算机工程师就利用一些复杂的计算方式，将这些没有使用到的空间“丢”出来，以让文件占用的空间变小，这就是压缩的技术。

简单的说，可以将它想成，其实文件里面有相当多的“空间”存在，并不是完全填满的，而“压缩”的技术就是将这些“空间”填满以让整个文件占用的容量下降。不过，这些“压缩过的文件”并无法直接被我们的操作系统所使用的，因此若要使用这些被压缩过的文件数据，则必须将其“还原”回来未压缩前的模样，那就是所谓的“解压缩”。

而至于压缩前与压缩后的文件所占用的硬盘空间大小，就可以被称为是“压缩比”。

这个“压缩”与“解压缩”的动作有什么好处呢？

最大的好处就是压缩过的文件容量变小了，所以硬盘容量无形之中就可以容纳更多的数据，此外，在一些网络数据的传输中，也会由于数据量的降低，好让网络带宽可以用来作更多的工作。而不是老是卡在一些大型的文件上面，目前很多的网站也是利用文件压缩的技术来进行数据的传送，好让网站的可利用率上升。

Tips：

压缩技术可以让网站上面“看的到的数据”在经过网络传输时,使用的是“压缩过的数据”,等到这些压缩过的数据到达计算机主机时,再进行解压缩。

由于目前的计算机运算速度相当的快速,因此其实在网页浏览的时候,时间都是花在“数据的传输”上面,而不是 CPU 的运算,如此一来,由于压缩过的数据量降低了,自然传送的速度就会加快不少。

Linux 系统常见的压缩指令

压缩过的文件具有节省带宽、节省硬盘空间等优点,并且还方便携带。而这些被压缩过的文件,通常其扩展名都是 *.tar, *.tar.gz, *.tgz, *.gz, *.Z, *.bz2 等,为什么要设定这些压缩文件扩展名为这样的模样呢?

这是因为在 Linux 上面压缩的指令相当的多,并且这些压缩指令可能无法针对每种压缩文件都可以解的开,毕竟目前的压缩技术五花八门,每种压缩计算的方法都不是完全相同的,所以当得到某个压缩文件时,自然就需要知道压缩他的是那个指令啦,好用来对照著解压缩,也就是说,虽然 Linux 文件的属性基本上是与文件名没有绝对关系的,能不能执行与它的文件属性有关而已,与文件名的关系很小。

适当的文件名称扩展名还是必要的,因此目前就有一些常常见到的压缩文件的扩展名。

我们仅列出常见的几样在下面,给大家权做参考之用:

*.Z	compress 程序压缩的文件;
*.bz2	bzip2 程序压缩的文件;
*.gz	gzip 程序压缩的文件;
*.tar	tar 程序打包的数据,并没有压缩过;
*.tar.gz	tar 程序打包的文件,其中并且经过 gzip 的压缩

目前常见的压缩程序主要就是如同上面提到的扩展名对应的那些指令,最早期的要算是 compress,不过这个 compress 指令目前已经不再是预设的压缩软件了。而后 GNU 计划开发出新一代的压缩指令 gzip(GNU zip)用来取代 compress 压缩指令,再来还有 bzip2 这个压缩比更好的压缩指令。

不过,这些指令通常仅能针对一个文件来压缩与解压缩,如此一来,每次压缩与解压缩都要一大堆文件,岂不烦人?此时“打包软件”就显的很重要。

在 Unix-Like 操作系统当中,tar 程序可以将很多文件“打包”成为一个文件,甚至是目录也可以这么处理。

不过,单纯的 tar 功能仅是“打包”而已,也就是将很多文件集结成为一个文件,事实上,它并没有提供压缩的功能,后来 GNU 计划中将整个 tar 与压缩的功能结合在一起,如此一来提供使用者更方便并且更强大的压缩与打包功能。

1、compress

```
[root@linux ~]# compress [-dcr] 文件或目录

参数 :
-d:用来解压缩的参数;
-r:可以连同目录下的文件也同时给予压缩;
-c:将压缩数据输出成为 standard output(输出到显示器)。

范例 :
范例一 : 将/etc/man.config 复制到/tmp,并加以压缩
[root@linux ~]# cd /tmp
[root@linux tmp]# cp /etc/man.config .
[root@linux tmp]# compress man.config
[root@linux tmp]# ls -l
-rw-r--r--  1 root root 2605 Jul 27 11:43 man.config.Z

范例二 : 将压缩文件解开
[root@linux tmp]# compress -d man.config.Z
```

```
范例三 ： 将 man.config 压缩成另外一个文件来备份
[root@linux tmp]# compress -c man.config > man.config.back.Z
[root@linux tmp]# ll man.config*
-rw-r--r--  1 root root 4506 Jul 27 11:43 man.config
-rw-r--r--  1 root root 2605 Jul 27 11:46 man.config.back.Z
# 使用-c 参数会将压缩过程的数据输出到显示器上，而不是写入成为 file.Z 文件。
所以可以通过数据流重导向的方法将数据输出成为另一个文件名。
```

这是用来压缩与解压缩扩展名为 *.Z 的指令，看到 *.Z 的文件时就应该要知道它是经由 compress 这个程序压缩的，这是最简单的压缩指令。

不过，使用的时候需要特别留意的是，当以 compress 压缩之后，如果没有其他的参数，那么原本的文件就会被后来的 *.Z 所取代。

以上面的案例来说明：原本压缩的文件为 man.config，那么当压缩完成之后，将只会剩下 man.config.Z 这个经过压缩的文件，那么解压缩则是将 man.config.Z 解压缩成 man.config。

解压缩除了可以使用 compress -d 这个参数之外，也可以直接使用意思相同的 uncompress。

另外，如果不想让原本的文件被更名成为 *.Z，而想制作出另外的一个文件名时，就可以利用数据流重导向，也就是那个大于 (>) 的符号，将原本应该在显示器上面出现的数据把它存储到其他文件去。当然，这要加上 -c 的参数才行。

此外，compress 已经很少人在使用了，因为这个程序无法解压 *.gz 的文件，而 gzip 则可以解决 *.Z 的文件，所以，如果 distribution 上面没有 compress 的话，也没有关系。

Tips：

compress 使用的频率越来越低了，现在已经集成到 ucompress 软件内，可以参考 RPM 的方式来安装。

2、gzip, zcat

```
[root@linux ~]# gzip [-cdt#] 文件名
[root@linux ~]# zcat 文件名.gz
参数 ：
-c  ： 将压缩的数据输出到显示器上，可通过数据流重导向来处理；
-d  ： 解压缩的参数；
-t  ： 可以用来检验一个压缩文件的一致性，看看文件有无错误；
-#  ： 压缩等级，-1 最快，但是压缩比最差，-9 最慢，但是压缩比最好，预设是 -6。

范例 ：
范例一 ： 将/etc/man.config 复制到 /tmp，并且以 gzip 压缩
[root@linux ~]# cd /tmp
[root@linux tmp]# cp /etc/man.config .
[root@linux tmp]# gzip man.config
# 此时 man.config 会变成 man.config.gz。

范例二 ： 将范例一的文件内容读出来。
[root@linux tmp]# zcat man.config.gz
# 此时显示器上会显示 man.config.gz 解压缩之后的文件内容。

范例三 ： 将范例一的文件解压缩
[root@linux tmp]# gzip -d man.config.gz

范例四 ： 将范例三解开的 man.config 用最佳的压缩比压缩，并保留原本的文件
[root@linux tmp]# gzip -9 -c man.config > man.config.gz
```

gzip 是用来压缩与解压缩扩展名为 *.gz 的指令，看到 *.gz 的文件时就应该要知道它是经由 gzip 这个程序压缩的。

另外，gzip 也提供压缩比的服务，-1 是最差的压缩比，但是压缩速度最快，而 -9 虽然可以达到较佳的压缩比（经过压缩之后，文件比较小一些），但是却会损失一些速度。预设是 -6 这个数值，gzip 也是相当常使用的一个压缩指令。

至于 zcat 则是用来读取压缩文件数据内容的指令，假如刚刚压缩的文件是一个文字文件，使用 cat 读取文字文件，使用 zcat 读取压缩文件。

由于 gzip 这个压缩指令主要想要用来取代 compress 的，所以 compress 的压缩文件也可以使用 gzip 来解开。同时，zcat 这个指令可以同时读取 compress 与 gzip 的压缩文件。

3、bzip2, bzcat

```
[root@linux ~]# bzip2 [-cdz] 文件名
[root@linux ~]# bzcat 文件名.bz2
参数 ：
-c  ：将压缩的过程产生的数据输出到显示器上
-d  ：解压缩的参数
-z  ：压缩的参数
-#  ：与 gzip 同样的，都是在计算压缩比的参数，-9 最佳，-1 最快。

范例 ：
范例一 ：将/tmp/man.config 以 bzip2 压缩
[root@linux tmp]# bzip2 -z man.config
# 此时 man.config 会变成 man.config.bz2。

范例二 ：将范例一的文件内容读出来。
[root@linux tmp]# bzcat man.config.bz2
# 此时显示器上会显示 man.config.bz2 解压缩之后的文件内容。

范例三 ：将范例一的文件解压缩
[root@linux tmp]# bzip2 -d man.config.bz2

范例四 ：将范例三解开的 man.config 用最佳的压缩比压缩并保留原本的文件
[root@linux tmp]# bzip2 -9 -c man.config > man.config.bz2
```

使用 compress 扩展名自动建立为.Z，使用 gzip 扩展名自动建立为.gz。

这里的 bzip2 则是自动的将扩展名建置为.bz2，所以当使用具有压缩功能的 bzip2 -z 时，那么 man.config 就会自动的变成 man.config.bz2 这个文件名。

可以使用简便的 bzcat 这个指令来读取压缩文件内容，例如上面的例子中可以使用 bzcat man.config.bz2 来读取数据而不需要解压缩。

此外，当要解开一个压缩文件时，这个文件的名称为.bz, .bz2, .tbz, .tbz2 等，那么就可以尝试使用 bzip2 来解压缩。

当然也可以使用 bunzip2 这个指令来取代 bzip2 -d。

4、tar

```
[root@linux ~]# tar [-cxtzjvfpPN] 文件与目录 ...
参数 ：
-c  ：建立一个压缩文件的参数指令(create)；
-x  ：解开一个压缩文件的参数指令；
-t  ：查看 tarfile 里面的文件。
      特别注意，在参数中 c/x/t 仅能存在一个，不可同时存在，因为不可能同时压缩与解压缩。
-z  ：是否同时具有 gzip 的属性？也就是否需要用 gzip 压缩？
-j  ：是否同时具有 bzip2 的属性？也就是否需要用 bzip2 压缩？
-v  ：压缩的过程中显示文件！这个常用，但不建议用在背景执行过程！
-f  ：使用文件名，注意在 f 之后要立即接文件名，不要再加参数。
      例如使用 tar -zcvfP tfile sfile 就是错误的写法，要写成 tar -zcvPf tfile sfile 才行。
-p  ：使用原文件的原来属性(属性不会依据使用者而变)。
-P  ：可以使用绝对路径来压缩
-N  ：比后面接的日期 (yyyy/mm/dd) 还要新的才会被打包进新建的文件中
--exclude FILE：在压缩的过程中，不要将 FILE 打包。

范例 ：
范例一 ：将整个/etc 目录下的文件全部打包成为/tmp/etc.tar。
```

```
[root@linux ~]# tar -cvf /tmp/etc.tar /etc  <== 仅打包,不压缩;
[root@linux ~]# tar -zcvf /tmp/etc.tar.gz /etc  <== 打包后,以 gzip 压缩;
[root@linux ~]# tar -jcvf /tmp/etc.tar.bz2 /etc  <== 打包后,以 bzip2 压缩。
# 特别注意,在参数 f 之后的文件文件名是自己取的,我们习惯上都用.tar 来作为辨识。
# 如果加 z 参数,则以.tar.gz 或.tgz 来代表 gzip 压缩过的 tar file。
# 如果加 j 参数,则以.tar.bz2 来作为扩展名。
# 上述指令在执行的时候会显示一个警告信息 
#  tar: Removing leading `/' from member names,那是关于绝对路径的特殊设定。

范例二  查阅上述/tmp/etc.tar.gz 文件内有哪些文件?
[root@linux ~]# tar -ztvf /tmp/etc.tar.gz
# 由于我们使用 gzip 压缩,所以要查阅该 tar file 内的文件时,
# 就得要加上 z 这个参数了,这很重要的

范例三  将 /tmp/etc.tar.gz 文件解压缩在 /usr/local/src 下
[root@linux ~]# cd /usr/local/src
[root@linux src]# tar -zxvf /tmp/etc.tar.gz
# 在预设的情况下,我们可以将压缩文件在任何地方解开,以这个范例来说,
# 我将工作目录变换到/usr/local/src 下,并且解开 /tmp/etc.tar.gz ,
# 则解开的目录会在/usr/local/src/etc,另外如果进入 /usr/local/src/etc
# 则会发现,该目录下的文件属性与/etc/可能会有所不同。

范例四  在/tmp 下,只想要将/tmp/etc.tar.gz 内的 etc/passwd 解开而已
[root@linux ~]# cd /tmp
[root@linux tmp]# tar -zxvf /tmp/etc.tar.gz etc/passwd
# 可以通过 tar -ztvf 来查阅 tarfile 内的文件名称,如果单只要一个文件,
# 就可以通过这个方式来下达,注意 etc.tar.gz 内的根目录/是被拿掉。

范例五  将/etc/内的所有文件备份下来,并且保存其权限。
[root@linux ~]# tar -zcvpf /tmp/etc.tar.gz /etc
# 这个-p 的属性是很重要的,尤其是当要保留原本文件的属性时。

范例六  在/home 当中,比 2005/06/01 新的文件才备份
[root@linux ~]# tar -N '2005/06/01' -zcvf home.tar.gz /home

范例七  要备份/home,/etc,但不要/home/dmtsai
[root@linux ~]# tar --exclude /home/dmtsai -zcvf myfile.tar.gz /home/* /etc

范例八  将/etc/打包后直接解开在/tmp,而不产生文件!
[root@linux ~]# cd /tmp
[root@linux tmp]# tar -cvf - /etc | tar -xvf -
# 这个动作有点像是 cp -r /etc /tmp 啦～依旧是有其有用途的!
# 要注意的地方在于输出档变成 - 而输入档也变成 - ,又有一个 | 存在～
# 这分别代表 standard output, standard input 与管道命令
```

这是一个多用途的压缩指令,compress 与 gzip 是可以适用在一个文件的压缩上面,但是如果是要将一个目录压缩成一个文件,可以使用 tar。

tar 可以将整个目录或者是指定的文件都整合成一个文件,例如上面的范例一,tar 可以将/etc 下的文件全部整合成一个文件。

同时,tar 可以配合 gzip(gzip 的功能已经已经附加上 tar 中了),同时整合并压缩。

tar 用来作备份是很重要的指令,tar 整合过后的文件我们通常会取名为 *.tar,而如果还含有 gzip 的压缩属性,那么就取名为 *.tar.gz。取这个文件名只是为了方便记忆这个文件是什么属性,并没有实际的意义。

绝对路径与权限的问题

另外,需要注意的是,在使用的参数方面,有还有几个有用的参数需要来了解一番,也就是 -p 与 -P 这两个。

在我们的范例一中有提到一个警告信息,那就是“tar: Removing leading ‘/’ from member names”意思是说,tar 将/etc 目录的那个/去掉了,这是因为担心未来在解压缩的时候会产生一些困扰,因为在 tar 里面的文件如果是具有“绝对路径”的话,那么解压缩后的文件将会“一定”在该路径下也就是/etc,而不是相对路径。

这样子的最大困扰是万一有人拿走了这个文件,并且将该文件在系统上面解开。万一系统上面正巧也有/etc 这个目录(那当然是一定有的),它的文件就会“正巧”被覆盖了。

在预设的情况中,如果是以“绝对路径”来建立打包文件,那么 tar 将会自动的将/拿掉。这是为了刚刚说明的“安全”前提所做的预设值。

但是就是要以绝对路径来建立打包的文件,那么就加入 -P 这个参数(注意是大写字符)。

那么 -p 是什么 (小写字符) ? -p 是 permission 的意思,也就是“权限”,使用 -p 之后被打包的文件将不会依据使用者的身份来改变权限。

关于文件的更新日期:

这里还有一个值得注意的参数,那就是在备份的情况中很常使用的 -N 的这个参数。

可以参考一下上面的例子就可以知道,在这个例子当中相当重要的就是那个日期。

在备份的情况当中都希望只要备份较新的文件就好了,为什么呢?

因为旧的文件我们已经有备份,还要再备份一次,浪费时间也浪费系统资源。

关于 standard input/standard output:

在上面的例子中,最后一个例子很有趣:

tar cvf - /etc | tar -xvf -

它是直接以管道命令“pipe”来进行压缩、解压缩的过程。

在上面的例子中,我们想要“将/etc 下的数据直接 copy 到目前所在的路径,也就是/tmp 下”,但是又觉得使用 cp -r,那么就直接以这个打包的方式来打包,其中指令里面的 - 就是表示那个被打包的文件,由于不想要让中间文件存在,所以就以这一个方式来进行复制的行为。

什么是 tarfile 与 tarball?

tar 的功能相当的多,而由于他是经由“打包”之后再处理的一个过程,所以常常我们会听到 tarball 的文件,那就是经由 tar 打包再压缩的文件。而如果仅是打包而没有压缩的话,我们就称为 tarfile。

此外,tar 也可以用在备份的存储媒体上面,最常见的就是磁带机。假设磁带机代号为/dev/st0,那么要将/home 下的数据都给他备份上去时,就是使用 tar /dev/st0 /home 就可以。

在 Linux 中 gzip 已经被整合在 tar 了,但是 Sun 或者其他较旧的 Unix 版本中,当中的 tar 并没有整合 gzip,所以如果需要解压缩的话,就需要这么做

gzip -d testing.tar.gz tar -xvf testing.tar

第一个步骤会将文件解压缩,第二个步骤才是将数据解出来,与其他压缩程序不太一样的是,bzip2、gzip 与 compress 在没有加入特殊参数的时候,原先的文件会被取代掉,但是使用 tar 则原来的与后来的文件都会存在。

dd

在制作出 swap file 时使用的 dd 指令不只是制作一个文件而已,这个 dd 指令最大的功效,应该是在于“备份”。因为 dd 可以读取设备的内容,然后将整个设备备份成一个文件。dd 的用途有很多,这里仅讲一些比较重要的参数,如下:

```
[root@linux ~]# dd if="input_file" of="outptu_file" bs="block_size" \
count="number"
参数
if :就是 input file 棉～也可以是设备喔!
of:就是 output file 喔～也可以是设备;
bs:规划的一个 block 的大小,如果没有设定时,预设是 512 bytes
count 多少个 bs 的意思。
```

范例

范例一 将/etc/passwd 备份到/tmp/passwd.back 当中

```
[root@linux ~]# dd if=/etc/passwd of=/tmp/passwd.back
3+1 records in
3+1 records out
[root@linux ~]# ll /etc/passwd /tmp/passwd.back
-rw-r--r--  1 root root 1746 Aug 25 14:16 /etc/passwd
-rw-r--r--  1 root root 1746 Aug 29 16:57 /tmp/passwd.back
# /etc/passwd 文件大小为 1746 bytes,因为没有设定 bs,
# 所以预设是 512 bytes 为一个单位,因此上面那个 3+1 表示有 3 个完整的
# 512 bytes,以及未满 512 bytes 的另一个 block 的意思。
# 事实上,感觉好像是 cp 这个指令。
```

范例二：备份/dev/hda 的 MBR

```
[root@linux ~]# dd if=/dev/hda of=/tmp/mbr.back bs=512 count=1
1+0 records in
1+0 records out
# 我们知道整颗硬盘的 MBR 为 512 bytes,
# 就是放在硬盘的第一个 sector,因此可以利用这个方式来将
# MBR 内的所有数据都记录下来,真的很厉害吧!
```

范例三：将整个 /dev/hda1 partition 备份下来。

```
[root@linux ~]# dd if=/dev/hda1 of=/some/path/filenaem
# 这个指令很厉害啊!将整个 partition 的内容全部备份下来～
# 后面接的 of 必须要不是在 /dev/hda1 的目录内啊～否则,怎么读也读不完～
# 这个动作是很有效用的,如果改天你必须要完整的将整个 partition 的内容填回去,
# 则可以利用 dd if=/some/file of=/dev/hda1 来将数据写入到硬盘当中。
# 如果想要整个硬盘备份的话,就类似 Norton 的 ghost 软件一般,
# 由 disk 到 disk ,利用 dd 就可以。
```

可以说,tar 可以用来备份关键数据,而 dd 则可以用来备份整块 partition 或整个 disk。

不过,如果要将数据填回到 filesystem 当中,可能需要考虑到原本的 filesystem 才能成功。

cpio

cpio 指令是通过数据流重导向的方法来将文件进行输出/输入的一个方式。

```
[root@linux ~]# cpio -covB  > [file|device] <== 备份
[root@linux ~]# cpio -icduv < [file|device] <== 还原
参数 ：
-o ： 将数据 copy 输出到文件或设备上
-i ： 将数据自文件或设备 copy 出来系统当中
-t ： 查看 cpio 建立的文件或设备的内容
-c ： 一种较新的 portable format 方式存储
-v ： 让存储的过程中文件名称可以在显示器上显示
-B ： 让预设的 Blocks 可以增加至 5120 bytes,预设是 512 bytes,这样的好处是可以让大文件的存储速度加快。
-d ： 自动建立目录,由于 cpio 的内容可能不是在同一个目录内,
      如此的话在反备份的过程会有问题,这个时候加上-d 就可以自动的将需要的目录建立起来了。
-u ： 自动的将较新的文件覆盖较旧的文件。
```

范例：范例一：将所有系统上的数据通通写入磁带机内。

```
[root@linux ~]# find / -print | cpio -covB > /dev/st0
# 一般来说,使用 SCSI 介面的磁带机,代号是 /dev/st0。
```

范例二：检查磁带机上面有什么文件？

```
[root@linux ~]# cpio -icdvt < /dev/st0
[root@linux ~]# cpio -icdvt < /dev/st0 > /tmp/content
# 第一个动作当中,会将磁带机内的文件名列出到显示器上面,而我们可以通过第二个动作,
# 将所有的文件名通通记录到 /tmp/content 文件去!
```

范例三：将磁带上的数据还原回来～

```
[root@linux ~]# cpio -icduv < /dev/st0
# 一般来说,使用 SCSI 介面的磁带机,代号是 /dev/st0 喔!
```

范例四：将 /etc 底下的所有『文件』都备份到 /root/etc.cpio 中！

```
[root@linux ~]# find /etc -type f | cpio -o > /root/etc.cpio
# 这样就能够备份棉～您也可以将数据以 cpio -i < /root/etc.cpio
# 来将数据捉出来。
```

cpio 是最适用于备份的时候使用的一个指令，它并不像 cp 一样，可以直接的将文件 copy 过去，例如 cp * /tmp 就可以将所在目录的所有文件 copy 到/tmp 下。

在 cpio 这个指令的用法中，由于 cpio 无法直接读取文件，而是需要“每一个文件或目录的路径连同文件名一起”才可以被记录下来，因此 cpio 最常跟 find 这个指令一起使用。

cpio 是备份的时候的一项利器，它可以备份任何的文件，包括/dev 下的任何设备文件，而由于 cpio 必需要配合其他的程序，例如 find 来建立文件名，所以 cpio 与管道命令及数据流重导向的相关性就相当的重要了。

36.1 文件压缩技术

36.2 Linux 常用压缩与打包命令

36.3 Linux 备份

36.4 其他压缩与备份工具

第 37 章

LVM

Bibliography

[1] Wikipedia. Filesystem hierarchy standard (fhs), . URL http://en.wikipedia.org/wiki/Filesystem_Hierarchy_Standard.

