Chinese translated version of Documentation/HOWTO

If you have any comment or update to the content, please contact the
original document maintainer directly. However, if you have a problem
communicating in English you can also ask the Chinese maintainer for
help. Contact the Chinese maintainer if this translation is outdated
or if there is a problem with the translation.

Maintainer: Greg Kroah-Hartman <greg@kroah.com>
Chinese maintainer: Li Yang <leoli@freescale.com>

Documentation/HOWTO 的中文翻译

如果想评论或更新本文的内容，请直接联系原文档的维护者。如果你使用英文
交流有困难的话，也可以向中文版维护者求助。如果本翻译更新不及时或者翻
译存在问题，请联系中文版维护者。

英文版维护者： Greg Kroah-Hartman <greg@kroah.com>
中文版维护者： 李阳 Li Yang <leoli@freescale.com>
中文版翻译者： 李阳 Li Yang <leoli@freescale.com>
中文版校译者： 钟宇 TripleX Chung <xxx.phy@gmail.com>
                陈琦 Maggie Chen <chenqi@beyondsoft.com>
                王聪 Wang Cong <xiyou.wangcong@gmail.com>

以下为正文

如何参与Linux内核开发

这是一篇将如何参与Linux内核开发的相关问题一网打尽的终极秘笈。它将指导你
成为一名Linux内核开发者，并且学会如何同Linux内核开发社区合作。它尽可能不
包括任何关于内核编程的技术细节，但会给你指引一条获得这些知识的正确途径。

如果这篇文章中的任何内容不再适用，请给文末列出的文件维护者发送补丁。

入门

你想了解如何成为一名Linux内核开发者？或者老板吩咐你“给这个设备写个Linux
驱动程序”？这篇文章的目的就是教会你达成这些目标的全部诀窍，它将描述你需
要经过的流程以及给出如何同内核社区合作的一些提示。它还将试图解释内核社区
为何这样运作。

Linux内核大部分是由C语言写成的，一些体系结构相关的代码用到了汇编语言。要
参与内核开发，你必须精通C语言。除非你想为某个架构开发底层代码，否则你并
不需要了解（任何体系结构的）汇编语言。下面列举的书籍虽然不能替代扎实的C
语言教育和多年的开发经验，但如果需要的话，做为参考还是不错的：
 - "The C Programming Language" by Kernighan and Ritchie [Prentice Hall]
   《C程序设计语言（第2版·新版）》（徐宝文 李志 译）[机械工业出版社]
 - "Practical C Programming" by Steve Oualline [O'Reilly]
   《实用C语言编程（第三版）》（郭大海 译）[中国电力出版社]
 - "C: A Reference Manual" by Harbison and Steele [Prentice Hall]
   《C语言参考手册（原书第5版）》（邱仲潘 等译）[机械工业出版社]

Linux内核使用GNU C和GNU工具链开发。虽然它遵循ISO C89标准，但也用到了一些标准中没有定义的扩展。内核是自给自足的C环境，不依赖于标准C库的支持，所以并不支持C标准中的部分定义。比如long long类型的大数除法和浮点运算就不允许使用。有时候确实很难弄清楚内核对工具链的要求和它所使用的扩展，不幸的是目前还没有明确的参考资料可以解释它们。请查阅gcc信息页（使用“info gcc”命令显示）获得一些这方面信息。

请记住你是在学习怎么和已经存在的开发社区打交道。它由一群形形色色的人组成，他们对代码、风格和过程有着很高的标准。这些标准是在长期实践中总结出来的，适应于地理上分散的大型开发团队。它们已经被很好得整理成档，建议你在开发之前尽可能多的学习这些标准，而不要期望别人来适应你或者你公司的行为方式。

法律问题

Linux内核源代码都是在GPL（通用公共许可证）的保护下发布的。要了解这种许可的细节请查看源代码主目录下的COPYING文件。如果你对它还有更深入问题请联系律师，而不要在Linux内核邮件组上提问。因为邮件组里的人并不是律师，不要期望他们的话有法律效力。

对于GPL的常见问题和解答，请访问以下链接：
        http://www.gnu.org/licenses/gpl-faq.html

文档

Linux内核代码中包含有大量的文档。这些文档对于学习如何与内核社区互动有着不可估量的价值。当一个新的功能被加入内核，最好把解释如何使用这个功能的文档也放进内核。当内核的改动导致面向用户空间的接口发生变化时，最好将相关信息或手册页(manpages)的补丁发到mtk.manpages@gmail.com，以向手册页(manpages)的维护者解释这些变化。

