

Linux/UNIX 高级编程

中科红旗软件技术有限公司 编著

清华大学出版社

(京)新登字 158 号

内 容 简 介

本书全面系统地介绍了 Linux/UNIX 高级编程的方法。其中第一章介绍了 Linux 的开发工具；第二章至第五章讲述了 Linux 的文件系统，I/O 系统调用、系统数据文件等；第六至第九章讲述了 Linux 进程模型等方面的内容；第十章讲述终端 I/O 编程；第十一章讲述网络套接字编程；第十二章讲述多线程、系统服务等方面的内容。第十三章讲述多线程编程等内容。

本书结合大量实例，概念清晰，特色鲜明，实用性强。

本书适合于 Linux/UNIX 高级编程人员。

版权所有，翻印必究。

本书封面贴有清华大学出版社激光防伪标签，无标签者不得销售。

书 名：Linux/UNIX 高级编程

作 者：中科红旗软件技术有限公司

出 版 者：清华大学出版社（北京清华大学学研大厦，邮编 100084）

<http://www.tup.tsinghua.edu.cn>

责任编辑：柴文强

印 刷 者：清华大学印刷厂

发 行 者：新华书店总店北京发行所

开 本：787×960 1/16 印张：31.5 字数：703 千字

版 次：2001 年 7 月第 1 版 2001 年 7 月第 1 次印刷

书 号：ISBN 7-302-04605-0/TP·2728

印 数：0001~5000

定 价：52.00 元

前 言

Linux 是运行在个人计算机和 workstation 上的 UNIX 操作系统,它继承了 UNIX 的全部优点,是真正的多用户、多任务、多平台的操作系统。在个人计算机和 workstation 上使用 Linux,能充分发挥硬件的功能,使个人计算机能够作为 workstation 和服务器使用,提高 workstation 效率;Linux 符合 POSIX 标准,能同时支持 i386、Alpha、SPARC 和 PowerPC 等处理器;对于在 UNIX 上运行的软件,不用改动或稍加改动就可以运行在 Linux 操作系统上。

同 UNIX 相比,Linux 要灵活得多。Linux 的用户界面非常友好,用户使用很方便。Linux 采用市集开发方式,受自由软件委员会指定的 GNU (General Public License) 公用许可证保护,Linux 系统及其应用软件的源代码都是向用户公开的,允许用户使用 Linux 自带的 C/C++ 语言编译器修改并编译自己喜欢的程序,同时也可以 GNU 的许可下使用这些源程序的部分或全部代码,这种利用 Linux 源程序的可能性大大激发了世界范围内热衷于计算机事业用户的兴趣和创造力,吸引了众多程序员为 Linux 工作;目前,所有 UNIX 的主要功能都已经有了相应的 Linux 工具和程序,大量的自由软件被移植到 Linux 操作系统上,世界各地流行着越来越多的 Linux 发行版本,如 Red Hat Linux, Red Flag Linux 和 Turbo Linux 等;在这些发行版本中,不仅包含 Linux 操作系统的内核,还包含各种编程语言的编译程序、数据库管理系统、图形用户界面、通信联网工具以及大量的其他应用软件。可见,Linux 已经是一个相当优秀的应用软件开发平台。

同 Windows 相比,Linux 也一样具有优势。Linux 强大的网络功能使 Internet 以及许许多多局域网中的服务器都被 Linux 占据,这不仅是由于 Linux 的稳定性相当出色,也是由于 Linux 的系统效率极高;与同类型的 Windows NT 服务器相比,Linux 对硬件的要求可以降低一到两个档次,一台普通的 PC 机就能胜任复杂的网络服务工作;诸如基于文本的 BBS 这样的电子公告牌的讨论性网络服务是 Windows NT 无法实现的,因为迄今为止,只有基于 UNIX 系统的使用文本方式的 BBS 系统;另外,Linux 下的 TCP/IP 网络方面的应用软件更是应有尽有,能实现许多在普通系统中无法实现的网络功能。

不可否认,Linux 具有较 UNIX 和 Windows 更卓越的性能;但是,价格却低得多。因此,如果您想提高自己的计算机使用和编程水平,学习 Linux 确实是一条捷径。