[2] Wikipedia. 文件系统, . URL http://zh.wikipedia.org/zh-cn/%E6%96%87%E4%BB%B6%E7%B3%BB%E7%BB%9F.

[3] Wikipedia. 文件系统层次结构标准, . URL http://zh.wikipedia.org/wiki/%E6%96%87%E4%BB%B6%E7%B3%BB%E7%BB%9F%E5%B1%82%E6%AC%A1%E7%BB%93%E6%9E%84%E6%A0%87%E5%87%86.

[4] Wikipedia. Unix filesystem, . URL http://en.wikipedia.org/wiki/Unix_filesystem.

[5] Wikipedia. Extended file system, 03 1992. URL http://en.wikipedia.org/wiki/Extended_file_system.

[6] Wikipedia. Xfs, 1993. URL http://zh.wikipedia.org/wiki/XFS.

Part IV

Software Management

第 38 章

Applications

38.1 Compile

通常当使用手动编译源码方式来安装软件时，表示没有找到 RPM 或 DEB 安装资源，或者是需要以自定义的方式安装软件。采用手动编译源码方式安装需要自己编译源文件后再安装，所有通常需要系统有 gcc、make 之类的编译软件。

1. 下载源码，通常是 tar 文件。
2. 解压 tar 包，并在 Linux 终端运行如下命令：

```
tar -xzvf <文件名>
```

或：

```
tar -xjvf <文件名>
```

参数说明：

- -x，表示解压；
- -z 解压 gzip 格式文件；
- -j 解压 bzip2 格式的文件
- -v 显示详细信息；
- -f 解压到文件。

3. 编译，(通常在解压好的文件夹下有个 configure 文件，运行该文件即可，如果需要自定义安装，就需要查看帮助文档，查看编译参数，在 Linux 终端中运行./configure 命令。)
4. make。
5. make install。

在 configure 的过程中会提示错误，通常是提示你缺少某个组件，只需按照提示安装组件即可完成编译。

38.1.1 configure

38.1.2 make

38.1.3 make install

38.2 Package

软件包是对于一种软件所进行打包的方式，一般包括二进制文件和元数据(例如描述、版本和依赖)等。

如果正在使用某一款音乐播放器，那么它并不需要直接去操作声卡之类的硬件设备，而只需要去调用系统内核间接地控制声卡即可。更复杂一点的是如果要设计一个视频播放器，但现在并不知道如何去使用内核去操作显卡等硬件。如果知道有某个别人已经设计好的组件可以实现这样的功能，那么就可以只设计播放器的界面效果，然后直接使用别人的组件去调用内核来间接地控制硬件设备。

在 Linux 中的软件一般都是比较小巧、零散的，所以也就出现了安装某一个软件时提示依赖关系错误。即使是安装一个非常小的软件，但该软件可能需要依托于其他几十个组件的帮助才可以实现该软件应有的功能等情况。

在不同的操作系统中，软件包的类型有很大的区别。例如，Linux、BSD 系统中的软件包主要以两种形式出现：二进制包以及源代码包，其中源代码包主要适用于自由软件的安装，需要用户自己进行编译。

在 Windows 操作系统中，软件包大多数以安装程序的方式出现，可以将软件安装在制定的目录中，也有直接使用压缩工具打包的，解压缩之后便可运行。

不同的 Linux 发行版对于安装软件[?] 提供了多种解决方案。

- rpm：传统的 Red Hat Linux 二进制包。
- deb：Debian 系列的二进制包。

38.2.1 RPM

RPM、deb 安装方式一般针对特定发行版本，RPM 是针对红帽系统的安装包，deb 是针对 Ubuntu 系统的安装包，这种包会把相关软件及组件打包在一起，可以直接从网上下载 RPM 格式或 deb 格式的文件直接安装到相对应的系统里，但这种方式还是不能彻底解决依赖关系的问题。因为每个个人用户在安装系统时选择安装的组件不同，所以 RPM 包也不可能把所有相关的软件及组件都包括在里面。

安装 RPM 包的方法也很简单，直接在终端运行中运行：

```
1 rpm -ivh <文件名>
```

参数说明：

- -i，表示安装（install）。
- -v，显示附加信息。
- -h，显示 hash 符号（#）。

现在，RPM 可能是指.rpm 的文件格式的软件包，也可能是指其本身的软件包管理器（RPM Package Manager）。

RPM 仅适用于安装用 RPM 来打包的软件，具体可以分为二进制包（Binary）、源代码包（Source）和 Delta 包三种。二进制包可以直接安装在计算机中，而源代码包将会由 RPM 自动编译、安装，源代码包经常以 src.rpm 作为后缀名。

38.2.2 deb

deb 是 Debian 软件包格式，文件扩展名为.deb。处理 deb 包的经典程序是 dpkg，经常是通过前端工具 apt 来执行。

Debian 包是 Unixar 的标准归档，将包文件信息以及包内容，经过 gzip 和 tar 打包而成。

具体来说，deb 文件是使用 ar 打包，包含了三个文件：

- debian-binary - deb 格式版本号码
- control.tar.gz - 包含包的元数据[1]，如包名称、版本、维护者、依赖、冲突等。
- data.tar.* - 实际安装的内容

其中，“*” 所指代的内容随压缩算法不同而不同，常见的可能值为 xz、gz、bz2 或 lzma 等。

另外，通过 Alien 工具可以将 deb 包转换成其他形式的软件包。

38.2.3 APK

Android 应用程序包文件（Android application package，APK）是一种 Android 操作系统上的应用程序安装文件格式。

[1] 在 dpkg 1.17.6 之后添加了对 xz 压缩和不压缩的 control 元数据的支持。

每一个 Android 应用程序的代码都必须先进行编译，然后被打包成为一个被 Android 系统所能识别的文件才可以被运行，而这种能被 Android 系统识别并运行的文件格式便是“APK”。

一个 APK 文件内包含被编译的代码文件 (.dex 文件)、文件资源（resources）、assets、证书（certificates）和清单文件（manifest file）。

第 39 章

Environment Variables

第 40 章

Printer

第 41 章

Scanner

41.1 SANE

第 42 章

USB

42.1 lsusb

第 43 章

Kernel

43.1 Setup

43.2 X Window

43.2.1 GNOME

43.2.2 KDE

第 44 章

Development Envirment

44.1 Introduction

就软件开发项目而言,开发环境这一术语是指软件在开发和部署系统时所需的全部工件,其中包括工具、指南、流程、模板和基础设施。

其中,软件开发环境(Software Development Environment)指的是在基本硬件和软件的基础上,为支持系统软件和应用软件的工程化开发和维护而使用的一组软件,简称 SDE。具体来说,SDE 由软件工具和环境集成机制组成,前者用以支持软件开发的相关过程、活动和任务,后者为工具集成和软件的开发、维护及管理提供统一的支持。

软件开发环境在欧洲又叫集成式项目支援环境(Integrated Project Support Environment,IPSE),其主要组成部分是软件工具。

具有统一的交互方式的人机界面是软件开发环境的重要质量标志,而存储各种软件工具加工所产生的软件产品或半成品(如源代码、测试数据和各种文档资料等)的软件环境数据库是软件开发环境的核心。工具间的联系和相互理解都是通过存储在信息库中的共享数据得以实现的。

软件开发环境数据库是面向软件工作者的知识型信息数据库,其数据对象是多元化、带有智能性质的,从而可以用来支撑各种软件工具,尤其是自动设计工具、编译程序等的主动或被动的工作。

- 较初级的 SDE 数据库一般包含通用子程序库、可重组的程序加工信息库、模块描述与接口信息库、软件测试与纠错依据信息库等;
- 较完整的 SDE 数据库还应包括可行性与需求信息档案、阶段设计详细档案、测试驱动数据库、软件维护档案等。

更进一步的要求是面向软件规划到实现、维护全过程的自动进行,这要求 SDE 数据库系统是具有智能的,其中比较基本的智能结果是软件编码的自动实现和优化、软件工程项目的多方面不同角度的自我分析与总结。

对软件开发环境的配置还应主动地被重新改造、学习,以丰富 SDE 数据库的知识、信息和软件积累,因此软件开发环境在软件工程人员的恰当的外部控制或帮助下会逐步向高度智能与自动化迈进。

根据开发阶段的不同,开发环境可以划分为前端开发环境(支持系统规划、分析、设计等阶段的活动)、后端开发环境(支持编程、测试等阶段的活动)、软件维护环境和逆向工程环境等,这些开发环境往往可通过对功能较全的环境进行剪裁而得到。

软件开发环境由工具集和集成机制两部分构成,工具集和集成机制间的关系犹如“插件”和“插槽”间的关系。软件开发环境由工具集和集成机制两部分构成,工具集和集成机制间的关系犹如“插件”和“插槽”间的关系。

软件开发环境中的工具可包括支持特定过程模型和开发方法的工具,例如支持瀑布模型及数据流方法的分析工具、设计工具、编码工具、测试工具、维护工具,支持面向对象方法的 OOA 工具、OOD 工具和 OOP 工具等。

独立于模型和方法的工具(例如界面辅助生成工具和文档出版工具)亦可包括管理类工具和针对特定领域的应用类工具。

集成机制对工具的集成及用户软件的开发、维护及管理提供统一的支持。按功能可划分为环境信息库、过程控制及消息服务器、环境用户界面三个部分。

- 环境信息库是软件开发环境的核心,用以储存与系统开发有关的信息并支持信息的交流与共

享。库中储存两类信息，一类是开发过程中产生的有关被开发系统的信息（例如分析文档、设计文档、测试报告等），另一类是环境提供的支持信息（例如文档模板、系统配置、过程模型、可复用构件等）。

- 过程控制和消息服务器是实现过程集成及控制集成的基础。过程集成是按照具体软件开发过程的要求进行工具的选择与组合，控制集成并行工具之间的通信和协同工作。
- 环境用户界面包括环境总界面和由它实行统一控制的各环境部件及工具的界面。统一的、具有一致视感（Look & Feel）的用户界面是软件开发环境的重要特征。

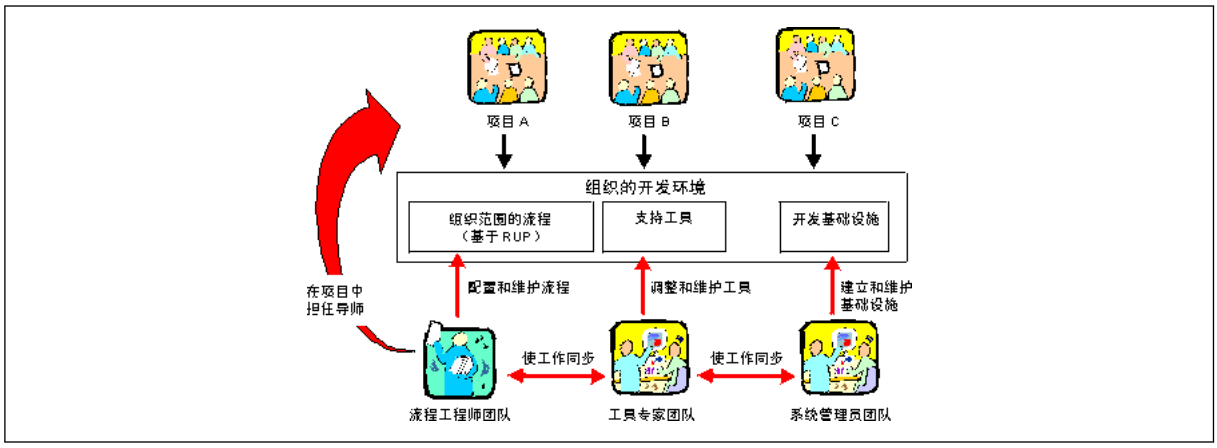
在某些情况下，有必要对软件开发环境的某些部分进行讨论，例如以下的两个示例：

- 测试环境，包括测试工件的模板（例如测试计划和测试评估概要的模板）、测试指南、测试工具和必要的开发基础设施。
- 实施环境，包括实施工件的模板（例如集成构建计划的模板）、编程指南、实施工具（例如编译器和调试器）以及必要的开发基础设施。

开发组织内的不同项目之间通常会存在许多相似之处，各项目集本上都以相似的方法使用同样的工具。对于不同的项目，流程大致相似，某些指南也可能一样。如果开发组织让一个团队来开发和维护组织的开发环境（即由组织范围的流程、工具使用和基础设施组成的开发环境），就可以提高开发效率。

流程工程师将负责开发和维护组织范围的流程。有了组织范围的流程，就可减少单独的软件开发项目中的流程定制工作，大部分定制工作已经在开发组织范围的流程时完成。

工具专家可以负责设置和维护支持工具，并协助各个软件开发项目进行工具设置。另外，系统管理员也可以成为此环境团队中的成员。



44.2 IDE

具体来说，集成开发环境（Integrated Development Environment，简称 IDE，也称为 Integration Design Environment、Integration Debugging Environment）是一种辅助程序开发人员开发软件的应用软件。

IDE 通常包括编辑器、自动构建工具、调试器、编译器/解释器，或者调用第三方编译器来实现代码的编译工作。另外，IDE 还可以包含 UML 建模工具、版本控制工具和图形用户界面设计工具，以及类浏览器、组件检查器等。

虽然也可以把 UNIX 当成是一个 IDE，但是多数的开发人员会把 IDE 当成是一个可以完成各种开发工作的一个程序，UNIX 提供了创建、修改、编译、发布、分析和调试等功能。IDE 试图把各种命令行的开发工具结合起来形成一个抽象化的工具，从而将各种开发工作进行更密切的整合（例如在编码的时候就直接编译，发现有语法错误就立即修改）。

在集成化开发环境之后出现的可视化开发环境使开发者可以将控件根据自己的意图进行组装，这样就解决了很多例行的、标准化的代码，比非可视化的开发环境更加直观，开发速度快，效率高。

以 Delphi 为例，Delphi 包含了程序代码文件（.PAS）和控件布局文件（.dfm），当在画布（FORM）上拖放一个按钮（BUTTON）时，Delphi 开发环境会自动创建一个 DFM 文件标明 BUTTON 位置，并且自动在 PAS 文件中自动输出最基本的完整代码，这样开发人员就只需要在需要修改的地方修改或者增加就可以完成很多功能。

44.3 SDK

软件开发工具包（Software Development Kit，SDK）一般是一些被开发人员用于为特定的软件包、软件框架、硬件平台、操作系统等创建应用软件的开发工具的集合。

SDK 可以只是简单的为某个程序设计语言提供应用程序接口的一些文件，但也可能包括能与某种嵌入式系统通讯的复杂的硬件，而且 SDK 一般都包括用于调试和其他用途的实用工具。

另外，SDK 还经常包括示例代码、支持性的技术文档或者其他的为基本参考资料澄清疑点的支持文档等。

为了鼓励开发者使用其系统或者语言，系统开发人员提供的 SDK 很多都是免费提供的。注意，SDK 可能附带了使其不能在不兼容的许可证下开发软件的许可证。例如，一个专有的 SDK 可能与自由软件开发抵触，而 GPL 能使 SDK 与专有软件开发近乎不兼容，只有 LGPL 下的 SDK 没有这个问题。

44.3.1 DirectX SDK

DirectX 是由 Microsoft 创建的一系列专为多媒体以及游戏开发的应用程序接口，包含了 Direct3D、Direct2D、DirectCompute 等多个不同用途的子部分，它们主要基于 C++ 编程语言实现，并遵循 COM 架构。

DirectX 被广泛用于 Microsoft Windows、Microsoft Xbox 电子游戏开发，并且只能支持这些平台。除了游戏开发之外，DirectX 亦被用于开发许多虚拟三维图形相关软件，例如 Direct3D 是 DirectX 中最广为应用的子模块。

- Direct3D 主要用于绘制 3D 图形。
- Direct2D 为 DirectDraw 的替代者，主要提供 2D 动画的硬件加速。
- DirectWrite 主要字体显示 API，可以控制 GPU 来使字体显示更为平滑，类似 ClearType。
- XInput 主要用于 Xbox360 的控制器。
- XAudio2 主要用于低延迟游戏音频播放。
- DirectCompute 为 GPU 通用计算 API。
- DirectXMath 为针对游戏优化的高速数学运算 API，使用 SSE2 指令集，特别支持单精度浮点运算及矩阵运算。
- DirectSetup 用于 DirectX 组件的安装，以及检查 DirectX 的版本。
- DirectX Media 包含 DirectAnimation、DirectShow 等，其中 DirectAnimation 可用于 2D 的网页动画（web animation），DirectShow 可支持多媒体回放、流媒体以及 DirectX 在网页上的转换。DirectShow 亦包含有 DirectX plugins 用于 audio signal processing 以及 DirectX Video Acceleration 加速影音音效。
- DirectX Media Objects 用于支持数据流对象（streaming objects），包括编码（encoders）、解码（decoder）以及效果（effects）等。

DirectX SDK 版本编号由微软的 DxDiag 工具获得（4.09.0000.0900 以及更高版本，在开始菜单 | 运行中输入 DxDiag 即可），编号统一使用 x.xx.xxxx.xxxx 格式，而微软网站上给出的编号使用 x.xx.xx.xxxx 格式，因此如果网站上编号为 4.09.00.0904，那么在电脑上安装后就会变为 4.09.0000.0904。

2010 年 6 月 7 日发布的 DirectX 11 SDK 是最后独立发布的 SDK 版本，之后的 DirectX SDK 被集成进新版的 Microsoft Windows SDK 里。例如，DirectX 11.2 SDK 被集成到 Windows Software Development Kit (SDK) for Windows 8.1 中。

44.3.2 Java SDK

Java Development Kit(JDK)是 Sun 针对 Java 开发人员发布的免费软件开发工具包。

自从 Java 推出以来，JDK 已经成为使用最广泛的 Java SDK。由于 JDK 的一部分特性采用商业许可证，因此 2006 年 Sun 宣布将发布基于 GPL 协议的开源 JDK，使 JDK 成为自由软件。在去掉了少量闭源特性之后，Sun 最终促成了 GPL 协议的 OpenJDK 的发布。