以下是内核代码中需要阅读的文档：

README
  文件简要介绍了Linux内核的背景，并且描述了如何配置和编译内核。内核的新用户应该从这里开始。

Documentation/Changes
  文件给出了用来编译和使用内核所需要的最小软件包列表。

Documentation/CodingStyle
  描述Linux内核的代码风格和理由。所有新代码需要遵守这篇文档中定义的规范。大多数维护者只会接收符合规定的补丁，很多人也只会帮忙检查符合风格的代码。

Documentation/SubmittingPatches
Documentation/SubmittingDrivers
  这两份文档明确描述如何创建和发送补丁，其中包括（但不仅限于）：
    - 邮件内容
    - 邮件格式
    - 选择收件人
  遵守这些规定并不能保证提交成功（因为所有补丁需要通过严格的内容和风格审查），但是忽视他们几乎就意味着失败。

其他关于如何正确地生成补丁的优秀文档包括：
"The Perfect Patch"
http://www.zip.com.au/~akpm/linux/patches/stuff/tpp.txt
"Linux kernel patch submission format"
http://linux.yyz.us/patch-format.html

Documentation/stable_api_nonsense.txt
论证内核为什么特意不包括稳定的内核内部API，也就是说不包括像这样的特性：
- 子系统中间层（为了兼容性？）
- 在不同操作系统间易于移植的驱动程序
- 减缓（甚至阻止）内核代码的快速变化
这篇文档对于理解Linux的开发哲学至关重要。对于将开发平台从其他操作系统转移到Linux的人来说也很重要。

Documentation/SecurityBugs
如果你认为自己发现了Linux内核的安全性问题，请根据这篇文档中的步骤来提醒其他内核开发者并帮助解决这个问题。

Documentation/ManagementStyle
描述内核维护者的工作方法及其共有特点。这对于刚刚接触内核开发（或者对它感到好奇）的人来说很重要，因为它解释了很多对于内核维护者独特行为的普遍误解与迷惑。

Documentation/stable_kernel_rules.txt
解释了稳定版内核发布的规则，以及如何将改动放入这些版本的步骤。

Documentation/kernel-docs.txt
有助于内核开发的外部文档列表。如果你在内核自带的文档中没有找到你想找的内容，可以查看这些文档。

Documentation/applying-patches.txt
关于补丁是什么以及如何将它打在不同内核开发分支上的好介绍

内核还拥有大量从代码自动生成的文档。它包含内核内部API的全面介绍以及如何妥善处理加锁的规则。生成的文档会放在 Documentation/DocBook/目录下。在内核源码的主目录中使用以下不同命令将会分别生成PDF、Postscript、HTML和手册页等不同格式的文档：

```
make pdfdocs
make psdocs
make htmldocs
make mandocs
```

如何成为内核开发者

如果你对Linux内核开发一无所知，你应该访问“Linux内核新手”计划：
http://kernelnewbies.org
它拥有一个可以问各种最基本的内核开发问题的邮件列表（在提问之前一定要记得查找已往的邮件，确认是否有人已经回答过相同的问题）。它还拥有一个可以获得实时反馈的IRC聊天频道，以及大量对于学习Linux内核开发相当有帮助的文档。

网站简要介绍了源代码组织结构、子系统划分以及目前正在进行的项目（包括内核中的和单独维护的）。它还提供了一些基本的帮助信息，比如如何编译内核和打补丁。

如果你想加入内核开发社区并协助完成一些任务，却找不到从哪里开始，可以访问“Linux内核房管员”计划：
	http://janitor.kernelnewbies.org/
这是极佳的起点。它提供一个相对简单的任务列表，列出内核代码中需要被重新整理或者改正的地方。通过和负责这个计划的开发者们一同工作，你会学到将补丁集成进内核的基本原理。如果还没有决定下一步要做什么的话，你还可能会得到方向性的指点。

如果你已经有一些现成的代码想要放到内核中，但是需要一些帮助来使它们拥有正确的格式。请访问“内核导师”计划。这个计划就是用来帮助你完成这个目标的。它是一个邮件列表，地址如下：
	http://selenic.com/mailman/listinfo/kernel-mentors

在真正动手修改内核代码之前，理解要修改的代码如何运作是必需的。要达到这个目的，没什么办法比直接读代码更有效了（大多数花招都会有相应的注释），而且一些特制的工具还可以提供帮助。例如，“Linux代码交叉引用”项目就是一个值得特别推荐的帮助工具，它将源代码显示在有编目和索引的网页上。其中一个更新及时的内核源码库，可以通过以下地址访问：
	http://sosdg.org/~coywolf/lxr/