为促进 Linux 在我国的推广应用,使您能更快地步入 Linux 的奇妙世界,少走一些弯路。针对目前市场上缺乏全面、翔实的 Linux / Unix 编程书籍,红旗 Linux 教育培训部拟组织经验丰富的 Linux 编程人员,参考国际经典 Linux、Unix 编程书籍,编写了这本《Linux / Unix 高级编程》。选择在自由软件日趋成熟之际发行,旨在充分发挥我国软件工作人员的积极性和创造力,振兴我国的软件事业。本书内容涵盖了 Linux / Unix 编程的主要方面,具有一定

的深度，具体内容如下：

- 第一章，为您详细介绍 Linux 的软件开发工具，包括编译、调试、维护工具以及广泛应用的集成开发环境。
- 第二、三、四、五章，讲述 Linux 的文件系统、I/O 系统调用、系统数据文件方面的内容。本部分不仅详细讲述了传统 Unix 的文件、I/O 系统调用，还专门介绍了 Linux 文件、I/O 系统的特点。
- 第六、七、八、九章，讲述 Linux 进程模型，包括进程关系、进程控制、进程间通信、信号处理等方面的内容。
- 第十章，讲述终端 I/O 编程，它的用途主要包括：终端、计算机之间的连接、调制解调器、打印机等方面，因此非常复杂。
- 第十一章，讲述网络套接字编程，socket 是最广泛使用和最成熟的网络编程接口规范，用来实现网络的互联。
- 第十二章，守护进程、系统服务等方面的内容，可以编写自己的守护进程来增加、增强系统服务。
- 第十三章，讲述多线程编程，在操作系统层面上讲述了线程的层次和模型，还包括线程同步、互斥等方面的内容。

本书结合大量实例，概念清晰，内容丰富翔实，特色鲜明，实用性强。本书不仅仅是一本手册，很多内容都在操作系统、核心中得到相互印证，真诚期待您的会心一笑。

本书由中科红旗软件技术有限公司编著，耿增强、宋立秋、张荣华、滕靖、朱昊、张焕强、时培植、黄小婉参与了写作工作，在此表示深深的感谢。由于本书涉及的内容丰富，加之篇幅、时间有限，书中难免有纰漏和不足，失误之处请务必谅解和指正。

中科红旗软件技术有限公司

2001 年 4 月

目 录

| | |
|--|-----------|
| 第一章 Linux 软件开发工具 | 错误！未定义书签。 |
| 1.1 gcc 和 g++..... | 错误！未定义书签。 |
| 1.1.1 全局选项..... | 错误！未定义书签。 |
| 1.1.2 语言选项..... | 错误！未定义书签。 |
| 1.1.3 预处理程序选项..... | 错误！未定义书签。 |
| 1.1.4 汇编程序选项..... | 错误！未定义书签。 |
| 1.1.5 连接程序选项..... | 错误！未定义书签。 |
| 1.1.6 目录选项..... | 8 |
| 1.1.7 优化选项..... | 8 |
| 1.1.8 调试选项..... | 错误！未定义书签。 |
| 1.1.9 警告选项..... | 9 |
| 1.2 gnu make | 10 |
| 1.2.1 make 命令的输入文件 | 11 |
| 1.2.2 gnu make | 17 |
| 1.3 autoconf..... | 19 |
| 1.3.1 configure.in 文件的获得 | 19 |
| 1.3.2 aclocal.m4 文件和 acsite.m4 文件的获得 | 25 |
| 1.3.3 configure 脚本的生成 | 25 |
| 1.3.4 configure 脚本的执行 | 25 |
| 1.3.5 软件包的发行和使用..... | 27 |
| 1.3.6 autoconf 的流程框图..... | 28 |
| 1.4 automake | 28 |
| 1.4.1 编写 Makefile.am 文件 | 29 |
| 1.4.2 修改 configure.in 文件 | 34 |
| 1.4.3 执行 automake | 34 |
| 1.4.4 automake 使用实例 | 35 |
| 1.5 gdb..... | 36 |
| 1.5.1 gdb 的命令行选项及参数..... | 36 |
| 1.5.2 gdb 的内部命令..... | 38 |
| 1.5.3 gdb 应用示例..... | 46 |