作为 Java 语言的 SDK，普通用户并不需要安装 JDK 来运行 Java 程序，而只需要安装 JRE（Java Runtime Environment），但是程序开发者必须安装 JDK 来编译、调试程序。

JDK 包含了一批用于 Java 开发的组件，其中包括：

Table 44.1: JDK 组件

JDK 组件	备注
appletviewer	运行和调试 applet 程序的工具，不需要使用浏览器
apt	注释处理工具
extcheck	检测 jar 包冲突的工具
idlj	IDL-to-Java 编译器，可以将 IDL 语言转化为 java 文件
jar	打包工具，将相关的类文件打包成一个文件
jarsigner	
java	运行器，执行.class 的字节码
javac	编译器，将后缀名为.java 的源代码编译成后缀名为.class 的字节码
javadoc	文档工具，从源码注释中提取文档，注释需符合规范
javafxpackager	
javah	从 Java 类生成 C 头文件和 C 源文件。这些文件提供了连接胶合，使 Java 和 C 代码可进行交互。
javap	反编译程序
javapackager	
java-rmi.cgi	
javaws	运行 JNLP 程序
jcmd	
jconsole	
jcontrol	
jdb	java 调试器
jdeps	
jhat	堆分析工具
jinfo	获取正在运行或崩溃的 java 程序配置信息
jjs	
jmap	获取 java 进程内存映射信息
jmc	
jmc.ini	
jps	显示当前 java 程序运行的进程状态
jrunscript	命令行脚本运行工具
jsadebugd	
jstack	栈跟踪程序
jstat	JVM 检测统计工具
jstatd	jstat 守护进程

JDK 组件	备注
jvisualvm	
keytool	
native2ascii	
orbd	
pack200	
policytool	策略文件创建和管理工具
rmic	
rmid	
rmiregistry	
schemagen	
serialver	
servertool	
tnameserv	
unpack200	
wsgen	
wsimport	
xjc	

JDK 中还包括完整的 JRE (Java Runtime Environment), 用于生产环境的各种库类 (例如基础类库 rt.jar), 以及给开发人员使用的补充库(例如国际化与本地化的类库、IDL 库等)。

另外, JDK 中还包括各种样例程序, 用以展示 Java API 中的各部分。

无论 Linux、Windows 或者 Mac OS 系统, JDK 均有 X86 与 X64 的发行版本, 并且均为多语言发行, 可以根据系统语言的不同自动显示不同语言的信息。

自 JDK 5.0 起, Java 以两种方式发布更新:

- Limited Update 包含新功能和非安全修正, 版本号是 20 的倍数。
- Critical Patch Updates(CPUs) 只包含安全修正, 版本号将是上一个 Limited Update 版本号加上五的倍数后的奇数。

44.3.3 OpenJDK

OpenJDK 是甲骨文公司公司为 Java 平台构建的 Java 开发环境的开源版本, 完全自由, 开放源码。

44.3.4 Android SDK

Android SDK 提供了 Android 开发过程中需要的 API 库和必要的工具用于构建、测试和调试 Android 应用程序。

- ADT Plugins
- Android SDK Tools
- Android Platform-tools
- Android Platform
- Android system image

44.3.5 Android NDK

44.3.6 iOS SDK

iOS 软件开发工具包(iOS SDK,亦称 iPhone SDK)是由苹果公司开发的为 iOS 设计的应用程序开发工具包,允许开发者开发 iPad、iPhone、iPod touch 应用程序。

iOS SDK 的首个版本于 2008 年 2 月发布,苹果通常会发布两个 iOS 软件开发工具包,包括主要的 iOS X.0 和次要的 iOS X.X。iOS SDK 只能在 Mac OS X Leopard 及以上系统并拥有英特尔处理器上运行,其他的操作系统(包括微软的 Windows 操作系统和旧版本的 Mac OS X 操作系统)都不被支持。

iOS 软件开发工具包本身是可以免费下载的,但开发人员如果希望向 App Store 发布应用,就必需加入 iOS 开发者计划,需要付款以获得苹果的批准。加入 iOS 开发者计划后,开发人员们将会得到一个牌照,可以用这个牌照将编写的软件发布到苹果的 App Store。

开发人员可以通过应用商店发布任意设价的应用程序,付费应用将让开发人员获得 70% 的费用配额,免费的应用程序没有任何费用配额。

iOS 开发者计划的出现使人们不能根据 GPLv3 的授权代码发布软件。任何根据 GPLv3 任何代码的开发者也必须得到 GPLv3 的授权。同时,开发商在散发布已经由 GPLv3 授权的应用软件的同时必须提供由苹果公司提供的密匙以允许该软件修改版本的上传。

iOS 是从于 Mac OS X 核心演变而来,因此从 Xcode 3.1 发布以后,Xcode 就成为了 iOS 软件开发工具包的开发环境。和 Mac OS X 的应用程序一样,iOS 应用程序使用 Objective-C 语言,也可以写成 C 或 C++ 语言。

- 触控 (Cocoa Touch):多点触控事件和控制 (Multi-touch events and controls)、加速支持 (Accelerometer support)、查看等级 (View hierarchy)、本地化(i18n)(Localization (i18n))、相机支持。
- 媒体: OpenAL、混音及录音 (Audio mixing and recording)、视频播放、图像文件格式 (Image file formats)、Quartz、Core Animation、OpenGL ES。
- 核心服务:网络、SQLite 嵌入式数据库、地理位置 (GeoLocation)、线程 (Threads)。
- OS X 核心:TCP/IP 协议、套接字 (Sockets)、电源管理、文件系统 (File system)、安全。

iOS 软件开发工具包中包含和 Xcode 工具一样的 iOS 模拟器,让开发人员在计算机上拥有仿真的外观和感觉,但是 iOS 模拟器并不是一个用于运行 x86 目标代码的工具。

从 iOS 2.1 固件开始,iPhone 版 Safari 开始支持 SVG 1.1 的编码特征和大部分静态功能。不过 Safari 的图形界面还不支持 SMIL 动画,这需要等 SMIL 引擎足够成熟之后才能被支持。另外,Safari 还支持 HTML Canvas。

目前,Apple 仍未开放在浏览器内执行 Flash 内容,因此只能在 iOS 越狱之后安装第三方 Flash 软件。

第 45 章

Library

45.1 Introduction

在计算机科学中，库(library)是用于开发软件的子程序集合，还可能包括配置、文档、帮助、模板、类或函数、值或类型规格等。

现代软件开发往往利用模块化开发的方式，每一个软件模块(Module)都是一套一致而互相有紧密关连的软件组织，分别包含了程序和数据结构两部份。

模块是可能分开地被编写的单位，从而使得它们可重用和允许不同的开发人员同时协作、编写及研究不同的模块。

模块的接口表达了由该模块提供的功能和调用它时所需的元素，因此库与可执行文件不同，它不是独立程序，其他程序必须通过库的接口来引用库的功能。

静态链接和动态链接都可以把一个或多个库包括到程序中。相应的，前者链接的库叫做静态库，后者的叫做动态库。

标准库是程序设计语言的每种实现中都统一提供的库。在某些情况下，编程语言规格说明中会直接提及该库，或者由编程社区中的非正式惯例决定。

根据宿主语言构成要素的不同，标准库可包含如下要素：

- 子程序
- 宏定义
- 全局变量
- 类定义
- 模板

大多数标准库都至少含有如下常用组件的定义：

- 算法(例如排序算法)
- 数据结构(例如表、树、哈希表)
- 与宿主平台的交互，包括输入输出和操作系统调用

45.2 Static Library

静态库是相对于动态库而言的。对静态库代码的调用中，在链接阶段就会把库代码包含入可执行文件。

在计算机科学里，静态库是一个外部函数与变量的集合体，静态函数库通常包含变量与函数，在编译期间由编译器与链接器将它们整合至应用程序内，并生成目标文件及可以独立运行的可执行文件。

链接器是一个独立程序，将一个或多个库或目标文件(先前由编译器或汇编器生成)链接到一块生成可执行程序。静态链接是由链接器(Linker)在链接时将库的内容加入到可执行程序中的做法，因此以过去的观点来说，函数库只能算是静态(static)类型。

静态函数库可以用 C/C++ 语言来建立，它们可以提供关键字来指定函数与变量是否为外部

(external) 或是内部 (internal) 链接。如果需要将函数或是变量导出 (export), 则一定要用外部链接 (external linkage)的语法来指定它们。

```
// static_lib.h

# ifndef _STATIC_LIB_H_
# define _STATIC_LIB_H_

# include <iostream>
# include <string>
# include <Windows.h>

using namespace std;

namespace STAIC_LIB
{
  BOOL PRINT(__in string& STRING);
}

# endif
```

Listing 45.1: static_lib.h

```
// static_lib.cpp

# include "static_lib.h"

BOOL STAIC_LIB::PRINT(__in string& STRING)
{
  if ( STRING.empty() )
  {
    return FALSE;
  }
  // 显示一个字符串
  cout<<STRING<<endl;
  return TRUE;
}
```

Listing 45.2: static_lib.cpp

GCC 编译器中的静态库文件名为 lib*.a, 为了使用其中的函数, 可以使用 -l* 参数要求链接器连入。例如, 在许多系统上, 当使用了 math.h 中的函数后, 需要使用 -lm 参数连接 libm.a 文件。

在 Visual Studio 中, 静态库文件名为 *.lib。为了使用其中的函数, 可以使用 `#pragma comment(lib,"*")` 预编译指令要求连接器连入。

与动态链接相比, 静态链接的最大缺点是生成的可执行文件太大, 需要更多的系统资源, 在装入内存时也会消耗更多的时间。但是, 使用静态库时, 只需保证在开发者的计算机有正确的库文件, 在以二进制发布时不需考虑在用户的计算机上库文件是否存在及版本问题, 可避免 DLL 地狱等问题。

45.3 Shared Library

动态链接是指在可执行文件装载时或运行时由操作系统的装载程序加载库。可以动态链接的库, 在 Windows 上是 dynamic link library (DLL), 在 UNIX 或 Linux 上是 Shared Library。

库文件是预先编译链接好的可执行文件, 存储在计算机的硬盘上。大多数情况下, 同一时间多个应用可以使用一个库的同一份拷贝, 操作系统不需要加载这个库的多个实例。

大多数操作系统将解析外部引用 (比如库) 作为加载过程的一部分, 这些系统中的可执行文件包含一个叫做 import directory 的表, 该表的每一项包含一个库的名字。根据表中记录的名字, 装载程序

在硬盘上搜索需要的库,然后将其加载到内存中预先不确定的位置,之后根据加载库后确定的库的地址更新可执行程序。

装载程序在加载应用软件时要完成的最复杂的工作之一就是加载时链接。可执行程序根据更新后的库信息调用库中的函数或引用库中的数据,这种类型的动态加载称为装载(load-time)时加载,已被 Windows 和 Linux 等大多数系统采用。

其他操作系统可能在运行时解析引用,这些系统中的可执行程序调用操作系统 API 后,将库的名字、函数在库中的编号和函数参数一同传递。操作系统负责立即解析然后代表应用调用合适的函数,这种动态链接叫做运行时链接。因为每个调用都会有系统开销,运行时链接要慢得多,对应用的性能有负面影响,现代操作系统已经很少使用运行时链接。

动态链接的最大缺点是可执行程序依赖分别存储的库文件才能正确执行。如果库文件被删除了、移动了、重命名了或者被替换为不兼容的版本了,那么可执行程序就可能工作不正常,这就是常说的 DLL-hell。

45.4 Runtime Library

在计算机程序设计领域中,运行时库(runtime library)是指一种被编译器用来实现编程语言内置函数以提供该语言程序运行时(执行)支持的一种特殊的计算机程序库。

运行时库一般包括基本的输入输出或是内存管理等支持,与操作系统合作提供诸如数学运算、输入输出等功能,让开发者可以不需要“重新发明轮子”,并高效使用操作系统提供的功能。

运行时库由编译器决定,以面向编程语言来提供其最基本的执行时需要。比如,Visual Basic 需要复杂的运行时库支持而 C 的运行时库则相对简单。

运行时库中的函数可能对程序员透明,也可能不透明,这取决于编译器对语言执行环境的需求。

早期的运行时库(例如 Fortran)提供了数学运算的能力。其他语言增加了诸如垃圾回收的先进功能,通常用于支持对象数据结构。

许多近代语言设计了更大的运行环境并添加更多功能,很多面向对象语言也包含了分派器与类读取器。例如,Java 虚拟机(JVM)便是此类的典型运行环境,它也在运行期直译或编译具可移植性的二进制 Java 程序。另外,.NET 架构也是另外一个运行时库的实例。

一个以 Java 语言开发的软件,可通过 Java 软件运行可预测的指令接收 Java 运行环境的服务功能。根据 Java 运行时库提供的这些服务,Java 运行环境可视为此程序的运行时环境,程序与 Java 环境都向操作系统提出请求并获取服务。

操作系统核心为它自己、所有进程与在它控制之下的软件提供服务,因此操作系统可视为自己提供自己的运行时环境。

异常处理(Exception handling)是专门处理运行期错误的语言机制,使程序员可以完全捕捉非预期错误,或没有适当处理的错误结果。

动态链接库或静态链接库与运行时库的分类角度不同,不得相提并论。

45.5 Class Libary

45.5.1 Framework Class Library

框架类库 (Framework Class Library) 通常被定义为可重用类、接口和值类型等, 其中基本类库(Base Class Library)是框架类库的核心,提供了基本类和名称空间等,从而实现了框架的基本功能。例如,.NET Framework Class Library 就包括了 System, System.CodeDom, System.Collections, System.Diagnostics, System.Globalization, System.IO, System.Resources and System.Text 等名称空间。

Microsft 的.NET Framework 就是一个以通用语言运行库(Common Language Runtime)为基础,致力

于敏捷软件开发(Agile software development)、快速应用开发(Rapid application development)、平台无关性和网络透明化的类库集合。

.NET Framework 也为应用程序接口(API)提供了新功能和开发工具,以及新的反射性的且面向对象程序设计编程接口。

.NET Framework 的初级组成是 CLI 和 CLR,其中:

- CLI(Common Language Infrastructure,通用语言架构)是一套运作环境帮助,包括一般系统、基础类库和与机器无关的中间代码。
- CLR(Common Language Runtime,通用语言运行平台)是对通用语言架构的实现。在通用中间语言(CIL)运行前,CLR 必须将指令及时编译转换成原始机器码。

所有 CIL 都可经由.NET 自我表述,CLR 检查元数据以确保正确的方法被调用。元数据通常是由语言编译器生成,但开发人员也可以通过使用客户属性创建自己的元数据。

如果一种语言实现生成了 CLI,它也可以通过使用 CLR 被调用,这样它就可以与任何其他.NET 语言生成的数据相交互,也就是说 CLR 被设计为具有操作系统无关性。

当一个汇编体被加载时,CLR 运行各种各样的测试(例如确认与核查)。在确认的时候,CLR 检查汇编体是否包含有效的元数据和 CIL,并且检查内部表的正确性。核查则不那么精确,核查机制主要检查代码是否会运行一些“不安全”的操作。

公共语言基础(CLI)、通用中间语言(CIL)以及 C# 与 Sun 的 Java 虚拟机和 Java 之间有不少的相似之处,二者都使用它们各自的中间码,因此它们无疑是强烈的竞争者。

CLI 被设计成支持任何面向对象的编程语言,CIL 被设计来实时编译(JIT),而 Java 的字节码在最初的时候则是设计成用于解释运行,而非实时编译。

ASP.NET 和 ADO.NET 也都是内含于.NET 框架中的可用于开发 Web 应用程序以及数据访问的类库。

45.5.2 Java Class Library

Java 被设计成具有平台无关性,因此 Java 类库(JCL)封装了一组在 Java 应用运行时可以动态载入的标准类库,包含了在不同操作系统上都通用的功能。

基本上来说,Java 类库都是由 Java 语言实现的,并且几乎所有的 Java 类库都包含在 rt.jar 中,它们都可以通过 Java 本地接口(Java Native Interface,JNI)来访问操作系统 API。

Java 集合框架(Java Collections Framework,JCF)是一组实现了通用数据结构的集合类和接口。

- Collection 接口是一组允许重复的对象。
- Set 接口继承 Collection,但不允许重复,使用自己内部的一个排列机制。
- List 接口继承 Collection,允许重复,以元素安插的次序来放置元素,不会重新排列。
- Map 接口是一组成对的键–值对象,即所持有的是 key-value pairs。Map 中不能有重复的 key,拥有自己的内部排列机制。

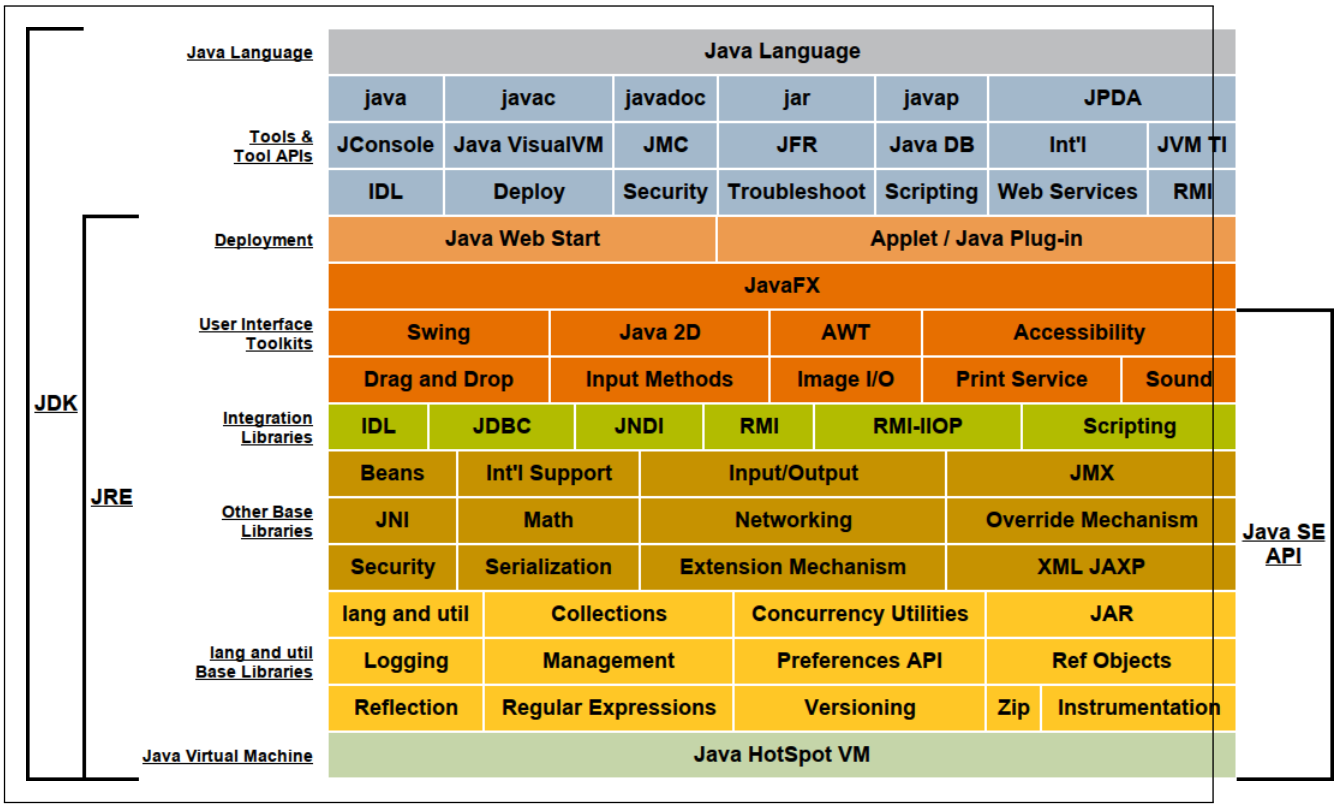
45.5.3 Java Package

Java Package 是一种将 Java 类或接口等组织到名字空间中的机制,它可以被保存为压缩的 jar 文件中,从而以组的形式下载 Java 类。

Java API 本身就是包的集合,例如 javax.xml 包就包含了处理 XML 的类。另外,每一个包都按照它所包含的类型提供了唯一的名字空间,而且在同一个包内的类可以访问其中的成员。

在开发 Java 应用时,可以使用 package 关键字来引入相关的类,例如:

```
1 package java.awt.event;
2 \end{lstlsiting}
3
4 为了使用某个Java包中的所有类，可以使用import关键字来导入某个包中的所有的类。
5
```




```
6
7  \begin{lstlisting}[language=Java,xleftmargin=.25in]
8  import java.awt.event.*;
9  \end{lstlsiting}
10
11 或者，使用improt关键字也可以导入某个确定的类。例如，下面的示例说明了如何从java.awt.event包中导入ActionEvent类。
12
13 \begin{lstlisting}[language=Java,xleftmargin=.25in]
14 import java.awt.event.ActionEvent;
15 \end{lstlsiting}
16
17 使用上述方式导入Java包之后，就可以在代码中使用相关的类。
18
19 \begin{lstlisting}[language=Java,xleftmargin=.25in]
20 ActionEvent testEvent = new ActionEvent();
21 \end{lstlsiting}
22
23 如果不使用Java包来导入类，也可以使用绝对位置来引用类，例如：
24
25 \begin{lstlisting}[language=Java,xleftmargin=.25in]
26 java.awt.event.ActionEvent testEvent = new java.awt.event.ActionEvent();
27 \end{lstlsiting}
28
29 为了更好地组织代码结构，可以使用jar命令来将.class文件打包到jar文件中，例如：
30
31 \begin{lstlisting}[language=bash,xleftmargin=.25in]
32 jar cf testPackage.jar *.class
33 \end{lstlsiting}
34
35 在Java应用开发中，经常使用的核心包如下：
36
37 \begin{compactitem}
```

```
\item java.lang--basic language functionality and fundamental types
\item java.util--collection data structure classes
\item java.io--file operations
\item java.math--multiprecision arithmetics
\item java.nio --the New I/O framework for Java
\item java.net--networking operations, sockets, DNS lookups, ...
\item java.security--key generation, encryption and decryption
\item java.sql--Java Database Connectivity (JDBC) to access databases
\item java.awt--basic hierarchy of packages for native GUI components
\item javax.swing--hierarchy of packages for platform-independent rich GUI components
\item java.applet--classes for creating an applet
\end{compactitem}




\chapter{Package Manager}



\section{Introduction}


软件一般是按模块被设计出来的，用户可以直接面向UI界面操作，而计算机在设计之初就是分层次分模块被设计出来的，
    包括系统内核部分（Linux内核、Unix内核、Windows内核）和基础服务部分。




软件包是对于一种软件所进行打包的方式。在不同的操作系统中，软件包的类型有很大的区别。



在Linux发行版中，几乎每一个发行版都重度依赖于某种软件包管理系统（例如RPM、dpkg等），可以将其理解为在计算机中执行安装、
    配置、卸载和升级软件包的工具组合。

采用RPM、dkpg等解决方案的原因是使用简单，基本无依赖关系问题。具体来说，
    软件包管理系统的原理是搭建一台文件服务器来把所有可能用到的软件包放进去，剩下的就是需要用户把自己电脑的Yum源（或deb
    源）指定到该服务器的地址。

\begin{compactitem}
\item RPM（RPM Package Manager）是由Red Hat推出的软件包管理系统。
\item dpkg（Debian Package）由Debian发行版开发以用于安装、卸载以及提供和deb软件包相关的信息。
\end{compactitem}

其他软件包管理系统有ArchLinux中使用的Pacman，Gentoo使用的基于源代码的Portage和Mac系统下的Homebrew等。

\begin{compactitem}
\item ArchLinux基于KISS原则，针对i686和x86-64的CPU做了优化，以.pkg.tar.xz格式打包并由包管理器进行跟踪维护。
\item Gentoo采用Portage包管理系统来自行编译及调整源码依赖等选项，以获得至高的自定义性及优化的软件。
\item Mac OS X系统使用Homebrew简化软件安装和2管理。
\end{compactitem}

```

```
92
93
94  在典型的软件包管理系统中，使用dpkg以及它的前端apt（使用于Debian、Ubuntu）来管理deb软件包，而rpm以及它的前端yum（
        使用于Fedora、RHEL、CentOS）则用来管理rpm软件包。
95
96
97
98  \subsection{YUM}
99
100
101 使用软件包管理系统将大大简化在Linux发行版中安装软件的过程，例如YUM源配置文件在/etc/yum.repos.d/目录下，
        并且文件名以.repo结尾，文件内容主要是名称和服务器地址。
102
103 .repo文件内容格式如下：
104
105 \begin{compactitem}
106 \item 可选项[]中的内容为项目名称。
107 \item name为服务器名称。
108 \item baseurl为服务器地址，该地址一定是一个真实、可用的YUM服务器地址。
109 \item enable=0表示不启动Yum服务，如果想使用该服务，需要修改为1。
110 \item gpgcheck=1表示是否对软件进行签名检验，0为不校验。
111 \item gpgkey表示校验签名文件位置。
112 \end{compactitem}
113
114 如果有多个网络YUM服务器，可以在下面继续添加YUM项目并且格式相同，修改完YUM配置文件后需要运行\texttt{yum clean all
        }来初始化新的配置文件。
115
116
117 \subsection{APT}
118
119
120 apt-get原理与YUM一样，只不过RedHat公司用的是yum命令，而Ubuntu公司用的是apt-get命令。
121
122
123
124
125
126
127 \section{Python}
128
129
130
131
132 \subsection{pip}
133
134 pip是一个以Python语言开发的软件包管理系统，可以用于安装和管理软件包。类似地，EasyInstall也是一个Python
        语言的软件包管理系统，提供了一个标准的分发Python软件和函数库的格式。
135
136 easy\_install是一个附带设置工具的模块和一个第三方函数库，类似用于Ruby语言的RubyGems，可以加快Python
        函数库的分发程序的速度。
137
138
139 通过pip提供的命令行接口，可以方便地安装软件包。
140
141 \begin{lstlisting}[language=bash,xleftmargin=.25in]
142 pip install package-name
```

Python Package Index(简称 PyPI)是一个由 Python 基金会维护的第三方 Python 软件仓库,因此 pip 安装软件包时默认都是从 PyPI 进行查找的。

此外,也可以方便地使用下面的命令来移除软件包。

```
pip uninstall package-name
```

pip 也可以通过“需求”文件来管理软件包和其相应版本数目的完整列表,从而可以对一个完整软件包组合在另一个环境(如另一台计算机)或虚拟化环境中进行有效率的重新创造。

```
pip install -r requirements.txt
```

另一方面,pip 也可以通过“Heroku”等软件支持 Python 在云端网页缓存中的使用。

45.5.4 easy_install

45.6 JavaScript

45.6.1 npm

Node.js 是一个基于 Google 的 V8 引擎的事件驱动 I/O 服务端 JavaScript 环境,可以用来开发可扩展的网络程序(例如 Web 服务)。

与一般 JavaScript 不同的地方在于,Node.js 并不是在浏览器上运行的,而是在服务器上运行服务端 JavaScript。

Node 包管理器(Node Package Manager)运行在命令行下,可以用来管理 Node.js 应用的依赖。例如,可以使用下面的命令安装 LESS。

```
$ npm install less
```

从 Node.js0.6 版本开始,npm 被自动附带在安装包中。

Node.js 实现了部份 CommonJS 规范,并包含了一个交互测试 REPL 环境。例如,下面是一个用 Node.js 撰写的 HTTP Server 版 hello world 示例。

```
var http = require('http');

http.createServer(function (request, response) {
   response.writeHead(200, {'Content-Type': 'text/plain'});
   response.end('Hello World\n');
}).listen(8000);

console.log('Server running at http://127.0.0.1:8000/');
```

下面是另一个简单的 TCP 服务器示例,监听(Listening)端口 7000 并输出 (echo) 之前输入的消息。

```
var net = require('net');

net.createServer(function (stream) {
   stream.write('hello\r\n');

   stream.on('end', function () {
      stream.end('goodbye\r\n');
   });

   stream.pipe(stream);
}).listen(7000);
```

45.6.2 nvm

目前，Node.js 的各种特性都没有稳定下来，经常由于老项目或尝新的原因，需要切换各种版本，因此开发了 nvm（Node Version Manager）。

首先，通过下面的命令来安装 nvm。

```
$ curl https://raw.githubusercontent.com/creationix/nvm/v0.17.2/install.sh | bash
```

安装完成后，在 shell 里面调用 nvm 命令，如果有如下输出，则说明 nvm 安装成功。

```
Node Version Manager

Usage:
   nvm help             Show this message
   nvm --version        Print out the latest released version of nvm
   nvm install [-s] <version> Download and install a <version>, [-s] from source. Uses .
       nvmrc if available
   nvm uninstall <version> Uninstall a version
   nvm use <version>    Modify PATH to use <version>. Uses .nvmrc if available
   nvm run <version> [<args>] Run <version> with <args> as arguments. Uses .nvmrc if
       available for <version>
   nvm current          Display currently activated version
   nvm ls               List installed versions
   nvm ls <version>     List versions matching a given description
   nvm ls-remote        List remote versions available for install
   nvm deactivate       Undo effects of NVM on current shell
   nvm alias [<pattern>] Show all aliases beginning with <pattern>
   nvm alias <name> <version> Set an alias named <name> pointing to <version>
   nvm unalias <name>   Deletes the alias named <name>
   nvm copy-packages <version> Install global NPM packages contained in <version> to
       current version
   nvm unload           Unload NVM from shell

Example:
   nvm install v0.10.24 Install a specific version number
   nvm use 0.10         Use the latest available 0.10.x release
   nvm run 0.10.24 myApp.js Run myApp.js using node v0.10.24
   nvm alias default 0.10.24 Set default node version on a shell

Note:
   to remove, delete, or uninstall nvm - just remove ~/.nvm, ~/.npm, and ~/.bower folders
```

45.6.3 Bower

Bower is a package management system for client-side programming on the World Wide Web. It depends on Node.js and npm. It works with git and GitHub repositories.

45.7 Perl

45.7.1 ppm

Perl Package Manager（PPM）可以用于简化查找、安装、升级和移除 Perl 软件包的工作。

ppm 可以检查已安装软件的版本，并且还可以从本地或远程主机来安装或升级软件。

ppm 可以从“PPM 仓库”中查找预编译的 Perl 模块，例如 ppm 可以重新构建从 CPAN 下载的 PPM 模块。

45.8 PHP

45.8.1 PECL

45.8.2 PEAR

45.8.3 Composer

45.9 Ruby

45.9.1 gem

RubyGems 是 Ruby 的一个包管理器，提供了分发 Ruby 程序和函数库的标准格式“gem”，旨在方便地管理 gem 安装的工具，以及用于分发 gem 的服务器，类似于 Python 的 pip。

RubyGems 创建于约 2003 年 11 月，从 Ruby 1.9 版起成为 Ruby 标准库的一部分。

Gem 是类似于 Ebuilds 的包，其中包含包信息，以及用于安装的文件。

Gem 通常是依照“.gemspec”文件构建的，其为包含了有关 Gem 信息的 YAML 文件。不过，Ruby 代码也可以利用 Rake 来直接建立 Gem。

具体来说，gem 命令用于构建、上传、下载以及安装 Gem 包，在功能上与 apt-get、portage、yum 和 npm 非常相似。

```
安装:
 gem install mygem
卸载:
 gem uninstall mygem
列出已安装的gem:
 gem list --local
列出可用的gem，例如:
 gem list --remote
为所有的gems创建RDoc文档:
 gem rdoc --all
下载一个gem，但不安装:
 gem fetch mygem
从可用的gem中搜索，例如:
 gem search STRING --remote
```

gem 命令也被用来构建和维护.gemspec 和.gem 文件。例如，可以通过下面的命令来利用.gemspec 文件构建.gem。

```
gem build mygem.gemspec
```

第 46 章

Update

第 47 章

内核

在计算机科学中,内核(Kernel)又称核心[?],是操作系统最基本的部分。

内核主要负责管理系统资源,作为众多应用程序提供对计算机硬件的安全访问的一部分软件,对内核的访问是有限的,并由内核决定一个程序在什么时候对某部分硬件操作多长时间。由于直接对硬件操作是非常复杂的,因此通常是通过内核来提供一种硬件抽象的方法以完成这些操作。

通过进程间通信机制及系统调用,应用进程可间接控制所需的硬件资源(特别是处理器及 IO 设备)。

严格地说,内核并不是计算机系统中必要的组成部分,程序也可以直接地被调入计算机中执行,不过这种情况常见于早期计算机系统的设计中。当时使用这样的设计,说明设计者不希望提供任何硬件抽象和操作系统的支持。最终,一些辅助性程序(例如程序加载器和调试器)被设计到机器核心当中,或者写入在只读记忆体里。这些变化发生时,操作系统内核的概念就渐渐明晰起来了。

内核可分为四大类:

- 单内核:单内核为潜在的硬件提供了大量完善的硬件抽象操作。
- 微内核:微内核只提供了很小一部分的硬件抽象,大部分功能由一种特殊的用户态程序——服务来完成。
- 混合内核:混合内核很像微内核结构,只不过它的组件更多的在核心态中运行,用以获得更快的执行速度。
- 外内核:外内核不提供任何硬件抽象操作,但是允许为内核增加额外的运行库,通过这些运行库应用程序可以直接地或者接近直接地对硬件进行操作。

47.1 Monolithic kernels

宏内核结构在硬件之上,定义了一个高阶的抽象界面,应用一组原语 (或者叫系统调用 (System call)) 来实现操作系统的功能,例如进程管理,文件系统,和存储管理等等,这些功能由多个运行在核心态的模块来完成。

尽管每一个模块都是单独地服务这些操作,内核代码是高度集成的,而且难以编写正确。因为所有的模块都在同一个内核空间上运行,一个很小的 bug 都会使整个系统崩溃。然而,如果开发顺利,单内核结构就可以从运行效率上得到好处。

很多现代的宏内核结构内核,如 Linux 和 FreeBSD 内核,能够在运行时将模块调入执行,这就可以使扩充内核的功能变得更简单,也可以使内核的核心部分变得更简洁。

宏内核结构的例子:

- 传统的 UNIX 内核,BSD 伯克利大学发行的版本
- Linux 内核
- MS-DOS, Windows 9x (Windows 95, 98, Me)

47.2 Micro kernels

微内核结构由一个非常简单的硬件抽象层和一组比较关键的原语或系统调用组成;这些原语,仅仅包括了创建一个系统必需的几个部分;如线程管理,地址空间和进程间通讯等。

微核的目标是将系统服务的实现和系统的基本操作规则分离开来。例如,进程的输入/输出锁定服务可以由运行在微核之外的一个服务组件来提供。这些非常模块化的用户态服务器用于完成操作系统中比较高级的操作,这样的设计使内核中最核心的部分的设计更简单。一个服务组件的失效并不会导致整个系统的崩溃,内核需要做的,仅仅是重新启动这个组件,而不必影响其它的部分。

微内核将许多 OS 服务放入分离的进程, 如文件系统, 设备驱动程序, 而进程通过消息传递调用 OS 服务. 微内核结构必然是多线程的, 第一代微内核, 在核心提供了较多的服务, 因此被称为' 胖微内核'; 它的典型代表是 MACH, 它既是 GNU HURD 也是 APPLE SERVER OS 的核心, 可以说, 蒸蒸日上. 第二代微内核只提供最基本的 OS 服务, 典型的 OS 是 QNX,QNX 在理论界很有名, 被认为是一种先进的 OS。

微内核结构的例子:

- AIX
- BeOS
- L4 微内核系列
- Mach, 用于 GNU Hurd
- Minix
- MorphOS
- QNX
- RadiOS
- VSTa

单内核结构是非常有吸引力的一种设计,由于在同一个地址空间上实现所有复杂的低阶操作系统控制代码的效率会比在不同地址空间上实现更高些。

20 世纪 90 年代初,单内核结构被认为是过时的。把 Linux 设计成为单内核结构而不是微内核,引起了无数的争议 (参见塔能鲍姆 -林纳斯辩论)。

现在,单核结构正倾向于设计不容易出错,所以它的发展会比微内核结构更迅速些。两个阵营中都有成功的案例。微核经常被用于机器人和医疗器械的嵌入式设计中,因为它的系统的关键部分都处在相互分开的,被保护的存储空间中。这对于单核设计来说是不可能的,就算它采用了运行时加载模块的方式。

尽管 Mach 是众所周知的多用途的微内核,人们还是开发了除此之外的几个微内核。L3 是一个演示性的内核,只是为了证明微内核设计并不总是低运行速度。它的后续版本 L4,甚至可以将 Linux 内核作为它的一个进程,运行在单独的地址空间。

QNX 是一个从 20 世纪 80 年代,就开始设计的微内核系统。它比 Mach 更接近微内核的理念。它被用于一些特殊的领域;在这些情况下,由于软件错误,导致系统失效是不允许的。例如航天飞机上的机械手,还有研磨望远镜镜片的机器,一点点失误就会导致上千美元的损失。

很多人相信,由于 Mach 不能够解决一些提出微内核理论时针对的问题,所以微内核技术毫无用处。Mach 的爱好者表明这是非常狭隘的观点,遗憾的是似乎所有人都开始接受这种观点。

47.3 Hybrid kernel

混合内核实质上是微内核,只不过它让一些微核结构运行在用户空间的代码运行在内核空间,这样让内核的运行效率更高些。这是一种妥协做法,设计者参考了微内核结构的系统运行速度不佳的理论。然而后来的实验证明,纯微内核的系统实际上也可以是高效率的。大多数现代操作系统遵循这种设计范畴,微软视窗就是一个典型的例子。另外还有 XNU,运行在苹果 Mac OS X 上的内核,也是一个混合内核。

混合内核的例子:

- Windows NT/2000/XP/Server 2003 以及 Windows Vista/7 等基于 NT 技术的微软视窗操作系统
- Mac OS X
- BeOS 内核
- DragonFly BSD
- ReactOS 内核
- XNU

一些人认为可以在运行时加载模块的单核系统和混合内核系统没有区别。这是不正确的,混合意味着它从单核和微核系统中都吸取了一定的设计模式,例如一些非关键的代码在用户空间运行,另一些在内核空间运行,单纯是为了效率的原因。

林纳斯·托瓦兹认为混合核心这种分类只是一种市场营销手法,因为它的架构实现与运作方式与宏内核并没有什么实质不同。

47.4 Exokernel

外内核系统,也被称为纵向结构操作系统,是一种比较极端的设计方法。

它的设计理念是让用户程序的设计者来决定硬件接口的设计。外内核本身非常的小,它通常只负责系统保护和系统资源复用相关的服务。

传统的内核设计 (包括单核和微核) 都对硬件作了抽象,把硬件资源或设备驱动程序都隐藏在硬件抽象层下。比方说,在这些系统中,如果分配一段物理存储,应用程序并不知道它的实际位置。

而外核的目标就是让应用程序直接请求一块特定的物理空间,一块特定的磁盘块等等。系统本身只保证被请求的资源当前是空闲的,应用程序就允许直接访问它。既然外核系统只提供了比较低级的硬件操作,而没有像其他系统一样提供高级的硬件抽象,那么就需要增加额外的运行库支持。这些运行库运行在外核之上,给用户程序提供了完整的功能。

理论上,这种设计可以让各种操作系统运行在一个外核之上,如 Windows 和 Unix。并且设计人员可以根据运行效率调整系统的各部分功能。

现在,外核设计还停留在研究阶段,没有任何一个商业系统采用了这种设计。

第 48 章

Linux 内核

Linux 内核(Linux Kernel)[?] 是 Linux 操作系统的内核,符合 POSIX 标准并以 C 语言开发。Linux 最早是由芬兰黑客林纳斯·托瓦兹为尝试在英特尔 x86 架构上提供自由免费的类 Unix 系统而开发的。该计划开始于 1991 年,林纳斯·托瓦兹当时在 Usenet 新闻组 comp.os.minix 发表帖子,这份著名的帖子标示着 Linux 计划的正式开始。

从技术上说 Linux 是一个内核,“内核”指的是一个提供硬件抽象层、磁盘及文件系统控制、多任务等功能的系统软件。一个内核不是一套完整的操作系统,一套基于 Linux 内核的完整操作系统叫作 Linux 操作系统(或是 GNU/Linux)。

Linux 实质上是一个宏内核,设备驱动程序可以完全访问硬件,而且 Linux 内的设备驱动程序可以方便地以模块化(modularize)的形式设置,并在系统运行期间可直接装载或卸载。

Linux 是用 C 语言中的 GCC 版开发的,还有几个用汇编语言 (用的是 GCC 的 “AT&T 风格”) 写的目标构架短段。

因为要支持扩展的 C 语言,GCC 在很长的时间里是唯一一个能正确编译 Linux 的编译器。在 2004 年,Intel 主张通过修改内核来使它的编译器能正确编译内核。在内核构建过程中 (这里指从源代码创建可启动镜像) 可以使用许多其他的语言,包括 Perl、Python 和多种脚本语言。有一些驱动可能是用 C++、Fortran 或其他语言写的,虽然这样是强烈不建议的,Linux 的官方构建系统仅仅支持 GCC 作为其内核和驱动的编译器。