| | | |
|-------|--------------------------|-----|
| 1.6 | KDevelop | 50 |
| 1.6.1 | KDevelop 特性 | 50 |
| 1.6.2 | KDevelop 对系统的要求 | 50 |
| 1.6.3 | KDevelop 的获取、安装及启动 | 51 |
| 1.6.4 | KDevelop 使用 | 52 |
| 1.7 | wpe 和 xwpe | 65 |
| 第二章 | 文件系统 | 66 |
| 2.1 | 文件系统简介 | 66 |
| 2.1.1 | 概述 | 66 |
| 2.1.2 | 文件模式 | 67 |
| 2.1.3 | 进程的 umask | 70 |
| 2.2 | Linux 文件系统简介 | 71 |
| 2.2.1 | Ext2 文件系统的结构 | 72 |
| 2.2.2 | 超级块 | 73 |
| 2.2.3 | 组描述符 | 73 |
| 2.2.4 | inode | 74 |
| 2.2.5 | 目录结构 | 74 |
| 2.2.6 | Ext2 磁盘空间的分配策略 | 75 |
| 2.3 | 基本文件操作 | 76 |
| 2.3.1 | 文件描述符和流 | 76 |
| 2.3.2 | 打开关闭文件 | 76 |
| 2.3.3 | 文件共享 | 79 |
| 2.3.4 | 顺序文件读写 | 81 |
| 2.3.5 | 随机文件读写 | 84 |
| 2.3.6 | 其他文件操作 | 86 |
| 2.4 | inode 的操作 | 92 |
| 2.4.1 | 查询 inode 信息 | 92 |
| 2.4.2 | 存取权限 | 94 |
| 2.4.3 | 文件所有权 | 95 |
| 2.4.4 | 时间属性 | 96 |
| 2.4.5 | Ext2 扩展属性 | 97 |
| 2.5 | 特殊文件 | 99 |
| 2.5.1 | 硬连接和符号连接 | 99 |
| 2.5.2 | 创建设备文件和管道 | 100 |
| 2.5.3 | /dev/fd | 101 |

| | |
|---------------------------------------|-----|
| 第三章 目录操作 | 102 |
| 3.1 当前目录 | 102 |
| 3.1.1 获得当前目录 | 102 |
| 3.1.2 设置当前目录 | 103 |
| 3.1.3 改变根目录 | 103 |
| 3.2 创建删除目录 | 104 |
| 3.3 浏览目录 | 104 |
| 3.4 名字匹配 | 108 |
| 3.4.1 使用子进程 | 108 |
| 3.4.2 内部匹配 | 109 |
| 第四章 高级 I/O 操作 | 113 |
| 4.1 同时进行多个 I/O 操作 (I/O 复用) | 113 |
| 4.1.1 轮询方式 | 113 |
| 4.1.2 使用非阻塞 I/O | 114 |
| 4.1.3 效率较高的 I/O 复用 | 116 |
| 4.1.4 系统调用 poll | 119 |
| 4.2 内存映像 | 121 |
| 4.2.1 分配页面 | 121 |
| 4.2.2 建立内存映像 | 121 |
| 4.2.3 内存与磁盘的同步 | 125 |
| 4.3 给内存区加锁 | 125 |
| 4.4 文件加锁 | 126 |
| 4.4.1 文件锁 | 126 |
| 4.4.2 记录锁 | 128 |
| 4.4.3 死锁 | 131 |
| 4.4.4 锁的继承与释放 | 132 |
| 4.4.5 建议锁与强制锁 | 132 |
| 4.5 非连续区域读写 | 133 |
| 第五章 系统数据文件和系统信息 | 135 |
| 5.1 简介 | 135 |
| 5.2 密码文件 | 135 |
| 5.2.1 /etc/passwd 文件和 passwd 结构 | 135 |
| 5.2.2 有关的函数接口 | 137 |
| 5.2.3 密码的 shadow 机制 | 141 |