在 Linux 系统中,vmlinux(vmlinuz)是一个包含 Linux kernel 的静态链接的可执行文件[1]。

一般来说,核心的位置会在文件系统的/root 目录下,然而 Bootloader 必须使用 BIOS 的硬盘驱动程序,在一些 i386 的机器上必须要放在前 1024 个磁柱内。为了克服这个限制,Linux 发行版鼓励用户创建一个扇区用来存放 bootloader 与核心相关的开机文件(例如 GRUB、LILO 与 SYSLINUX 等)。这个扇区通常会挂载到系统的/boot 上,这也是 FHS(Filesystem Hierarchy Standard0 标准内定义的。

Linux 内核是在 GNU 通用公共许可证第 2 版之下发布的(加上一些固件与各种非自由许可证)。原先托瓦兹将 Linux 置于一个禁止任何商业行为的条例之下,但之后改用 GNU 通用公共许可证第二版。该协议允许任何人对软件进行修改或发行,包括商业行为,只要其遵守该协议,所有基于 Linux 的软件也必须以该协议的形式发表,并提供源代码。

另外,关于 GPL v2 许可证争议的一个重点是 Linux 使用固件二进制包以支持某些硬件设备。理察·马修·斯托曼认为这些东西让 Linux 某部份成为非自由软件,甚至以此散布 Linux 更会破坏 GPL,因为 GPL 需要完全可获取的源代码。林纳斯·托瓦兹及 Linux 社区中的领导者,支持较宽松的许可证,不支持理察·马修·斯托曼的立场。社区中的 Linux-libre 提供完整的自由软件固件。

另一个争论点,就是加载式核心模块是否算是知识产权下的派生创作,意即 LKM 是否也受 GPL 约束?托瓦兹本人相信 LKM 仅用一部分“公开”的核心接口,因此不算派生创作,因此允许一些仅有二进制包裹的驱动程序或不以 GPL 声明的驱动程序用于核心。但也不是每个人都如此同意,且托瓦兹也同意很多 LKM 的确是纯粹的派生创作,也写下“基本上,核心模块是派生创作”这样的句子。另一方面托瓦兹也说过:

[1]vmlinux 若要用于除错时则必须要在开机前增加 symbol table。

有时候一些驱动程序原先并非为 Linux 设计，而是为其他操作系统而作（意即并非为 Linux 作的派生创作），这是个灰色地带……这“的确”是个灰色地带，而我个人相信一些模块可视为非 Linux 派生创作，是针对 Linux 设计，也因此不会遵守 Linux 订下的行为准则。

48.1 Loadable Kernel Module

可加载核心模块（Loadable kernel module，缩写为 LKM）[?] 又译为加载式核心模组、可装载模块、可加载内核模块，是一种目标文件（object file），在其中包含了能在操作系统内核空间运行的代码。

LKM 运行在核心基底（base kernel），通常是用来支持新的硬件、新的文件系统或是新增的系统调用（system calls）。当不需要时，它们也能从内存中被卸载，清出可用的内存空间。

Microsoft Windows 及类 UNIX 系统都支持 LKM 功能，只是在不同的操作系统中有不同的名称，比如 FreeBSD 称为核心加载模组（kernel loadable module，缩写为 kld），Mac OS X 称为核心扩充（kernel extension，缩写为 kext）。也有人称它为核心可加载模组（Kernel Loadable Modules，缩写为 KLM），或核心模组（Kernel Modules，KMOD）。

没有可加载模块时，操作系统需要将所有可能需要的功能，一次全加入核心之中。其中许多功能虽然占据着内存空间，但是从来没被使用过。这不但浪费内存空间，而且每次在增加新功能时，使用者需要重新编译整个内核，之后重新开机。可加载模组避免了以上的缺点，让操作系统可以在需要新功能时可以动态加载，减少开发及使用上的困难。

48.2 Dynamic Kernel Module Support

动态内核模块支持（Dynamic Kernel Module Support，缩写为 DKMS）[?] 是用来生成 Linux 的内核模块的一个框架，其源代码一般不在 Linux 内核源代码树。当新的内核安装时，DKMS 支持的内核设备驱动程序到时会自动重建。DKMS 可以用在两个方向：如果一个新的内核版本安装，自动编译所有的模块，或安装新的模块（驱动程序）在现有的系统版本上，而不需要任何的手动编译或预编译软件包需要。例如，这使得新的显卡可以使用在旧的 Linux 系统上。

DKMS 是由戴尔的 Linux 工程团队在 2003 年开发的，并已经被许多 Linux 发行版所包含。

DKMS 是以 GNU 通用公共许可证（GPL）v2 或以后的条款发布下的免费软件，DKMS 原生支持 RPM 和 DEB 软件包格式。

DKMS 旨在创建一个内核相关模块源可驻留的框架，以便在升级内核时可以很容易地重建模块。这将允许 Linux 供应商提供较低版本的驱动程序，而无需等待新内核版本发行，同时还可以省去尝试重新编译新内核模块的客户预期要完成的工作。

Oikawa 等人在 1996 年提出一种与 LKM 类似的动态核心模块（DKMs）技术。与 LKM 一样，DKMs 以文件的形式存储并能在系统运行过程中动态地加载和卸载。DKMs 由一个用户层的 DKM 服务器来管理，并非由内核来管理。当核心需要某模块时，由 DKM 服务器负责把相应的 DKM 加载；当核心的内存资源紧缺时，由 DKM 服务器负责卸载一个没有被使用的 DKM。缺点是所有的 DKM 是存储在本地系统上的，占用了大量宝贵的存储空间。

48.3 Preemptive Scheduling

抢占式多任务处理（Preemption）[?] 是计算机操作系统中的一种实现多任务处理（multi task）的方式，相对于协作式多任务处理而言。协作式环境下，下一个进程被调度的前提是当前进程主动放弃时间片；抢占式环境下，操作系统完全决定进程调度方案，操作系统可以剥夺耗时长的进程的时间片，转而提供给其它进程。

- 每个任务赋予唯一的一个优先级(有些操作系统可以动态地改变任务的优先级);
- 假如有几个任务同时处于就绪状态,优先级最高的那个将被运行;
- 只要有一个优先级更高的任务就绪,它就可以中断当前优先级较低的任务的执行;

48.4 Kernel Panic

在 Linux 中,内核错误 (Kernel panic) 是指操作系统在监测到内核系统内部无法恢复的错误,相对于在用户空间代码类似的错误。操作系统试图读写无效或不允许的内存地址是导致内核错误的一个常见原因。内核错误也有可能在遇到硬件错误或操作系统 BUG 时发生。在许多情况中,操作系统可以在内存访问违例发生时继续运行。然而,系统处于不稳定状态时,操作系统通常会停止工作以避免造成破坏安全和数据损坏的风险,并提供错误的诊断信息。

48.5 Kernel oops

在 Linux 上,oops 即 Linux 内核的行为不正确,并产生了一份相关的错误日志。许多类型的 oops 会导致内核错误,即令系统立即停止工作,但部分 oops 也允许继续操作,作为与稳定性的妥协。这个概念只代表一个简单的错误。

当内核检测到问题时,它会打印一个 oops 信息然后杀死全部相关进程。oops 信息可以帮助 Linux 内核工程师调试,检测 oops 出现的条件,并修复导致 oops 的程序错误。

Linux 官方内核文档中提到的 oops 信息被放在内核源代码 Documentation/oops-tracing.txt 中。通常 klogd 是用来将 oops 信息从内核缓存中提取出来的,然而,在某些系统上,例如最近的 Debian 发行版中,rsyslogd 代替了 klogd,因此,缺少 klogd 进程并不能说明 log 文件中缺少 oops 信息的原因。

若系统遇到了 oops,一些内部资源可能不再可用。即使系统看起来工作正常,非预期的副作用可能导致活动进程被终止。内核 oops 常常导致内核错误,若系统试图使用被禁用的资源。

Kernelloops 提到了一种用于收集和提交 oops 到 http://www.kerneloops.org/的软件,Kerneloops.org 同时也提供 oops 的统计信息。

计算机安全是一个非常公众化的主题,因为大量在内核中的错误可能成为潜在的安全漏洞,是否允许提升权限漏洞或拒绝服务攻击源漏洞。在过去一直有许多这样的缺陷被发现,并在 Linux 内核中被修补好。例如在 2012 年五月,SYSRET 指令被发现在 AMD 和英特尔处理器间在实现方面有差异,这个差异在 Windows、FreeBSD、XenServer 和 Solaris 这些主流操作系统会导致漏洞。2012 年六月,Linux 核心中该问题已被修复。

第 49 章

内核模块

在操作系统启动的过程中，内核用来驱动主机的硬件及配置。一般内核都是压缩文件，因此在启动过程中会把内核解压缩后才能加载到内存中。

现代操作系统内核都具有可读取模块化驱动程序的功能，所谓的模块化可以理解为“插件”，模块可能由硬件厂商提供，也可能由内核提供。一般情况下，Linux 的内核文件位于/boot/vmlinuz 或/boot/vmlinuz-version，内核解压缩所需的 RAM Disk 为/boot/initrd 或/boot/initrd-version。

Linux 内核整体结构包含了很多的模块，有两种方法将需要的功能包含进内核当中，分别是：

- 将所有的功能都编译进 Linux 内核，这样不会有版本不兼容的问题，不需要进行严格的版本检查，但同时也会导致生成的内核会很大，而且要在现有的内核中添加新的功能时，就需要编译整个内核。
- 将需要的功能编译成模块，在需要的时候动态地添加，这样模块本身不编译进内核，从而控制了内核的大小，而且模块一旦被加载后，将和其它的部分完全一样。缺点是可能会有内核与模块版本不兼容的问题，导致内核崩溃，而且还会造成内存的利用率比较低。

目前，Linux 内核采用的是模块化技术，这样的设计使得系统内核可以保持最小化，同时确保了内核的可扩展性与可维护性，模块化设计允许我们在需要时才将模块加载至内核，实现动态内核调整[?]。

内核模块一般位于/lib/modules/version/kernel 或/lib/modules/$(uname -r)/kernel 中，内核源码则位于/usr/src/linux 或/usr/src/kernels。内核被加载到系统中后会产生相关的记录如下：

- 内核版本：/proc/version
- 系统内核功能：/proc/sys/kernel

如果要增加新的内核模块，首先需要将其编译为模块，然后通过 modprobe 等命令来加载。如果希望系统开机自动挂载内核模块则需要将 modprobe 命令写入/etc/rc.sysinit 文件中。

Linux 内核模块文件的命名方式通常为 < 模块名称.ko>，CentOS 6.3 系统的内核模块被集中存放在`/lib/modules/`uname -r`/`目录下(这里，`uname -r`获得的信息为当前内核的版本号)。

与内核模块加载相关的配置文件是/etc/modules.conf 或 /etc/modprobe.conf[?]。在配置文件中，一般是写入模块的加载命令或模块的别名的定义等，比如在 modules.conf 中可能会发行类似的一行：

```
1 alias eth0 8139too
```

49.1 lsmod

lsmod 命令用来显示当前系统中加载的 Linux 内核模块状态，不是使用任何参数会显示当前已经加载的所有内核模块。输出的三列信息分别为模块名称、占用内存大小、是否在被使用，如果第三列为 0 则该模块可以随时卸载，非 0 则无法执行 modprobe 删除模块。

```
1 [root@localhost ~]# lsmod
2 Module           Size Used by
3 loop            27870 2
4 vfat            17411 0
5 fat             60923 1 vfat
6 nls_utf8        12557 2
```

```
isofs           39794 2
tcp_lp          12663 0
fuse            86889 3
...
```

lsmod 命令可以和 grep 命令结合使用，例如：

```
[root@localhost ~]# lsmod | grep mei
mei_me          18581 0
mei             76861 1 mei_me
```

另外还可以通过查看 /proc/modules 来知道系统已经加载的模块。

49.2 depmod

通过 depmod 命令可以了解内核提供的模块之间的依赖关系。所谓的依赖关系，指的是加载指定的模块之前，必须要首先加载其所依赖的其他模块，该模块加载后才能正常工作。

depmod 会依据相关目录的定义将所有模块进行分析，并将分析的结果写入 modules.dep 文件，而且这个文件还将影响到 modprobe 命令的使用。

内核模块目录的结构如下：

```
arch #与硬件平台有关的选项，例如CPU的等级等；
crypto #内核所支持的加密的技术，例如MD5或DES等；
drivers #一些硬件的驱动程序，例如显卡、网卡、PCI相关硬件等；
fs #内核所支持的文件系统，例如vfat、reiserfs等；
lib #函数库；
net #与网络有关的协议数据以及防火墙模块（net/ipv4/netfilter/*）等；
sound #与音效有关的模块。
```

Linux 使用/lib/modules/$(uname -r)/modules.dep 文件来记录模块之间的依赖关系，可以通过 depmod 命令来创建该文件，因此 Linux 内核是自动解决依赖关系的。

```
#为所有列在/etc/modprobe.conf 或/etc/modules.conf 中的所有模块创建依赖关系，并且写入到modules.dep文件；
[root@localhost beinan]# depmod -a
# 列出已挂载但不可用的模块；
[root@localhost beinan]# depmod -e
# 列出所有模块的依赖关系
[root@localhost beinan]# depmod -n
```

49.3 modprobe

modprobe 命令可以动态加载与卸载指定的内核模块，或是载入一组相依赖的模块。

modprobe 会根据 depmod 所产生的依赖关系，决定要载入哪些模块。若在载入过程中发生错误，在 modprobe 会卸载整组的模块。依赖关系是通过读取 /lib/modules/2.6.xx/modules.dep 得到的，而该文件是通过 depmod 建立的。

在/etc/modprobe.conf 文件中存在的内容形式如下：

```
alias scsi_hostadapter mptbase
alias scsi_hostadapter1 mptspi
...
```

其中，最后一列是模块名字，中间的是模块的别名。如果要通过模块的名字来查询它的别名，可以用下面的命令：

```
#modprobe -c
```

当然 modprobe 也有列出内核所有模块和移除模块的功能，其中 `modprobe -l` 是列出内核中所有的模块，包括已挂载和未挂载的。模块名是不能带有后缀的，实际上模块都是带有.ko 或.o 后缀。

通过 `modprobe -l` 可以查看到所需要的模块，然后根据需要来挂载，实际上 `modprobe -l` 读取的模块列表就位于`/lib/modules/`uname -r`` 目录中，这里 `uname -r` 是内核的版本。

```
[root@localhost ~]# modprobe ip_vs #动态加载ip_vs模块
[root@localhost ~]# lsmod |grep ip_vs #查看模块是否加载成功
[root@localhost ~]# modprobe -r ip_vs #动态卸载ip_vs模块
```

通过上述 modprobe 方式加载的内核模块仅在当前有效，计算机重启后并不会再次加载该模块，如果希望系统开机自动挂载内核模块则需要将 modprobe 命令写入/etc/rc.sysinit 文件中：

```
[root@localhost ~]# echo "modprobe ip_vs" >> /etc/rc.sysinit
```

当内核模块不再需要时可以通过将/etc/rc.sysinit 文件中的对应 modprobe 命令删除，但需要重启计算机才生效。此时，可以通过 `modprobe -r` 命令来立刻删除内核模块：

```
[root@localhost ~]# modprobe -r ip_vs
```

49.4 insmod

将模块插入内核中。

与 modprobe 相比，insmod 完全是由用户自行加载一个完整路径和文件后缀名的模块，并不会主动去分析模块依赖性，而 modprobe 则会去主动分析 modules.dep 后才加载模块，这样既可以克服模块的依赖性问题，而且还不需要了解模块的具体路径。

```
[root@localhost ~]# insmod xxx.ko
```

49.5 rmmod

将模块从内核中删除。

```
[root@localhost ~]# rmmod xxx.ko
```

49.6 modinfo

modinfo 命令可以查看内核模块信息，通过查看模块信息来判定这个模块的用途。

```
[root@localhost ~]# modinfo ip_vs
filename:   /lib/modules/3.13.8-200.fc20.x86_64/kernel/net/netfilter/ipvs/ip_vs.ko
license:    GPL
depends:    nf_conntrack,libcrc32c
intree:     Y
vermagic:   3.13.8-200.fc20.x86_64 SMP mod_unload
signer:     Fedora kernel signing key
sig_key:    68:89:8C:AD:02:C5:A1:BA:89:74:28:C5:5E:53:C8:F9:0C:BC:BB:8C
sig_hashalgo: sha256
parm:       conn_tab_bits:Set connections' hash size (int)
```

事实上，modinfo 除了可以查阅内核内的模块之外，还可以检查其他模块文件。

49.7 tree

查看当前目录的整个树结构。

```
[root@localhost ~]# tree -a
```

49.8 临时调整内核参数

Linux 内核参数随着系统的启动会被写入内存中，我们可以直接修改/proc 目录下的大量文件来调整内核参数，并且这种调整是立刻生效的。

```
# 开启内核路由转发功能（通过0或1设置开关）
[root@localhost ~]# echo "1" > /proc/sys/net/ipv4/ip_forward
[root@localhost ~]# echo "1" > echo "1" >t /proc/sys/net/ipv4/icmp_echo_ignore_all
```

```
# 调整所有进程总共可以打开的文件数量 （ 当大量的用户访问网站资源时可能会因该数字过小而导致错误 ）
[root@localhost ~]# echo "108248" >/proc/sys/fs/file-max
```

49.9 永久调整内核参数

可以通过 `man proc` 可以获得大量关于内核参数的描述信息。但以上通过直接修改/proc 相关文件的方式在系统重启后将不再有效，如果希望设置参数并永久生效可以修改/etc/sysctl.conf 文件，文件格式为选项 = 值，我们通过修改该文件将前面 3 个案例参数设置为永久有效：

```
[root@localhost ~]# vim /etc/sysctl.conf
net.ipv4.ip_forward = 1
net.ipv4.icmp_echo_ignore_all = 1
fs.file-max = 108248
```

注意，通过 sysctl.conf 文件修改的内核参数不会立刻生效，修改完成后使用 `sysctl -p` 命令可以使这些设置立刻生效。

49.10 内核模块程序结构

内核模块程序结构[?] 主要由加载、卸载函数功能函数等一系列声明组成。它可以被传入参数，也可以导出符号，供其它的模块使用。

49.10.1 模块加载函数

在用 insmod 或 modprobe 命令加载模块时，一般需要执行模块加载函数以完成模块的初始化工作。

Linux 内核的模块加载函数一般用 __init 标识声明，模块加载函数必须以 module_init(函数名) 的形式被指定。该函数返回整型值，如果执行成功，则返回 0，初始化失败时则返回错误编码，Linux 内核当中的错误编码是负值，在 <linux/errno.h> 中定义。

在 Linux 中，标识 __init 的函数在连接时放在.init.text 这个区段，而且在.initcall.init 中保留一份函数指针，初始化的时候内核会根据这些指针调用初始化函数，初始化结束后释放这些 init 区段（包括前两者）。

代码清单：

```
static int __init XXX_init(void)
{
  return 0;
}

moudle_init(XXX_init);
```

49.10.2 模块卸载函数

在用 rmmod 或 modprobe 命令卸载模块时，一般需要执行模块卸载函数来完成与加载相反的工作。

模块的卸载函数和模块加载函数实现相反的功能，主要包括：

- 若模块加载函数注册了 XXX，则模块卸载函数注销 XXX；
- 若模块加载函数动态分配了内存，则模块卸载函数释放这些内存；
- 若模块加载函数申请了硬件资源，则模块卸载函数释放这些硬件资源；
- 若模块加载函数开启了硬件资源，则模块卸载函数一定要关闭这些资源。

```
static void __exit XXX_exit(void)
{

}

moudle_exit(XXX_exit);
```

49.10.3 模块许可证声明

模块许可证必须声明，如果不声明，则在模块加载时会收到内核被污染的警告，一般应遵循 GPL 协议。

```
MODULE_LICENSE("GPL");
```

49.10.4 模块参数

模块参数是模块在被加载时传递给模块的值，本身应该是模块内部的全局变量，是可选的。

```
/*
* book.c
*/
#include <linux/init.h>
#include <linux/module.h>

static char *bookName = "Good Book.";
static int bookNumber = 100;

static int __init book_init(void)
{
  printk(KERN_INFO "Book name is %s\n", bookName);
  printk(KERN_INFO "Book number is %d\n", bookNumber);

   return 0;
}

static void __exit book_exit(void)
{
  printk(KERN_INFO "Book module exit.\n");
}

module_init(book_init);
module_exit(book_exit);
module_param(bookName, charp, S_IRUGO);
module_param(bookNumber, int, S_IRUGO);

MODULE_LICENSE("GPL");
```

在向内核插入模块的时候可以用以下方式，并且可以在内核日志中看到模块加载以后变量已经有了值。

```
[root@localhost ~]# rmod book.ko
[root@localhost ~]# insmod book.ko bookName="The World is Plat" bookNumber="100"
[root@localhost ~]# tail -n /var/log/kern.log
Book name is The World is Plat
Book number is 100
```

49.10.5 模块导出符号

模块导出符号是可选的。

使用模块导出符号，方便其它模块依赖于该模块，并使用模块中的变量和函数等。

在 Linux2.6 的内核中，/proc/kallsyms 文件对应着符号表，它记录了符号和符号对应的内存地址。对于模块而言，使用下面的宏可以导出符号。

```
EXPORT_SYMBOL(符号名);
```

或

```
EXPORT_GPL_SYMBOL(符号名);
```

49.10.6 模块信息

模块信息是可选的，用于存储模块的作者信息等。

49.11 模块使用计数

Linux 内核提供了 MOD_INC_USE_COUNT 和 MOD_DEC_USE_COUNT 宏来管理模块使用计数。但是对于内核模块而言，一般不会自己管理使用计数。

49.12 模块的编译

将下面的 Makefile 文件放在 book.c 同级的目录下，然后使用 `make` 命令或 `make all` 命令编译即可生成 book.ko 模块文件。

对应的 Makefile：

```
ifneq ($(KERNELRELEASE),)
mymodule_objs := book.o
obj-m := book.o
else
PWD := $(shell pwd)
KVER ?= $(shell uname -r)
KDIR := /usr/src/linux-headers-2.6.38-8-generic

all:
    $(MAKE) -C $(KDIR) M=$(PWD)

clean:
    rm -rf *.mod.c *.mod.o *.ko *.o *.tmp_versions *.order *symvers
endif
```

如果功能不编译成模块，则无法绕开 GPL，编译成模块后公司发布产品则只需要发布模块，而不需要发布源码。为了 Linux 系统能够支持模块，需要做以下的工作：

- 内核编译时选择“可以加载模块”,嵌入式产品一般都不需要卸载模块,则可以不选择“可卸载模块”
- 将我们的 ko 文件放在文件系统中
- Linux 系统实现了 insmod、rmmod 等工具
- 使用时可以用 insmod 手动加载模块,也可以修改/etc/init.d/rcS 文件,从而在系统启动的时候就加载模块。

Part V

Administration

第 50 章

Introduction

50.1 Superuser

In computing, the superuser[?] is a special user account used for system administration. Depending on the operating system, the actual name of this account might be: root, administrator, admin or supervisor. In some cases the actual name is not significant, rather an authorization flag in the user's profile determines if administrative functions can be performed.