| | |
|---|------------|
| 5.3 组 (groups) | 141 |
| 5.3.1 /etc/group 文件和 group 结构 | 141 |
| 5.3.2 有关的函数接口 | 142 |
| 5.3.3 附加组 | 144 |
| 5.4 其他数据文件 | 147 |
| 5.5 有关用户登录的系统文件 | 148 |
| 5.6 系统标识 | 150 |
| 5.7 系统时钟 | 151 |
| 第六章 进程模型与进程关系 | 155 |
| 6.1 进程 | 155 |
| 6.2 线程 | 155 |
| 6.3 启动例程 | 155 |
| 6.4 终止进程 | 156 |
| 6.5 atexit 函数 | 157 |
| 6.6 命令行参数 错误！未定义书签。 6.7 环境变量列表 | 160 |
| 6.8 C 程序在内存中的分布 | 160 |
| 6.9 共享库 | 161 |
| 6.10 内存分配机制 | 162 |
| 6.10.1 alloca 函数 | 163 |
| 6.11 环境变量的访问与修改 | 163 |
| 6.12 setjmp 和 longjmp 函数 | 165 |
| 6.13 使用局部变量的问题 | 167 |
| 6.14 getrlimit 和 setrlimit 函数 | 169 |
| 6.15 getrusage 函数 | 171 |
| 6.16 终端登录 | 172 |
| 6.16.1 4.3 + BSD 终端登录 | 173 |
| 6.16.2 SVR4 终端登录 | 174 |
| 6.16.3 4.3 + BSD 网络登录 | 174 |
| 6.17 进程组 | 175 |
| 6.18 会话 | 176 |
| 6.19 控制终端 | 177 |
| 6.20 tcgetpgrp 和 tcsetpgrp 函数 | 178 |
| 6.21 作业控制 | 178 |
| 6.22 程序在 shell 下的运行 | 180 |

| | |
|---------------------------------|------------|
| 6.23 孤儿进程组..... | 181 |
| 6.24 4.3 + BSD 对进程关系实现..... | 183 |
| 第七章 进程控制..... | 185 |
| 7.1 进程标识..... | 185 |
| 7.2 fork 函数..... | 185 |
| 7.3 exit 函数..... | 189 |
| 7.4 wait 和 waitpid 函数..... | 191 |
| 7.5 wait3 和 wait4 函数..... | 195 |
| 7.6 竞争条件..... | 195 |
| 7.7 exec 函数..... | 197 |
| 7.8 setuid 和 setgid 函数..... | 200 |
| 7.9 setreuid 和 setregid 函数..... | 202 |
| 7.10 seteuid 和 setegid 函数..... | 202 |
| 7.11 system 函数..... | 203 |
| 7.12 getlogin 函数..... | 205 |
| 7.13 times 函数..... | 205 |
| 7.14 守护进程..... | 207 |
| 7.14.1 守护进程的特点..... | 207 |
| 7.14.2 守护进程的例子..... | 208 |
| 7.14.3 syslog 函数..... | 210 |
| 第八章 进程间通信..... | 212 |
| 8.1 管道和命名管道..... | 212 |
| 8.1.1 管道..... | 212 |
| 8.1.2 流管道..... | 226 |
| 8.1.3 FIFO..... | 227 |
| 8.2 System V IPC..... | 232 |
| 8.2.1 System V IPC 访问方式..... | 232 |
| 8.2.2 消息队列..... | 236 |
| 8.2.3 信号量..... | 242 |
| 8.2.4 共享内存..... | 250 |
| 8.2.5 System V IPC 使用总结..... | 261 |
| 第九章 信号处理..... | 262 |
| 9.1 概述..... | 262 |

| | | |
|--------|--|-----|
| 9.2 | LINUX 系统中的信号 | 263 |
| 9.3 | 对信号的处理 | 265 |
| 9.3.1 | 设置信号处理函数 | 265 |
| 9.3.2 | 系统对信号的处理 | 267 |
| 9.3.3 | 不可靠的信号 | 268 |
| 9.3.4 | 信号的阻塞 | 269 |
| 9.3.5 | 向进程发送信号 | 269 |
| 9.3.6 | 用定时器使进程睡眠 | 270 |
| 9.3.7 | 信号与系统调用 | 273 |
| 9.3.8 | 信号集 | 274 |
| 9.3.9 | 使用信号集屏蔽信号 | 275 |
| 9.3.10 | 设置信号的处理函数 | 276 |
| 9.3.11 | 非局部跳转 | 278 |
| 9.3.12 | 屏蔽信号并使进程等待 | 279 |
| 9.3.13 | 使进程退出 | 280 |
| 9.3.14 | 等待一个进程结束 | 281 |
| 9.3.15 | 实现函数 <code>system</code> 的一种方法 | 282 |
| 9.3.16 | 实现函数 <code>sleep</code> 的一种方法 | 284 |
| 9.3.17 | 作业控制信号 | 285 |
| 第十章 | 终端及伪终端编程 | 287 |
| 10.1 | 引言 | 287 |
| 10.1.1 | 终端 | 287 |
| 10.1.2 | 终端驱动程序 | 287 |
| 10.1.3 | 系统与终端之间的关系 | 288 |
| 10.1.4 | 版本 | 289 |
| 10.2 | UNIX/Linux 中的终端 | 289 |
| 10.2.1 | 概述 | 289 |
| 10.2.2 | 控制终端 | 290 |
| 10.2.3 | 数据传输 | 290 |
| 10.2.4 | 正则模式和非正则模式 | 291 |
| 10.2.5 | 正则模式下的编辑键 | 292 |
| 10.3 | 终端的应用程序设计 | 294 |
| 10.3.1 | 终端的打开与读写 | 294 |
| 10.3.2 | 库函数 <code>ttyname</code> 和 <code>isatty</code> | 299 |
| 10.3.3 | <code>termios</code> 结构 | 301 |

| | | |
|---------|---|-----|
| 10.3.4 | 利用 ioctl 系统调用对终端进行控制..... | 308 |
| 10.3.5 | 另一种对终端进行控制的方法：通过 13 个 termios 系统调用..... | 313 |
| 10.3.6 | 非正则模式..... | 325 |
| 10.3.7 | 终端与 SIGHUP 信号..... | 331 |
| 10.3.8 | 终端窗口大小..... | 331 |
| 10.3.9 | ctermid..... | 333 |
| 10.3.10 | termcap、terminfo 和 curses..... | 334 |
| 10.3.11 | stty 命令..... | 335 |
| 10.4 | 程序 tty_transfer 的设计..... | 336 |
| 10.4.1 | 总体描述..... | 336 |
| 10.4.2 | 头文件、常量定义和 main 函数..... | 337 |
| 10.4.3 | serial_conn 函数..... | 340 |
| 10.4.4 | 文件传输函数..... | 344 |
| 10.4.5 | tty_transfer 的使用..... | 347 |
| 10.5 | 终端管理的发展..... | 348 |
| 10.5.1 | 数据结构的变化..... | 348 |
| 10.5.2 | 流的概念的提出..... | 349 |
| 10.6 | 基于 STREAMS 的终端子系统..... | 350 |
| 10.6.1 | 流的概念..... | 350 |
| 10.6.2 | 基于 STREAMS 终端的优点..... | 351 |
| 10.6.3 | 线路规程模块..... | 352 |
| 10.6.4 | 硬件仿真模块..... | 357 |
| 10.7 | 伪终端程序设计..... | 358 |
| 10.7.1 | 伪终端简介..... | 358 |
| 10.7.2 | SVR4 中的 ptym_open 和 ptys_open..... | 361 |
| 10.7.3 | BSD 中的 ptym_open 和 ptys_open..... | 365 |
| 10.7.4 | pty_fork..... | 367 |
| 10.7.5 | pty 编程举例..... | 372 |
| 10.7.6 | Linux 下的伪终端例程..... | 376 |
| 10.7.7 | 远程方式和分组方式..... | 377 |
| 10.7.8 | 基于 STREAMS 的伪终端子系统..... | 378 |
| 第十一章 | socket 编程..... | 384 |
| 11.1 | 协议支持..... | 384 |
| 11.1.1 | 网络基础知识..... | 384 |
| 11.1.2 | Linux 系统网络模块的结构..... | 386 |