In operating systems which have the concept of a superuser, it is generally recommended that most application work be done using an ordinary account which does not have the ability to make system-wide changes.

Most configuration, testing and maintenance of networked systems has the potential to adversely affect multiple systems. In an effort to prevent inexperienced and disruptive individuals from causing problems, operating systems network utilities often require superuser authority. For example stress testing, if done at inappropriate times or without clear understanding of the effects, can deny users access to portions or all of the services of multiple computers. This is more of a problem than the original implementers of some utilities envisioned since it is now common for novices to build systems they own and have the ability to use the superuser account.

50.1.1 UNIX and UNIX-like

In Unix-like computer operating systems, root is the conventional name of the user who has all rights or permissions (to all files and programs) in all modes (single- or multi-user). Alternative names include baron in BeOS and avatar on some Unix variants.[1] BSD often provides a toor (“root” backwards) account in addition to a root account.[2] Regardless of the name, the superuser always has user ID 0. The root user can do many things an ordinary user cannot, such as changing the ownership of files and binding to network ports numbered below 1024. The name ”root” may have originated because root is the only user account with permission to modify the root directory of a Unix system and this directory was originally considered to be root's home directory.

The first process bootstrapped in a Unix-like system, usually called init, runs with root privileges. It spawns all other processes directly or indirectly, which inherit their parents' privileges. Only a process running as root is allowed to change its user ID to that of another user; once it's done so, there is no way back. Doing so is sometimes called dropping root privileges and is often done as a security measure to limit the damage from possible contamination of the process. Another case is login and other programs that ask users for credentials and in case of successful authentication allow them to run programs with privileges of their accounts.

It is never good practice for anyone (including system administrators) to use root as their normal user account, since simple typographical errors in entering commands can cause major damage to the system. It is advisable to create a normal user account instead and then use the su command to switch when necessary. The sudo utility is preferred as it only executes a single command as root, then returns to the normal user.

Some operating systems, such as OS X and some Linux distributions, allow administrator accounts which provide greater access while shielding the user from most of the pitfalls of full root access. In some cases the root account is disabled by default, and must be specifically enabled. In mobile platform-oriented Unixes such as Apple

iOS and Android, the device's security systems must be cracked in order to obtain superuser access. In a few systems, such as Plan 9, there is no superuser at all.

50.1.2 Windows NT

In Windows NT and later systems derived from it (such as Windows 2000, Windows XP, Windows Server 2003, and Windows Vista/7/8), there must be at least one administrator account (Windows XP and earlier) or one able to elevate privileges to superuser (Windows Vista/7/8 via User Account Control).[4] In Windows XP and earlier systems, there is a built-in administrator account that remains hidden when a user administrator-equivalent account exists.[5] This built-in administrator account is created with a blank password.[5] This poses security risks, so the built-in administrator account is disabled by default in Windows Vista and later systems due to the introduction of User Account Control (UAC).

A Windows administrator account is not an exact analogue of the Unix root account - some privileges are assigned to the "Local System account". The purpose of the administrator account is to allow making system-wide changes to the computer (with the exception of privileges limited to Local System).

The built-in administrator account and a user administrator account have the same level of privileges. The default user account created in Windows systems is an administrator account. Unlike OS X, Linux, and Windows Vista/7/8 administrator accounts, administrator accounts in Windows systems without UAC do not insulate the system from most of the pitfalls of full root access. One of these pitfalls includes decreased resilience to malware infections. In Microsoft Windows 2000, Windows XP Professional, and Windows Server 2003, administrator accounts can be insulated from more of the these pitfalls by changing the account from the administrator group to the power user group in the user account properties[6] but this solution is not as effective as using newer Windows systems with UAC.

In Windows Vista/7/8 administrator accounts, a prompt will appear to authenticate running a process with elevated privileges. No user credentials are required to authenticate the UAC prompt in administrator accounts but authenticating the UAC prompt requires entering the username and password of an administrator in standard user accounts. In Windows XP (and earlier systems) administrator accounts, authentication is not required to run a process with elevated privileges and this poses another security risk that lead to the development of UAC. Users can set a process to run with elevated privileges from standard accounts by setting the process to "run as administrator" or using the "runas" command and authenticating the prompt with credentials (username and password) of an administrator account. Much of the benefit of authenticating from a standard account is negated if the administrator account's credentials being used has a blank password (as in the built-in administrator account in Windows XP and earlier systems).

In Windows Vista/7/8, the root user is TrustedInstaller. In Windows NT, 2000, and XP, the root user is System.

50.1.3 Other

In Novell NetWare, the superuser was called "upervisor", later "admin". In OpenVMS, "SYSTEM" is the superuser account for the operating system.

Many older operating systems on computers intended for personal and home use, including MS-DOS and Windows 95, do not have the concept of multiple accounts and thus have no separate administrative account; anyone using the system has full privileges. The lack of this separation in these operating systems has been cited as one major source of their insecurity.

超级用户(Superuser)在计算机操作系统领域中指一种用于进行系统管理的特殊用户,其在系统中的实际名称也因系统而异,如 root、administrator 与 supervisor。为了使病毒、恶意软件与普通的用户错误不对整个系统产生不利的影响,在系统里日常任务都是由无法进行全系统变更的普通用户账户所完成。

在 Unix 与类 Unix 系统中,root 是在所有模式(单/多用户)下对拥有对所有文件与程序拥有一切权限的用户(也即超级用户)的约定俗成的通名,但也有例外,如在 BeOS 中超级用户的实名是 baron,在其他一些 Unix 派生版里则以 avatar 作为超级用户的实名 [1],而 BSD 中一般也提供“toor”账户(也即“root”的反写)作为 root 账户的副本 [2],但无论实名为何,超级用户的用户 ID(UID)一般都为 0。root 用户可以进行许多普通用户无法做的操作,如更改文件所有者或绑定编号于 1024 之下的网络端口。之所以将“root”设定为超级用户之名,可能是因为 root 是唯一拥有修改 UNIX 系统根目录(root directory)的权限的用户,而根目录最初就被认为是 root 的家目录一般的存在。

在类 Unix 系统引导过程中引导的第一个程序(常被称为 init)就是以 root 权限运行的,其他所有进程都由其直接或间接派生而出,并且这些进程都继承了各自的父进程的权限。只有以 root 权限运行的进程才能将自己的 UID 修改为其他用户对应的 UID,且对应 UID 在修改完成之后无法改回,这种行为有时也被称为丢弃 root 权限,其目的主要是为了安全考虑 -(在进程出错等情况下)降低进程污染所造成的损失。另一种情况是,用户登录后,有些程序会向用户请求认证提升权限,当认证成功后用户就能以其账户所对应的权限来执行程序。

对任何人(也包括系统管理员自己)来说,将 root 当作一般用户账户使用都绝不是一个好习惯,因为即使是输入命令时的微小错误都可能对系统造成严重破坏,因而相较之下较为明智的做法是创建一个普通用户账户用作日常使用,需要 root 权限时再用 su 切换到 root 用户。sudo 工具也是个暂时性获取 root 权限的替代方法。

在 Mac OS X 与一些 Linux 发行版中则允许管理员账户(这与 root 这样的全权账户有别)拥有更多的权限,同时也屏蔽掉大部分容易(因误操作)造成损害的 root 权限。某些情况下,root 账户是被默认禁用的,需要时必须另外启用。另外在极少数系统(如 Plan 9)中,系统中根本没有超级用户。

某些软件缺陷能使用户获得 root 权限(即提升权限来以 root 权限执行用户提供的代码),这会造成严重的计算机安全问题,相对应的修复这些软件则是系统安全维护的重要组成部分。让某个正以超级用户权限运行的程序的缓冲区溢出(某些情况下亦称缓冲区攻击)是一种常见的(非法)获得 root 权限的方式,在现代的操作系统中则一般采取将关键程序(如网络服务器程序)运行于一个特别的限权用户之下的方式来预防这种情况的发生。

在 Windows NT 及其派生的后继系统(如 Windows 2000,Windows XP,Windows Server 2003 及 Windows Vista/7),系统里要么至少有一个管理员账户(Windows XP 及更早的系统),要么可以使用用户账户控制(User Account Control,简称 UAC)机制提升到超级用户权限(Windows Vista/7)。在 Windows XP 及更早的系统中有内建一个初始密码为空的管理账户(实名为“Administrator”),此账户在存在其他有系统管理权限的用户账户时是默认隐藏的 [6];显而易见的,这种做法会带来安全问题,所以在 Windows Vista 与其后继系统中默认禁止了内建的 Administrator 账户,并引入 UAC 机制取而代之。

Windows 的 Administrator 与 Unix 的 root 账户不尽相同 -Windows 将部分权限分配给了“本地系统账户”,Administrator 账户的目的只是为了允许在计算机上进行全系统范围(包括本地系统账户权限所不能及之范围)的更改。

Windows 内建的 Administrator 账户与一般的管理员用户享有同等的权限,Windows 系统中默认创建的用户账户也拥有管理权限,而与 Mac OS X,Linux 与 Windows Vista/7 的管理员账户不同的是,Windows 中无 UAC 限制的 Administrator 账户无法将系统与超级用户权限容易造成的损害(如降低对恶意软件的抗力)相隔离。在 Windows 2000,Windows XP 专业版与 Windows Server 2003 中,管理员可用在账户属性中将账户所属组由管理员组更改为权力用户(Power User)组的方式来解决这一问题 [9],但这一解决方法不如使用带有 UAC 机制的新版 Windows 系统有效。

对于 Windows XP(及早期的系统)的管理员账户来说,提升权限执行程序并不需认证;这种做法有相当的安全隐患,解决这一问题也是开发 UAC 的目的之一。与之相对的,在 Windows Vista/7 中,管理

员账户提升权限运行进程的时候会有确认提示，但不需要进行用户资格认证，而普通用户账户则需在提示框内输入任一管理员账户的用户名和密码才能通过认证；具体来说，用户可以以将进程设置为“以管理员权限运行”或使用“runas”命令并用任一管理员账户的用户名和密码进行资格认证的方式在普通用户下提升权限运行进程，但如若提权运行认证时所使用的管理员账号对应的密码为空（就像 XP 及早期系统内建的 Administrator 账户一样）时，认证机制的意义就大减了。

在 Novell NetWare 中，超级用户的实名是“Supervisor”，后来又改为“admin”。另外，许多早期的针对个人/家庭应用而设计的操作系统（如 MS-DOS 与 Windows 9x）上并没有多用户的概念，因而也没有单独的管理账户，所有使用系统的人都有所有管理特权，这种没有分隔权限的情况也被认为是这些系统的重要安全隐患之一。

50.2 su

The su command, also referred to as substitute user, super user, or switch user, allows a computer operator to change the current user account associated with the running virtual console.

By default, and without any other command line argument, this will elevate the current user to the superuser of the local system.

When run from the command line, su asks for the target user's password, and if authenticated, grants the operator access to that account and the files and directories that account is permitted to access.

```
theqiong@localhost:~$ su
Password:
root@localhost:/home/john# exit
logout
john@localhost:~$
```

Additionally, one can switch to another user who is not the superuser; e.g. su jane.

```
theqiong@localhost:~$ su jane
Password:
theqiong@localhost:/home/john$ exit
logout
john@localhost:~$
```

It should generally be used with a hyphen by administrators (su -, which is identical to su - root), which can be used to start a login shell. This way users can assume the user environment of the target user:

```
john@localhost:~$ su – jane
Password:
jane@localhost:~$
```

A related command called sudo executes a command as another user but observes a set of constraints about which users can execute which commands as which other users (generally in a configuration file named /etc/sudoers, best editable by the command visudo). Unlike su, sudo authenticates users against their own password rather than that of the target user (to allow the delegation of specific commands to specific users on specific hosts without sharing passwords among them and while mitigating the risk of any unattended terminals).

Some Unix-like systems have a wheel group of users, and only allow these users to su to root. This may or may not mitigate these security concerns, since an intruder might first simply break into one of those accounts. GNU su, however, does not support a wheel group for philosophical reasons. Richard Stallman argues that because a wheel group would prevent users from utilizing root passwords leaked to them, the group would allow existing admins to ride roughshod over ordinary users.

su 命令也被称为“替代用户”、“超级用户”或“切换用户”,是可以让计算机操作者在虚拟控制台切换当前用户帐户的命令。没有其他命令行参数时,默认将会将当前用户提权至本地超级用户。

在命令行中运行时,su 会要求目标用户的密码。如果验证通过,操作者将会授予该帐户的权限,并且允许访问该帐户可以访问的文件和目录。

```
john@localhost:~$ su
密码:
root@localhost:/home/john# exit
登出
john@localhost:~$
\begin{Verbatim}
```

此外,还可以切换到另一个不是超级用户的帐户,例如 `su jane`。

```
\begin{Verbatim}
john@localhost:~$ su jane
密码:
jane@localhost:/home/john$ exit
登出
john@localhost:~$
\begin{Verbatim}
```

一般来说,管理员应该使用一个连字号(`su -`,等同于 `su - root`),来启动登录 `shell`。这样,用户可以获得目标用户的用户环境:

```
\begin{Verbatim}
john@localhost:~$ su - jane
密码:
jane@localhost:~$
\begin{Verbatim}
```

相关的命令 `sudo` 也可以允许以另一个用户的身份执行命令,但遵守一组的限制,以决定哪些用户可以以什么用户身份执行什么命令(通常是

一些类 `Unix` 系统有 `wheel` 组,且只允许组内用户 `su` 到 `root`。很难说这是否会降低安全风险,因为入侵者可能会轻易入侵其中一个

```
\section{sudo}


sudo\cite{sudo} is a program for Unix-like computer operating systems that allows users to

Sudo (su "do") allows a system administrator to delegate authority to give certain users (

\begin{compactitem}
\item The ability to restrict what commands a user may run on a per-host basis.
\item Sudo does copious logging of each command, providing a clear audit trail of who did
\item Sudo uses timestamp files to implement a ``ticketing'' system. When a user invokes su
\item Sudo's configuration file, the sudoers file, is setup in such a way that the same su
\end{compactitem}
```

The program was originally written by Robert Coggeshall and Cliff Spencer ``around 19

Unlike the \texttt{su} command, users typically supply their own password to sudo rat

The /etc/sudoers file allows listed users access to execute a huge amount of configur

\begin{compactitem}
\item Enabling root commands only from the invoking terminal;
\item Not requiring a password for certain commands;
\item Requiring a password per user or group;
\item Requiring re-entry of a password every time or never requiring a password at al
\end{compactitem}

It can also be configured to permit passing arguments or multiple commands, and even

By default the user's password can be retained through a grace period (15 minutes per

sudo is able to log each command run. Where a user attempts to invoke sudo without be

In some cases sudo has completely supplanted the superuser login for administrative t

visudo is a command-line utility that allows editing of the /etc/sudoers file in a sa

The runas command provides similar functionality in Microsoft Windows, but it cannot

There exist several frontends to sudo for use in a GUI environment, notably kdesudo,

\clearpage

Sudo(substitute user [或 superuser] do)是一种程序,用于类 Unix 操作系统如 BSD、Mac OS X 以及 GNU/

在 sudo 于 1980 年前后被写出之前,一般用户管理系统的方式是利用 su 切换为超级用户,只是使用 su 的缺点之一在于必须

sudo 使一般用户不需要知道超级用户的密码即可获得权限。首先超级用户将普通用户的名字、可以执行的特定命令、按照哪种用户或

在一般用户需要取得特殊权限时,其可在命令前加上“sudo”,此时 sudo 将会询问该用户自己的密码(以确认终端机前的是该用户本

由于不需要超级用户的密码,部分 Unix 系统甚至利用 sudo 使一般用户取代超级用户作为管理账号,例如 Ubuntu、Mac OS X

\begin{Verbatim}
语法:
sudo [-bhHpV][-s][-u < 用户>][指令]
或

`sudo [-klv]`

参数

`-b` 在后台执行指令。

`-h` 显示帮助。

`-H` 将 `HOME` 环境变量设为新身份的 `HOME` 环境变量。

`-k` 结束密码的有效期限，也就是下次再执行 `sudo` 时便需要输入密码。

`-l` 列出目前用户可执行与无法执行的指令。

`-p` 改变询问密码的提示符号。

`-s` 执行指定的 `shell`。

`-S` 从标准输入流替代终端来获取密码

`-u <用户>` 以指定的用户作为新的身份。若不加上此参数，则预设以 `root` 作为新的身份。

`-v` 延长密码有效期限 `5` 分钟。

`-V` 显示版本信息。

50.3 root

Android rooting is the process of allowing users of smartphones, tablets, and other devices running the Android mobile operating system to attain privileged control (known as "root access") within Android's sub-system.

Rooting lets all user-installed applications run privileged commands typically unavailable to the devices in the stock configuration. Rooting is required for more advanced and potentially dangerous operations including modifying or deleting system files, removing carrier- or manufacturer-installed applications, and low-level access to the hardware itself (rebooting, controlling status lights, or recalibrating touch inputs.)

The process of rooting varies widely by device, but usually includes exploiting one or more security bugs in the firmware of (i.e., in the version of the Android OS installed on) the device. Once an exploit is discovered, a custom recovery image can be flashed which will skip the digital signature check of firmware updates. Then a modified firmware update can be installed which typically includes the utilities needed to run apps as root.

A typical rooting installation also installs the Superuser application, which supervises applications that are granted root or superuser rights. For example, the `su` binary can be copied to a location in the current process' PATH (e.g., `/system/xbin/`) and granted executable permissions with the `chmod` command. A supervisor application, like SuperUser or SuperSU, can then regulate and log elevated permission requests from other applications.

The process of rooting a device may be simple or complex, and it even may depend upon serendipity. For example, shortly after the release of the HTC Dream (HTC G1), it was discovered that anything typed using the keyboard was being interpreted as a command in a privileged (root) shell. Although Google quickly released a patch to fix this, a signed image of the old firmware leaked, which gave users the ability to downgrade and use the original exploit to gain root access. By contrast, the Google-branded Android phones, the Nexus One, Nexus S, Galaxy Nexus, Nexus 4 and Nexus 5, as well as their tablet counterparts, the Nexus 7 and Nexus 10, can be boot-loader unlocked by simply connecting the device to a computer while in boot-loader mode and running the Fastboot program with the command fastboot oem unlock. After accepting a warning, the boot-loader is unlocked, so a new system image can be written directly to flash without the need for an exploit.