| | | |
|--------|---|-----|
| 11.1.3 | 关于网络地址..... | 387 |
| 11.2 | 几个工具函数..... | 387 |
| 11.3 | socket 编程的基本流程和要用到的函数..... | 388 |
| 11.3.1 | 服务器端程序的基本操作..... | 389 |
| 11.3.2 | 客户端程序的基本操作..... | 391 |
| 11.4 | UNIX 域 socket..... | 391 |
| 11.4.1 | UNIX 域地址..... | 392 |
| 11.4.2 | UNIX 域 socket 服务程序..... | 393 |
| 11.4.3 | 客户程序..... | 395 |
| 11.4.4 | 运行 UNIX 域 socket 示例程序..... | 396 |
| 11.4.5 | 用 socketpair 函数建立未命名 UNIX 域 socket..... | 396 |
| 11.4.6 | 用 UNIX 域 socket 在进程间传递文件描述符..... | 396 |
| 11.5 | TCP/IP 网络编程..... | 401 |
| 11.5.1 | 关于字节序..... | 401 |
| 11.5.2 | IPv4 地址..... | 420 |
| 11.5.3 | Socket 编程中的 IP 地址结构..... | 403 |
| 11.5.4 | 十进制点式 IP 地址与二进制 IP 地址间的转换..... | 403 |
| 11.5.5 | 使用域名..... | 404 |
| 11.5.6 | 域名解析示例..... | 405 |
| 11.5.7 | 查询服务程序的端口号..... | 407 |
| 11.5.8 | TCP 服务程序示例..... | 409 |
| 11.5.9 | TCP client application..... | 411 |
| 11.6 | Socket 出错常量..... | 413 |
| 第十二章 | 守护进程..... | 415 |
| 12.1 | 守护进程简介..... | 415 |
| 12.2 | Syslogd 守护进程..... | 415 |
| 12.3 | Syslog 函数..... | 416 |
| 12.4 | daemon_init 函数..... | 418 |
| 12.4.1 | fork..... | 419 |
| 12.4.2 | setsid..... | 419 |
| 12.4.3 | 忽略 SIGHUP 并再次调用 fork..... | 419 |
| 12.4.4 | 改变工作目录并清除文件创建掩码..... | 420 |
| 12.4.5 | 关闭所有打开的描述符..... | 420 |
| 12.4.6 | 打开 syslog..... | 420 |
| 12.5 | Inetd 守护进程..... | 421 |

| | |
|-----------------------------------|-----|
| 12.6 如何编制一个由 inetd 启动的服务器程序 | 422 |
| 12.6.1 程序 | 422 |
| 12.6.2 配置/etc/services 文件 | 423 |
| 第十三章 多线程编程 | 425 |
| 13.1 概念 | 425 |
| 13.1.1 什么叫线程 | 425 |
| 13.1.2 内核线程与用户层线程 | 426 |
| 13.1.3 进程、LWP 和线程 | 427 |
| 13.1.4 线程的优势 | 436 |
| 13.2 多线程编程基础 | 437 |
| 13.2.1 概念 | 437 |
| 13.2.2 线程的创建和终止 | 438 |
| 13.2.3 线程的合并和分离 | 441 |
| 13.2.4 线程属性 | 443 |
| 13.2.5 其他 | 448 |
| 13.3 线程之间的互斥与同步 | 450 |
| 13.3.1 线程互斥与同步的概念 | 450 |
| 13.3.2 互斥锁 | 450 |
| 13.3.3 利用条件变量实现同步 | 456 |
| 13.3.4 利用互斥锁和条件变量实现同步 | 460 |
| 13.4 线程编程中的其他问题 | 464 |
| 13.4.1 线程的私有变量 | 464 |
| 13.4.2 初始化函数 | 471 |
| 13.4.3 撤消其他线程的执行 | 471 |
| 13.4.4 清场函数 | 474 |
| 13.4.5 线程的信号操作 | 479 |
| 13.4.6 Semaphore.c | 481 |

第一章 Linux 软件开发工具

1.1 gcc 和 g++

gcc 和 g++ 分别是 gnu 的 C 及 C++ 编译器，是 Linux 系统中将 C 及 C++ 语言源文件生成可执行程序的工具。

一个可执行的 C 或 C++ 程序需要经过如下四步生成：

第一步：由预处理程序对 C 或 C++ 语言源文件 (*.c 或 *.cpp、*.C、*.cxx 等) 进行宏扩展和条件处理，并导入前导文件，生成以 .i 为后缀的文件；