A secondary operation, unlocking the device's bootloader verification, is required to remove or replace the installed operating system. In contrast to iOS jailbreaking, rooting is not needed to run applications distributed outside of the Google Play Store, sometimes called sideloading. The Android OS supports this feature natively in two ways: through the "Unknown sources" option in the Settings menu and through the Android Debug Bridge. However some carriers, like AT&T, prevent the installation of applications not on the Store in firmware, although several devices (including the Samsung Infuse 4G) are not subject to this rule, and AT&T has since lifted the restriction on several older devices.

As of 2012 the Amazon Kindle Fire defaults to the Amazon Appstore instead of Google Play, though like most other Android devices, Kindle Fire allows sideloading of applications from unknown sources, and the ”easy installer” application on the Amazon Appstore makes this easy. Other vendors of Android devices may look to other sources in the future. Access to alternate apps may require rooting but rooting is not always necessary. Rooting an Android phone lets the owner modify or delete the system files, which in turn lets them perform various tweaks and use apps that require root access.

Rooting is often performed with the goal of overcoming limitations that carriers and hardware manufacturers put on some devices, resulting in the ability to alter or replace system applications and settings, run specialized apps that require administrator-level permissions, or perform other operations that are otherwise inaccessible to a normal Android user. On Android, rooting can also facilitate the complete removal and replacement of the device's operating system, usually with a more recent release of its current operating system.

As Android derives from the Linux kernel, rooting an Android device gives similar access administrative permissions as on Linux or any other Unix-like operating system such as FreeBSD or OS X.

Root access is sometimes compared to jailbreaking devices running the Apple iOS operating system. However, these are different concepts. Jailbreaking describes the bypass of several types of Apple prohibitions for the end user: modifying the operating system (enforced by a “locked bootloader”), installing non-officially approved apps via sideloading, and granting the user elevated administration-level privileges. Only a minority of Android devices lock their bootloaders—and many vendors such as HTC, Sony, Asus and Google explicitly provide the ability to unlock devices, and even replace the operating system entirely. Similarly, the ability to sideload apps is typically permissible on Android devices without root permissions. Thus, it is primarily the third aspect of iOS jailbreaking relating to giving users superuser administrative privileges that most directly correlates to Android rooting.

In the past, many manufacturers have tried to make “unrootable” phones with harsher protections (like the Droid X), but they're usually still rootable in some way, shape, or form. There may be no root exploit available for new or recently updated phones, but one is usually available within a few months.

In 2011, Motorola, LG Electronics and HTC added security features to their devices at the hardware level in an attempt to prevent users from rooting retail Android devices.[citation needed] For instance, the Motorola Droid X has a security boot-loader that puts the phone in ”recovery mode” if a user loads unsigned firmware onto the device, and the Samsung Galaxy S II displays a yellow triangle indicator if the device firmware has been modified.

root 通常是针对 Android 系统的手机而言，它使得用户可以获取 Android 操作系统的超级用户权限。root 通常用于帮助用户越过手机制造商的限制，使得用户可以卸载手机制造商预装在手机中某些应用，以及运行一些需要超级用户权限的应用程序。

原始出厂的手机并未开放 root 权限，获取 root 的方法都是不受官方支持的，目前获取 root 的方法都是利用系统漏洞实现的。

不同手机厂商可能存在的漏洞不同，也就导致了不同手机 root 的原理可能不同。为了让用户可以控制 root 权限的使用，防止其被未经授权的应用所调用，通常还有一个 Android 应用程序来管理 su 程序的行为。不过，不管采用什么原理实现 root，最终都需要将 su 可执行文件和对应的 Android 管理应用复制到 Android 系统的`/system` 分区下 (例如`/system/xbin/su`) 并用 chmod 命令为其设置可执行权限和 setuid 权限。

目前最广泛利用的系统漏洞是 zergRush，该漏洞适用于 Android 2.2-2.3.6 的系统，其它的漏洞还有 Gingerbreak, psneuter 等。其中，zergRush 漏洞必须在 adb shell 下运行，而 adb shell 只能将手机用 USB 数据线与 PC 连接之后才能在 PC 上打开，因此目前常用的 root 工具都是 PC 客户端程序，亦有部分工具能直接在 Android 设备上运行。

在 Android 设备上直接运行的 root 工具中，部分以 App 的形式在各类应用商店(非 Google 官方)上发布，供用户下载使用。这些工具通常通常采用傻瓜化操作，即用户只需按下一个按钮就可以等应用来获取 root 权限。利用这些应用获取 root 权限之后，应用本身就会成为 root 权限的授权者，其他应用使用 root 权限时都需要通过此授权者的允许。有些应用(如百度一键 root)会在通知栏推送广告，还有一些应用在获取 root 权限之后将自己变成系统应用，就像厂商预装的那样。

50.4 sa

A system administrator[?], or sysadmin, is a person who is responsible for the upkeep, configuration, and reliable operation of computer systems; especially multi-user computers, such as servers.

The system administrator seeks to ensure that the uptime, performance, resources, and security of the computers he or she manages meet the needs of the users, without exceeding the budget.

To meet these needs, a system administrator may acquire, install, or upgrade computer components and software; automate routine tasks; write computer programs; troubleshoot; train and/or supervise staff; and provide technical support.

The duties of a system administrator are wide-ranging, and vary widely from one organization to another. Sysadmins are usually charged with installing, supporting, and maintaining servers or other computer systems, and planning for and responding to service outages and other problems. Other duties may include scripting or light programming, project management for systems-related projects.

Many organizations staff other jobs related to system administration. In a larger company, these may all be separate positions within a computer support or Information Services (IS) department. In a smaller group they may be shared by a few sysadmins, or even a single person.

- A database administrator (DBA) maintains a database system, and is responsible for the integrity of the data and the efficiency and performance of the system.
- A network administrator maintains network infrastructure such as switches and routers, and diagnoses problems with these or with the behavior of network-attached computers.
- A security administrator is a specialist in computer and network security, including the administration of security devices such as firewalls, as well as consulting on general security measures.
- A web administrator maintains web server services (such as Apache or IIS) that allow for internal or external access to web sites. Tasks include managing multiple sites, administering security, and configuring necessary components and software. Responsibilities may also include software change management.
- A computer operator performs routine maintenance and upkeep, such as changing backup tapes or replacing failed drives in a RAID. Such tasks usually require physical presence in the room with the computer; and while less skilled than sysadmin tasks require a similar level of trust, since the operator has access to possibly sensitive data.
- A postmaster administers a mail server.
- A Storage (SAN) Administrator. Create, Provision, Add or Remove Storage to/from Computer systems. Storage can be attached local to the system or from a Storage Area Network (SAN) or Network Attached Storage (NAS). Create File Systems from newly added storage.

In some organizations, a person may begin as a member of technical support staff or a computer operator, then gain experience on the job to be promoted to a sysadmin position.

Most important skills to a system administrator is problem solving. This can some times lead into all sorts of constraints and stress. When a workstation or server goes down, the sysadmin is called to solve the problem. They should be able to quickly and correctly diagnose the problem. They must figure out what is wrong and how best it can be fixed in a short time.

The subject matter of system administration includes computer systems and the ways people use them in an organization. This entails a knowledge of operating systems and applications, as well as hardware and software troubleshooting, but also knowledge of the purposes for which people in the organization use the computers.

Perhaps the most important skill for a system administrator is problem solving—frequently under various sorts of constraints and stress. The sysadmin is on call when a computer system goes down or malfunctions, and must be able to quickly and correctly diagnose what is wrong and how best to fix it. They may also need to have team work and communication skills; as well as being able to install and configure hardware and software.

System administrators are not software engineers or developers. It is not usually within their duties to design or write new application software. However, sysadmins must understand the behavior of software in order to deploy it and to troubleshoot problems, and generally know several programming languages used for scripting or automation of routine tasks.

Particularly when dealing with Internet-facing or business-critical systems, a sysadmin must have a strong grasp of computer security. This includes not merely deploying software patches, but also preventing break-ins and other security problems with preventive measures. In some organizations, computer security administration is a separate role responsible for overall security and the upkeep of firewalls and intrusion detection systems, but all sysadmins are generally responsible for the security of computer systems.

A system administrator's responsibilities might include:

- Analyzing system logs and identifying potential issues with computer systems.
- Introducing and integrating new technologies into existing data center environments.
- Performing routine audits of systems and software.
- Performing backups.
- Applying operating system updates, patches, and configuration changes.
- Installing and configuring new hardware and software.
- Adding, removing, or updating user account information, resetting passwords,etc.
- Answering technical queries and assisting users.
- Responsibility for security.
- Responsibility for documenting the configuration of the system.
- Troubleshooting any reported problems.
- System performance tuning.
- Ensuring that the network infrastructure is up and running.
- Configure, Add, Delete File Systems. Knowledge of Volume management tools like Veritas (now Symantec), Solaris ZFS, LVM.
- The system administrator is responsible for following things:
- User administration (setup and maintaining account)
- Maintaining system
- Verify that peripherals are working properly
- Quickly arrange repair for hardware in occasion of hardware failure
- Monitor system performance
- Create file systems
- Install software
- Create a 'backup'and recover policy
- Monitor network communication
- Update system as soon as new version of OS and application software comes out
- Implement the policies for the use of the computer system and network
- Setup security policies for users. A sysadmin must have a strong grasp of computer security (e.g.
- firewalls and intrusion detection systems)
- Documentation in form of internal wiki
- Password and identity management

In larger organizations, some of the tasks above may be divided among different system administrators or members of different organizational groups. For example, a dedicated individual(s) may apply all system upgrades, a Quality Assurance (QA) team may perform testing and validation, and one or more technical writers may be responsible for all technical documentation written for a company. System administrators, in larger organizations, tend not to be systems architects, system engineers, or system designers.

In smaller organizations, the system administrator might also act as technical support, Database Administrator, Network Administrator, Storage (SAN) Administrator or application analyst.

Unlike many other professions, there is no single path to becoming a system administrator. Many system administrators have a degree in a related field: computer science, information technology, computer engineering, information systems, or even a trade school program. On top of this, nowadays some companies require an IT certification. Other schools have offshoots of their Computer Science program specifically for system administration.

Some schools have started offering undergraduate degrees in System Administration. The first, Rochester Institute of Technology started in 1992. Others such as Rensselaer Polytechnic Institute, University of New Hampshire, [2] Marist College, and Drexel University have more recently offered degrees in Information Technology. Symbiosis Institute of Computer Studies and Research (SICSR) in Pune, India offers Masters degree in Computers Applications with a specialization in System Administration. The University of South Carolina[1] offers an Integrated Information Technology B.S. degree specializing in Microsoft product support.

As of 2011, only five U.S. universities, Rochester Institute of Technology,[3] Tufts,[4] Michigan Tech, and Florida State University [5] have graduate programs in system administration.[citation needed] In Norway, there is a special English-taught MSc program organized by Oslo University College [6] in cooperation with Oslo University, named "Masters programme in Network and System Administration." There is also a "BSc in Network and System Administration" [7] offered by Gjøvik University College. University of Amsterdam (UvA) offers a similar program in cooperation with Hogeschool van Amsterdam (HvA) named "Master System and Network Engineering". In Israel, the IDF's ntmm course in considered a prominent way to train System administrators.[8] However, many other schools offer related graduate degrees in fields such as network systems and computer security. One of the primary difficulties with teaching system administration as a formal university discipline, is that the industry and technology changes much faster than the typical textbook and coursework certification process. By the time a new textbook has spent years working through approvals and committees, the specific technology for which it is written may have changed significantly or become obsolete.

In addition, because of the practical nature of system administration and the easy availability of open-source server software, many system administrators enter the field self-taught. Some learning institutions are reluctant to, what is in effect, teach hacking to undergraduate level students.

Generally, a prospective will be required to have some experience with the computer system he or she is expected to manage. In some cases, candidates are expected to possess industry certifications such as the Microsoft MCSA, MCSE, MCITP, Red Hat RHCE, Novell CNA, CNE, Cisco CCNA or CompTIA's A+ or Network+, Sun Certified SCNA, Linux Professional Institute among others.

Sometimes, almost exclusively in smaller sites, the role of system administrator may be given to a skilled user in addition to or in replacement of his or her duties. For instance, it is not unusual for a mathematics or computing teacher to serve as the system administrator of a secondary school.

系统管理员(System Administrator, 简称 SA)是负责管理计算机系统的人。其具体的含义因环境而异。

拥有庞大复杂的计算机系统的组织, 通常按照专长分派计算机员工, 其中系统管理员负责维护现有的计算机系统, 因此其工作不同于:

- 系统设计师(Systems design, SD)
- 系统分析师(Systems analyst, SA)
- 数据库管理员(Database administrator, DBA)
- 存储管理员 (Storage Administrator)
- 程序员(Programmer, PG)
- 技术支持(Technical support), 或信息技术(Information technology, IT)
- 网络管理员(Network administrator)
- Security administrator
- 网站管理员(Web administrator, Webmaster)

而在较小的企业和组织中, 没有这么多不同的专家。系统管理员的称呼含义更加广泛 - 系统管理员就是那个了解计算机系统如何工作, 并能处理有关问题的人。

第 51 章

User

第 52 章

User Group

第 53 章

login

第 54 章

root

第 55 章

Permission

第 56 章

SELinux

Part VI

Shell

第 57 章

Introduction

在计算机科学中，shell[1] 起源于 Multics 计划，具体是指“提供用户使用接口”的软件，通常指的是命令行界面的解析器。一般来说，Shell 是指操作系统中提供访问内核所提供之服务的程序，也用于泛指所有为用户提供操作界面的程序，也就是程序和用户交互的层面。因此与之相对的是程序内核（Core），内核不提供和用户的交互功能，而是通过 Shell 来提供。

通常将 Shell 分为两类：命令行与图形界面，其中命令行壳层提供一个命令行界面（CLI），而图形壳层提供一个图形用户界面（GUI），因此 shell 也可以用来指代应用软件或任何在特定组件外围的软件，例如浏览器或电子邮件软件是 HTML 排版引擎的 shell。

普通意义上的 shell 就是可以接受用户输入命令的程序，它之所以被称作 shell 是因为它隐藏了操作系统低层的细节。同样的 Unix 下的图形用户界面 GNOME 和 KDE，有时也被叫做“虚拟 shell”或“图形 shell”。

- 图形用户界面（GUI shell）通常会建构在视窗系统上，例如 X Window System 有独立的 X 窗口管理器以及依靠窗口管理器的完整桌面环境。
- 命令行界面（CLI shell）包括：
  - Thompson shell(sh, UNIX Version 1 ～ Version 6)
  - Bourne shell(sh, UNIX Version 7)
  - Bourne-Again shell(bash)
  - Korn shell(ksh)
  - Z shell(zsh)
  - C shell(csh)
  - TENEX C shell(tcsh)
  - COMMAND.COM(DOS shell)
  - cmd.exe(windows shell)
  - Windows PowerShell

  以及 linux 系统上的：‘/etc/shells’

Unix 上的第一个 Unix shell 是肯·汤普逊（Ken Thompson）以 Multics 上的 shell 为模板而为 Unix 所开发的 sh。

Unix shell 也叫做命令行界面，它是 Unix 操作系统下传统的用户和计算机的交互界面。用户直接输入命令来执行任务，微软的 Windows 操作系统也提供了这样的功能，它们是 Windows 95/98 下的 command.com 和 Windows NT 系统下的 cmd.exe。

Unix 操作系统下的 shell 既是用户交互的界面，也是控制系统的脚本语言。具体而言，shell 仍然是控制操作系统启动、X Window 启动和很多其他实用工具的脚本解释程序。

57.1 Usage

在终端下，可以使用 tab 键自动完成来加快输入速度，如果想更快一点可能就需要熟悉一下终端下的快捷键。

57.1.1 Move Commmands

`<C-f>` 向前一个字符
`<C-b>` 向后一个字符
`<alt-f>` 向前一个单词
`<alt-b>` 向后一个单词
`<C-a>` 跳转到命令行首
`<C-e>` 跳转到命令行尾

57.1.2 Copy/Paste Commands

`<C-u>` 剪切光标之前的内容
`<C-k>` 剪切光标之后的内容
`<C-w>` 剪切光标之前的一个单词
`<C-y>` 粘贴终端下最后剪切的字符串, 到当前光标位置

如果需要粘贴系统剪切板中的字符串, 可以使用 Ctrl + Shift + V, 同时适用于 VIM。

57.1.3 Histroy Commands

`<C-p>` 上一个历史命令
`<C-n>` 下一个历史命令
`<C-r>` 搜索历史命令

57.1.4 Virtual Terminal

`Ctrl + Alt + Fn` 切换虚拟终端 Fn (n 为 1~6, 代表第几个终端)

系统启动过程中, 在完成了系统所有服务的启动后, init 启动终端模拟程序就产生了第一个终端 (当前终端或图形界面) Ctrl + Alt + F1(有的系统使用的是 Ctrl + Alt + F7, 例如 openSUSE)。

另外, 还有一些终端快捷键如下:

`<C-c>` 终止命令
`<C-d>` 退出 shell 或 注销
`<C-l>` 清除屏幕, 同 clear
`<C-z>` 转入后台运行, 当前用户退出后会终止, 如果不想终止 & 或 screen 虚拟终端可供选择

第 58 章

vi/vim

58.1 Introduction

不同的 Linux 发行版都有其不同的附加软件，例如 Red Hat Enterprise Linux 与 Fedora 的 ntsysv 与 setup 等，而 SuSE 则有 YAST 管理工具等。

在所有的 Linux 发行版中都会有的一种文本编辑器就是 vi[1]，而且很多软件默认也是使用 vi 作为编辑器的。vi[2]是一种计算机文本编辑器，由美国计算机科学家比尔·乔伊（Bill Joy）于 1976 年以 BSD 授权发布[2]。

vi 是一种模式编辑器，使用不同的按钮和键击可以更改不同的“模式”，比如说：

- 在“插入模式”下，输入的文本会直接被插入到文档；
- 当按下“退出键”，“插入模式”就会更改为“命令模式”，并且光标的移动和功能的编辑都由字母来响应，例如：
  - “j”用来移动光标到下一行；
  - “k”用来移动光标到上一行；
  - “x”可以删除当前光标处的字符；
  - “i”可以返回到“插入模式”（也可以使用方向键）。

在“命令模式”下，敲入的键（字母）并不会插入到文档，而且多重文本编辑操作是由一组键（字母）来执行，而不是同时按下 <Alt>、<Ctrl> 和其他特殊键来完成。更多更复杂的编辑操作可以使用多重功能基元的组合，比如说：

- “dw”用来删除一个单词；
- “c2fa”可以更改当前的光标处中“a”之前的文本。

因此，对于熟练的 vi 用户可以有更快的操作，因为双手就可以不必离开键盘。

早期的版本中，vi 并没有指示出当前的模式，用户必须按下“退出键”来确认编辑器返回“命令模式”（会有声音提示），后来的 vi 版本可以在“状态条”中（或用图形显示），而在最新的版本中，用户可以在“终端”中设置并使用除主键盘以外的其他键，例如：PgUp，PgDn，Home，End 和 Del 键。另外，图形化界面的 vi 可以很好的支持鼠标和菜单。

直到 Emacs 的出现（1984 年以后），vi 几乎是所有“黑客”所使用的标准 UNIX 编辑器。从 2006 年开始，vi 成为了“单一 UNIX 规范”（Single UNIX Specification）的一部分，因此 vi 或 vi 的一种变形版本（vim）一定会在 UNIX 中找到。当救急软盘作为恢复硬盘崩溃的媒介以来，vi 通常被用户选择，因为一张软盘正好存储下 vi，并且几乎所有人都可以很轻松的使用 vi。

vim（Vi IMproved）[4] 是一种升级版，类似 nvi。作为从 vi 发展出来的一种编辑器[3]，vim 的第一个版本由布莱姆·米勒在 1991 年发布。最初，vim 的简称是 Vi IMitation，随着功能的不断增加，正式名称改成了 Vi IMproved。

vim 具有代码补全、编译和错误调转等功能，还能够进行 shell script 等编程功能，因此可以将 vim 视为一种程序编辑器。另外 vim 还支持正则表达式的搜索模式、多文件编辑、区块复制等。

vim 会依据文件的扩展名或者是文件内的开头信息，判断该文件的内容而自动的调用该程序的语法判断式，再以颜色来显示程序代码与一般信息。

[1] vim 也是 Linux 下第二强大的编辑器，emacs 是公认的世界第一。
[2] vi 是“Visual”的不正规的缩写，来源于另外一个文本编辑器 ex 的命令 visual。

- 1994 年的 vim 3.0 加入了多视窗编辑模式(分区视窗)。
- 1996 年发布的 vim 4.0 是第一个利用 GUI(图形用户界面)的版本。
- 1998 年 5.0 版本的 vim 加入了 highlight(语法高亮)功能。
- 2001 年的 vim 6.0 版本加入了代码折叠、插件、多国语言支持、垂直分区视窗等功能。
- 2006 年 5 月发布的 vim 7.0 版更加入了拼字检查、上下文相关补全,标签页编辑等新功能。
- 2008 年 8 月发布的 vim 7.2,合并了 Vim 7.1 以来的所有修正补丁,并且加入了脚本的浮点数支持。
- 2010 年 8 月发布的 vim 7.3, 加入了“永久撤销”、“Blowfish 算法加密”、“文本隐藏”和“Lua 以及 Python3 的接口”等新功能。

和集成开发环境一样,vim 具有可以配置成在编辑代码源文件之后直接进行编译的功能。编译出错的情况下,可以在另一个窗口中显示出错误。根据错误信息可以直接跳转到正在编辑的源文件出错位置。

在文件比较方面,vim 可以逐行的对文本文件进行比较。例如,vim 可以并排显示两个版本的文件,同时以不同的颜色来表示有差别部分。修改过的、新增的或者是被删除的行会以颜色高亮来强调,没有改变过的部分则会被自动折叠表示。

- 对于已经在 vim 中打开的两个缓冲区,可以使用“:diffthis”对这两个缓冲区的内容进行比较,被比较的缓冲区可以是一个尚未存盘的内存中的缓冲区。
- 在比较两个文件的不同之处时,可以用“:diffget”和“:diffput”命令对每一处不同进行双向的同步,也可以在比较不同时对内容进行其它编辑,然后用“:diffupdate”对最新内容重新进行比较。
- 在浏览两个文件的不同之处时,可以用“[c”和“]c”两个普通(Normal)模式的命令直接跳转到上一个和下一个不同之处。
- 可以通过“:diffopt”等选项更精细地控制哪些区别被认为是真正的不同之处,比如可以设置比较时忽略空白字符数量的不同。

UNIX 下可以用 vimdiff 命令来使用这个功能。

vim 有其脚本语言 vimscript,使用 vimscript 写成的宏可以实现自动执行复杂的操作。使用“-s”选项启动 vim,或者直接切换到宏所在目录使用“:source”命令都可以执行 Vim 脚本。其中,vim 的配置文件就可以作为 vim 脚本的一个范例,在 UNIX 和 Linux 下 vim 的配置文件名是.vimrc,在 Windows 下 vim 的配置文件一般叫做 _vimrc。这个文件在启动 vim 的时候被自动执行。

- vimscript 可以使用 vim 命令行模式的所有命令,使用“:normal”命令则可以使用通常模式中的所有命令。
- vimscript 具有数字和字符串两种数据类型,其中用数字代表布尔类型,0 代表假,之外的数全代表真。vim 7 还提供了列表、关联数组等高级数据结构。
- vimscript 也拥有各种比较运算符和算术运算符。
- vimscript 的控制结构实现了 if 分支和 for/while 循环。
- 用户可以自己定义函数,并且可以使用超过 100 种的预定义函数。
- vimscript 编写成的脚本文件可以在调试模式中进行调试。

不过,vimscript 处理速度并不快,导致 vim 读取大文件的速度很慢(可以通过 LargeFile 脚本优化),而且在处理非常长的行时,vim 速度也会变慢。

58.2 Documents

通常可以在 Unix 系统命令行下输入“vimtutor”进入《VIM 教程》,在 Vim 用户手册中更加详细的描述了 Vim 的基础和进阶功能。另外,可以在 Vim 中输入“:help user-manual”进入用户手册。

vim 提供了文本形式的大量文档,并且提供了各种各样的功能用于快速找到问题的解决方案。根据 vim 自己的帮助文件语法,关键字会被各种各样醒目的颜色表示出来,可以用快捷键像在浏览器中那样浏览帮助文件。

在 GUI 版的 vim(gvim)中还可以使用鼠标在帮助文件中移动。方便用户寻找问题解决方案的功

能还不止这些，其中最主要的是 ":helpgrep" 命令。使用这条命令，用户可以在所有帮助文件中搜索想要察看的内容，用 ":cwindows" 可以在另一个窗口中表示搜索的结果，根据搜索的结果自动在帮助文件内跳转。使用 vim 的帮助功能，还可以在搜索的结果中继续进行搜索。

58.3 Basic Modes

从 vi 演生出来的 vim 具有多种模式，而且 vim 和 vi 都是仅仅通过键盘来在这些模式之中切换。这就使得 vim 可以不用进行菜单或者鼠标操作，并且最小化组合键的操作。具体来说，vim 包括 6 种基本模式和 5 种派生模式。

几乎所有的编辑器都会有插入和执行命令两种模式，并且大多数的编辑器使用了与 Vim 截然不同的方式：命令目录（鼠标或者键盘驱动），组合键（通常通过 control 键和 alt 键组成）或者鼠标输入。vim 和 vi 一样，仅仅通过键盘来在这些模式之中切换，这使得 vim 可以不用进行菜单或者鼠标操作，并且最小化组合键的操作。

58.3.1 Normal

在普通模式中，用户可以执行一般的编辑器命令（比如移动光标，删除文本等），这也是 Vim 启动后的默认模式。这正好和许多新用户期待的操作方式相反（大多数编辑器默认模式为插入模式）。

Vim 强大的编辑能力中很大部分是来自于其普通模式命令。普通模式命令往往需要一个操作符结尾，例如：

- 普通模式命令 "dd" 删除当前行，但是第一个 "d" 的后面可以跟另外的移动命令来代替第二个"d"；
- 用移动到下一行的 "j" 键就可以删除当前行和下一行；
- 可以指定命令重复次数，"2dd"（重复 "dd" 两次）和 "dj" 的效果是一样的。

用户熟悉了文本间移动/跳转的命令和其他的普通模式的编辑命令，并能够灵活组合使用的话，可以比那些没有模式的编辑器更加高效的进行文本编辑。

在普通模式中，有很多方法可以进入插入模式。比较普通的方式是 "a"（append/追加）键或者 "i"（insert/插入）键。

58.3.2 Insert

在 Insert 模式中，大多数按键都会向文本缓冲中插入文本。大多数新用户希望文本编辑器编辑过程中一直保持这个模式。

在插入模式中，可以按 ESC 键回到普通模式。

58.3.3 Visual

可视（Visual）模式与普通模式比较相似，只是移动命令会扩大高亮的文本区域，高亮区域可以是字符、行或者是一块文本。

当执行一个非移动命令时，命令会被执行到这块高亮的区域上，而且 vim 的"文本对象"也能和移动命令一样用在可视模式中。

58.3.4 Select

选择模式和无模式编辑器的行为比较相似（Windows 标准文本控件的方式）。在选择模式中，可以用鼠标或者光标键高亮选择文本，不过输入任何字符的话，vim 会用这个字符替换选择的高亮文本块，并且自动进入插入模式。

58.3.5 Commandline

在命令行模式中可以输入会被解释成并执行的文本，例如执行命令(":" 键)、搜索("/" 和 "?" 键)或者过滤命令("!" 键)。在命令执行之后，vim 返回到命令行模式之前的模式，通常是普通模式。

58.3.6 Ex

Ex 模式和命令行模式比较相似，在使用 ":visual" 命令离开 Ex 模式前，可以一次执行多条命令。

58.4 Derivative Modes

58.4.1 Operator-pending

操作符等待模式是指在普通模式中，执行一个操作命令后 vim 等待一个"动作"来完成这个命令。vim 也支持在操作符等待模式中使用"文本对象"作为动作，包括 "aw" 一个单词(a word)、"as" 一个句子(a sentence)、"ap" 一个段落(a paragraph)等。

比如，在普通模式下 "d2as" 删除当前和下一个句子。在可视模式下 "apU" 把当前段落所有字母大写。

58.4.2 Insert Normal

插入普通模式是在插入模式下按下 ctrl-o 键的时候进入。这个时候暂时进入普通模式，执行完一个命令之后，vim 返回插入模式

58.4.3 Insert Visual

插入可视模式是在插入模式下按下 ctrl-o 键并且开始一个可视选择的时候开始。在可视区域选择取消的时候，vim 返回插入模式。

58.4.4 Insert Select

通常，插入选择模式由插入模式下鼠标拖拽或者 shift 方向键来进入。当选择区域取消的时候，vim 返回插入模式。

58.4.5 Replace

替换模式是一个特殊的插入模式，在这个模式中可以做和插入模式一样的操作，但是每个输入的字符都会覆盖文本缓冲中已经存在的字符。在普通模式下按 "R" 键进入。

另外，Evim(Easy Vim)是一个特殊的 GUI 模式用来尽量表现的和 "无模式" 编辑器一样。编辑器自动进入并且停留在插入模式，用户只能通过菜单、鼠标和键盘控制键来对文本进行操作。可以在命令行下输入 "evim" 或者 "vim -y" 进入。在 Windows 下，通常也可以点击桌面上 Evim(Easy Vim)的图标。

Table 58.1: VIM 使用说明

操作	说明
h 或向左箭头键 (←)	光标向左移动一个字符
j 或向下箭头键 (⊠)	光标向下移动一个字符[3]

[3]键盘上的 hjkl 是排列在一起的，因此可以使用这四个按钮来移动光标。如果想要进行多次移动的话，例如向下移动 30 行，可以使用"30j" 或"30⊠" 的组合按键，亦即加上想要进行的次数 (数字) 后，按下动作即可，其他按键同理。

操作	说明
k 或向上箭头键 (⌧)	光标向上移动一个字符
l 或向右箭头键 (→)	光标向右移动一个字符
[Ctrl] + [f]	屏幕『向下』移动一页，相当于 [Page Down] 按键 (常用)
[Ctrl] + [b]	屏幕『向上』移动一页，相当于 [Page Up] 按键 (常用)
[Ctrl] + [d]	屏幕『向下』移动半页
[Ctrl] + [u]	屏幕『向上』移动半页
+	光标移动到非空格符的下一列
-	光标移动到非空格符的上一列
n<space>	那个 n 表示『数字』，例如 20 。按下数字后再按空格键，光标会向右移动这一行的 n 个字符。例如 20<space> 则光标会向后面移动 20 个字符距离。
0 或功能键 [Home]	这是数字『0 』：移动到这一行的最前面字符处 (常用)
$ 或功能键 [End]	移动到这一行的最后面字符处 (常用)
H	光标移动到这个屏幕的最上方那一行的第一个字符
M	光标移动到这个屏幕的中央那一行的第一个字符
L	光标移动到这个屏幕的最下方那一行的第一个字符
G	移动到这个文件的最后一行 (常用)
nG	n 为数字。移动到这个文件的第 n 行。例如 20G 则会移动到这个文件的第 20 行 (可配合:set nu)
gg	移动到这个文件的第一行，相当于 1G 啊！(常用)
n<Enter>	n 为数字。光标向下移动 n 行 (常用)
/word	向光标之下寻找一个名称为 word 的字符串。例如要在文件内搜寻 vbird 这个字符串，就输入 /vbird 即可！(常用)
?word	向光标之上寻找一个字符串名称为 word 的字符串。
n	这个 n 是英文按键。代表『重复前一个搜寻的动作』。举例来说，如果刚刚我们执行 /vbird 去向下搜寻 vbird 这个字符串，则按下 n 后，会向下继续搜寻下一个名称为 vbird 的字符串。如果是执行?vbird 的话，那么按下 n 则会向上继续搜寻名称为 vbird 的字符串！
N	这个 N 是英文按键。与 n 刚好相反，为『反向』进行前一个搜寻动作。例如 /vbird 后，按下 N 则表示『向上』搜寻 vbird 。使用 / word 配合 n 及 N 是非常有帮助的！可以让你重复的找到一些你搜寻的关键词
:n1,n2s/word1/word2/g	n1 与 n2 为数字。在第 n1 与 n2 行之间寻找 word1 这个字符串，并将该字符串取代为 word2！举例来说，在 100 到 200 行之间搜寻 vbird 并取代为 VBIRD 则：『:100,200s/vbird/VBIRD/g』。(常用)
:1,$s/word1/word2/g	从第一行到最后一行寻找 word1 字符串，并将该字符串取代为 word2！(常用)
:1,$s/word1/word2/gc	从第一行到最后一行寻找 word1 字符串，并将该字符串取代为 word2！且在取代前显示提示字符给用户确认 (confirm) 是否需要取代！(常用)

操作	说明
x, X	在一行字当中，x 为向后删除一个字符 (相当于 [del] 按键)，X 为向前删除一个字符 (相当于 [backspace] 亦即是退格键) (常用)
nx	n 为数字，连续向后删除 n 个字符。举例来说，我要连续删除 10 个字符，『10x』。
dd	删除游标所在的那一整列 (常用)
ndd	n 为数字。删除光标所在的向下 n 列，例如 20dd 则是删除 20 列 (常用)
d1G	删除光标所在到第一行的所有数据
dG	删除光标所在到最后一行的所有数据

58.5 Visual Block

58.6 Visual Windows

58.7 Configuration

58.8 Hotkey

除了上面简易范例的 i, [Esc], :wq 之外，其实 vim 还有非常多的按键可以使用喔！在介绍之前还是要再次强调，vim 的三种模式只有一般模式可以与编辑、指令列模式切换，编辑模式与指令列模式之间并不能切换的！这点在图 2.1 里面有介绍到，注意去看看喔！底下就来谈谈 vim 软件中会用到的按键功能。

移动光标的方法	
h 或向左箭头键 (←)	光标向左移动一个字符
j 或向下箭头键 (↓)	光标向下移动一个字符
k 或向上箭头键 (↑)	光标向上移动一个字符
l 或向右箭头键 (→)	光标向右移动一个字符
如果你将右手放在键盘上的话，你会发现 hjkl 是排列在一起的，因此可以使用这四个按钮来移动光标。如果想要进行多次移动的话，例如向下移动 30 行，可以使用”30j” 或”30↓” 的组合按键，亦即加上想要进行的次数 (数字) 后，按下动作即可	
[Ctrl] + [f]	屏幕『向下』移动一页，相当于 [Page Down] 按键 (常用)
[Ctrl] + [b]	屏幕『向上』移动一页，相当于 [Page Up] 按键 (常用)
[Ctrl] + [d]	屏幕『向下』移动半页
[Ctrl] + [u]	屏幕『向上』移动半页
+	光标移动到非空格符的下一列
-	光标移动到非空格符的上一列
n<space>	那个 n 表示『数字』，例如 20 。按下数字后再按空格键，光标会向右移动这一行的 n 个字符。例如 20<space> 则光标会向后面移动 20 个字符距离。
0 或功能键 [Home]	这是数字『0 』：移动到这一行的最前面字符处 (常用)
$ 或功能键 [End]	移动到这一行的最后面字符处 (常用)

H	光标移动到这个屏幕的最上方那一行的第一个字符
M	光标移动到这个屏幕的中央那一行的第一个字符
L	光标移动到这个屏幕的最下方那一行的第一个字符
G	移动到这个档案的最后一行 (常用)
nG	n 为数字。移动到这个档案的第 n 行。例如 20G 则会移动到这个档案的第 20 行 (可配合:set nu)
gg	移动到这个档案的第一行, 相当于 1G 啊! (常用)
n<Enter>	n 为数字。光标向下移动 n 行 (常用)
搜寻与取代	
/word	向光标之下寻找一个名称为 word 的字符串。例如要在档案内搜寻 vbird 这个字符串, 就输入 /vbird 即可! (常用)
?word	向光标之上寻找一个字符串名称为 word 的字符串。
n	这个 n 是英文按键。代表『重复前一个搜寻的动作』。举例来说, 如果刚刚我们执行 /vbird 去向下搜寻 vbird 这个字符串, 则按下 n 后, 会向下继续搜寻下一个名称为 vbird 的字符串。如果是执行?vbird 的话, 那么按下 n 则会向上继续搜寻名称为 vbird 的字符串!
N	这个 N 是英文按键。与 n 刚好相反, 为『反向』进行前一个搜寻动作。例如 /vbird 后, 按下 N 则表示『向上』搜寻 vbird。
使用 /word 配合 n 及 N 是非常有帮助的! 可以让你重复的找到一些你搜寻的关键词!	
:n1,n2s/word1/word2/g	n1 与 n2 为数字。在第 n1 与 n2 行之间寻找 word1 这个字符串, 并将该字符串取代为 word2 ! 举例来说, 在 100 到 200 行之间搜寻 vbird 并取代为 VBIRD 则:『:100,200s/vbird/VBIRD/g』。(常用)
:1,$s/word1/word2/g	从第一行到最后一行寻找 word1 字符串, 并将该字符串取代为 word2 ! (常用)
:1,$s/word1/word2/gc	从第一行到最后一行寻找 word1 字符串, 并将该字符串取代为 word2 ! 且在取代前显示提示字符给用户确认 (confirm) 是否需要取代! (常用)
x, X	在一行字当中, x 为向后删除一个字符 (相当于 [del] 按键), X 为向前删除一个字符 (相当于 [backspace] 亦即是退格键) (常用)
nx	n 为数字, 连续向后删除 n 个字符。举例来说, 我要连续删除 10 个字符,『10x』。
dd	删除游标所在的那一整列 (常用)
ndd	n 为数字。删除光标所在的向下 n 列, 例如 20dd 则是删除 20 列 (常用)
d1G	删除光标所在到第一行的所有数据
dG	删除光标所在到最后一行的所有数据
d$	删除游标所在处, 到该行的最后一个字符

d0	那个是数字的 0 , 删除游标所在处, 到该行的最前面一个字符
yy	复制游标所在的那一行 (常用)
nyy	n 为数字。复制光标所在的向下 n 列, 例如 20yy 则是复制 20 列 (常用)
y1G	复制游标所在列到第一列的所有数据
yG	复制游标所在列到最后一列的所有数据
y0	复制光标所在的那个字符到该行行首的所有数据
y$	复制光标所在的那个字符到该行行尾的所有数据
p, P	p 为将已复制的数据在光标下一行贴上, P 则为贴在游标上一行! 举例来说, 我目前光标在第 20 行, 且已经复制了 10 行数据。则按下 p 后, 那 10 行数据会贴在原本的 20 行之后, 亦即由 21 行开始贴。但如果是按下 P 呢? 那么原本的第 20 行会被推到变成 30 行。(常用)
J	将光标所在列与下一列的数据结合成同一列
c	重复删除多个数据, 例如向下删除 10 行, [10cj]
u	复原前一个动作。(常用)
[Ctrl]+r	重做上一个动作。(常用)
这个 u 与 [Ctrl]+r 是很常用的指令! 一个是复原, 另一个则是重做一次	
.	不要怀疑! 这就是小数点! 意思是重复前一个动作的意思。如果你想要重复删除、重复贴上等等动作, 按下小数点. 即可

第 59 章

bash

59.1 Type

```
type [-aftpP] name [name ...]
 With no options, indicate how each name would be interpreted if used as a command name.
 If the -t option is used, type prints a string which is one of alias, keyword, function,
     builtin, or file if name is an alias, shell reserved word, function, builtin, or disk
      file, respectively.
 If the name is not found, then nothing is printed, and an exit status of false is
     returned.
 If the -p option is used, type either returns the name of the disk file that would be
     executed if name were specified as a command name, or nothing if ``type -t name''
     would not return file.
 The -P option forces a PATH search for each name, even if ``type -t name'' would not
     return file.
 If a command is hashed, -p and -P print the hashed value, not necessarily the file that
     appears first in PATH.
 If the -a option is used, type prints all of the places that contain an executable named
     name.
 This includes aliases and functions, if and only if the -p option is not also used. The
     table of hashed commands is not consulted when using -a.
 The -f option suppresses shell function lookup, as with the command builtin. type returns
      true if all of the arguments are found, false if any are not found.
```

59.2 Variables

59.3 Redirect

59.4 Pipe

Bibliography

[1] Wikipedia. Shell, . URL http://zh.wikipedia.org/zh-cn/%E6%AE%BC%E5%B1%A4.

[2] Wikipedia. vi, . URL http://zh.wikipedia.org/zh-cn/Vi.

[3] Wikipedia. vim, . URL http://zh.wikipedia.org/zh-cn/Vim.

[4] 吉庆. Vim 使用笔记, 06 2012. URL http://www.cnblogs.com/jiqingwu/archive/2012/06/14/vim_notes.html.

Part VII

Management

第 60 章

log

第 61 章

proc

第 62 章

dmesage

第 63 章

tail

第 64 章

more/less

Part VIII

Service

第 65 章

daemon

第 66 章

process

第 67 章

signal

第 68 章

thread

68.1 Multithread

Part IX

Network

第 69 章

Serial/Parallel

第 70 章

NFS

第 71 章

NIS

第 72 章

DHCP

第 73 章

HTTP

第 74 章

FTP

第 75 章

Samba

第 76 章

NTP

第 77 章

Gateway

第 78 章

Wireless Network

第 79 章

Bluetooth

第 80 章

Bridging

第 81 章

NAS

第 82 章

ATM

Part X

Performance Analysis

Part XI

Security

第 83 章

Firewall

第 84 章

SELinux

第 85 章

MAC

Part XII

Virtualization