第二步：由编译程序将预处理后的文件 (*.i) 中的 C 或 C++ 语言源代码转换成汇编语言代码，生成以 .s 为后缀的汇编文件；

第三步：由汇编程序将汇编文件 (*.s) 中的汇编语言代码转换成目标代码（机器代码，*.o），生成以 .o 为后缀的目标文件；

第四步：由连接程序将目标文件 (*.o) 同指定的库文件进行连接，生成可执行程序。

若非特别指定，gcc 或 g++ 将自动调用上述工具，生成可执行程序。

gcc 和 g++ 命令的调用格式为：

```
gcc [options] filenames
g++ [options] filenames
```

其中：options 为 gcc 和 g++ 命令的选项，可以为空；多个选项必须分开写，如“-dr”不同于“-d-r”。filenames 是 gcc 和 g++ 命令的输入文件列表，多个文件之间以空格分隔。

gcc 和 g++ 命令的选项很多，它们规定着 gcc 和 g++ 命令的行为；掌握 gcc 和 g++ 命令的关键就是正确使用它们的选项。

gcc 和 g++ 命令的选项大致可分为如下几类：

- 全局选项，影响编译的全过程（从预处理到连接）；
- 语言选项；
- 预处理程序选项；
- 汇编程序选项；
- 连接程序选项；
- 目录选项；
- 优化选项；
- 调试选项；

- 警告选项。

下面分别介绍各类的常用选项（除非特殊声明，选项将同时适用于 `gcc` 和 `g++` 命令）。

1.1.1 全局选项

1. `-x language filenames`

此选项指明 `filenames` 文件的编程语言（C 语言、C++ 语言等）及 `gcc/g++` 编译的起始阶段（预处理、编译、汇编、连接）。

`language` 的可选值为 `C`、`C++`、`assembler-with-cpp` 等；其中：`C`、`C++` 指明输入文件为 C 语言、C++ 语言文件，编译过程从预处理阶段开始；`assembler`、`assembler-with-cpp` 指明编译过程从汇编阶段开始。

`-x language` 选项对其后的多个文件都有效，直至遇到下一个 `-x` 为止。

在不使用此选项时，`gcc/g++` 利用文件名后缀来确定语言及编译的起始阶段。如：输入文件名以 `.c` 或 `.cpp` 为后缀时，从预处理阶段开始编译；以 `.i` 为后缀时，从编译阶段开始编译；以 `.s` 为后缀时，从汇编阶段开始编译；以 `.o` 为后缀时，从连接阶段开始编译。使用 `-x` 选项后，文件名后缀将不再对编译的起始阶段起作用。

2. `-x none filenames`

此选项使 `-x language` 选项失去作用，其后文件以名字后缀决定语言及编译的起始阶段。

3. `-c`、`-S`、`-E`

这三个选项决定编译过程到哪一个阶段（预处理、编译、汇编、连接）结束，它们不能同时使用。

`-c` 选项激活预处理、编译及汇编程序，不执行连接程序；编译过程到汇编阶段结束；输出文件为目标文件（`*.o`），可用 `-o` 选项（见下文）另行指定输出文件名。

`-S` 选项激活预处理及编译程序，不执行汇编和连接程序；编译过程到编译阶段结束；输出文件为汇编文件（`*.s`），可用 `-o` 选项（见下文）另行指定输出文件名。

`-E` 选项激活预处理程序，不执行编译、汇编和连接程序；编译过程到预处理阶段结束；输出被送到标准输出，可用 `-o` 选项重新定向到文件输出。

4. `-o filename`

此选项以 `filename` 覆盖 `gcc/g++` 默认的输出文件名；它可以同 `-c`、`-S`、`-E` 中的任何一个选项一同使用，为各阶段的输出文件（可执行文件、目标文件、汇编文件、预处理后的 C/C++ 语言源文件）改名。

因为 `-o` 选项只能指定一个输出文件名，所以当有多个输出文件产生时，`-o` 选项不起作用。

如不使用此选项，默认情况下，可执行程序以 `a.out` 命名，目标文件以 `*.o` 命名，汇编文件以 `*.s` 命名，预处理后的 C/C++ 语言源文件定向到标准输出。

5 . -pipe

此选项用管道代替临时文件来处理不同编译阶段（预处理、编译、汇编、连接）间的通信。这个选项在某些不支持管道操作的系统上不工作，但在 GNU 编译器上不会有问題。

1.1.2 语言选项

1 . -ansi

此选项关闭 GNU C 中与 ANSI C 不兼容的特性，激活 ANSI C 的专有特性。当使用此选项时，预编译器定义了一个宏 `_STRICT_ANSI_`，您可在源程序中加入对此宏的判断来避免使用 ANSI 标准不兼容的特性。

比如：在 ANSI C 中，禁止使用 `asm`、`inline` 和 `typeof` 关键字及 `UNIX`、`vax` 等预定义宏，取而代之的是 `_asm_`、`_inline_`、`_typeof_`、`_UNIX_` 和 `_vax_`；这些被释放出来的关键字和宏可由您赋予新的含义。

注意：-ansi 选项并不拒绝非 ansi 程序。

2 . -fno-asm

此选项实现 -ansi 选项功能的一部分，它禁止将 `asm`、`inline` 和 `typeof` 用作关键字。使用此选项的目的是程序已经为 `asm`、`inline` 和 `typeof` 赋予新的含义，不希望编译器（`gcc`、`g++`）将它们解释为关键字。

3 . -fno-strict-prototype

此选项只对 `g++` 起作用，对 `gcc` 没有影响。

通常情况下，`g++` 编译器将不带参数的函数声明理解为没有参数；使用此选项后，`g++` 认为不带参数的函数声明是没有显式地对参数的个数及类型进行声明，而不是没有参数。

不论是否使用此选项，`gcc` 编译器总是将不带参数的函数声明理解成没有对参数的个数及类型进行声明。

4 . -fthis-is-variable

此选项只对 `g++` 有效。

默认情况下，C++ 语言中的类已经将其对象指定给 `this` 指针，因此不能再将 `this` 用作一般变量；为了同传统 C++ 语言兼容，此选项允许将 `this` 用作一般变量。

5 . -fcond-mismatch

此选项允许条件表达式的第二和第三参数类型不匹配。这样的表达式的值为 `void` 类型。

6 . -funsigned-char

```
-fno-signed-char  
-fsigned-char  
-fno-unsigned-char
```

这四个选项对 char 类型进行设置。char 类型可能为 signed-char，也可能为 unsigned char。每种机器对 char 类型都有默认设置，使用这四个选项可以重新指定 char 类型的设置。其中前两个选项将 char 类型指定为 unsigned char，后两个选项将 char 类型指定为 signed char。

1.1.3 预处理程序选项

此类选项用于设定 gcc/g++ 在预处理阶段的行为。前文曾经讲过，-E 选项使 gcc/g++ 只执行预处理程序；此类中的部分选项必须同-E 选项一起使用，因为它们使预处理程序的输出不适合继续编译。

1 . -include file

```
-imacros file
-Dmacro
-Dmacro=defn
-Umacro
```

上述五个选项的作用相当于 C/C++ 语言源文件中的 #include、#define 和 #undef 语句，但是优先于 gcc/g++ 的输入文件被处理。当 gcc/g++ 的命令行中有上述选项出现时，gcc/g++ 不论 -Dmacro、-Umacro 选项的位置如何，总是最先处理它们；然后再按照命令行中的顺序处理 -include 和 -imacros 选项；只有在命令行中的预处理选项都被处理完之后，gcc/g++ 才去处理输入文件中的预处理语句。

在上述五个选项中：-include file 选项的作用相当于 C/C++ 语言源文件中的“#include”语句，导出 file 文件内容到 gcc/g++ 的输入文件中；-imacros file 选项的作用是将 file 文件中的宏定义扩展到 gcc/g++ 的输入文件中，宏定义本身并不出现在输入文件中；-Dmacro 选项的作用相当于 C/C++ 语言源文件中的“#define macro”语句，定义 macro 宏为串“1”或“TRUE”；-Dmacro=defn 选项的作用相当于 C/C++ 语言源文件中的“#define macro defn”语句，定义 macro 宏为串 defn；-Umacro 选项的作用相当于 C/C++ 语言源文件中的“#undef macro”语句，取消对 macro 宏的预定义。

2 . -undef

此选项取消对任何非标准宏的定义。

```
-Idir
-I-
-idirafter dir
-iprefix prefix
-iwithprefix dir
```

这些选项限定 gcc/g++ 搜索头文件（前导文件）的路径。

C/C++ 语言源文件中约定使用 #include <file.h> 语句来包含任何由 C 编译系统提供的标准前导文件，使用 #incldue "file.h" 语句来包含您自己目录中的前导文件。