开发流程

目前Linux内核开发流程包括几个“主内核分支”和很多子系统相关的内核分支。这些分支包括：
- 2.6.x主内核源码树
- 2.6.x.y -stable内核源码树
- 2.6.x -git内核补丁集
- 2.6.x -mm内核补丁集
- 子系统相关的内核源码树和补丁集

2.6.x内核主源码树

2.6.x内核是由Linus Torvalds（Linux的创造者）亲自维护的。你可以在kernel.org网站的pub/linux/kernel/v2.6/目录下找到它。它的开发遵循以下步骤：
- 每当一个新版本的内核被发布，为期两周的集成窗口将被打开。在这段时间里维护者可以向Linus提交大段的修改，通常这些修改已经被放到-mm内核中几个星期了。提交大量修改的首选方式是使用git工具（内核的代码版本管理工具，更多的信息可以在http://git.or.cz/获取），不过使用普通补丁也是可以的。
- 两个星期以后-rc1版本内核发布。之后只有不包含可能影响整个内核稳定性的新功能的补丁才可能被接受。请注意一个全新的驱动程序（或者文件系统）有可能在-rc1后被接受是因为这样的修改完全独立，不会影响其他的代码，所以没有造成内核退步的风险。在-rc1以后也可以用git向Linus提交补丁，不过所有的补丁需要同时被发送到相应的公众邮件列表以征询意见。
- 当Linus认为当前的git源码树已经达到一个合理健全的状态足以发布供人测试时，一个新的-rc版本就会被发布。计划是每周都发布新的-rc版本。
- 这个过程一直持续下去直到内核被认为达到足够稳定的状态，持续时间大概是6个星期。
- 以下地址跟踪了在每个-rc发布中发现的退步列表：
  http://kernelnewbies.org/known_regressions

关于内核发布，值得一提的是Andrew Morton在linux-kernel邮件列表中如是说：

“没有人知道新内核何时会被发布，因为发布是根据已知bug的情况来决定的，而不是根据一个事先制定好的时间表。”

2.6.x.y -stable（稳定版）内核源码树

由4个数字组成的内核版本号说明此内核是-stable版本。它们包含基于2.6.x版本内核的相对较小且至关重要的修补，这些修补针对安全性问题或者严重的内核退步。

这种版本的内核适用于那些期望获得最新的稳定版内核并且不想参与测试开发版或者实验版的用户。

如果没有2.6.x.y版本内核存在，那么最新的2.6.x版本内核就相当于是当前的稳定版内核。

2.6.x.y版本由“稳定版”小组（邮件地址<stable@kernel.org>）维护，一般隔周发布新版本。

内核源码中的Documentation/stable_kernel_rules.txt文件具体描述了可被稳定版内核接受的修改类型以及发布的流程。

2.6.x -git补丁集

Linus的内核源码树的每日快照，这个源码树是由git工具管理的（由此得名）。这些补丁通常每天更新以反映Linus的源码树的最新状态。它们比-rc版本的内核源码树更具试验性质，因为这个补丁集是全自动生成的，没有任何人来确认其是否真正健全。

2.6.x -mm补丁集

这是由Andrew Morton维护的试验性内核补丁集。Andrew将所有子系统的内核源码和补丁拼凑到一起，并且加入了大量从linux-kernel邮件列表中采集的补丁。这个源码树是新功能和补丁的试炼场。当补丁在-mm补丁集里证明了其价值以后Andrew或者相应子系统的维护者会将补丁发给Linus以便集成进主内核源码树。

在将所有新补丁发给Linus以集成到主内核源码树之前，我们非常鼓励先把这些补丁放在-mm版内核源码树中进行测试。

这些内核版本不适合在需要稳定运行的系统上运行，因为运行它们比运行任何其他内核分支都更具有风险。

如果你想为内核开发进程提供帮助，请尝试并使用这些内核版本，并在linux-kernel邮件列表中提供反馈，告诉大家你遇到了问题还是一切正常。

通常-mm版补丁集不光包括这些额外的试验性补丁，还包括发布时-git版主源码树中的改动。

-mm版内核没有固定的发布周期，但是通常在每两个-rc版内核发布之间都会有若干个-mm版内核发布（一般是1至3个）。

子系统相关内核源码树和补丁集

相当一部分内核子系统开发者会公开他们自己的开发源码树，以便其他人能了解内

核的不同领域正在发生的事情。如上所述，这些源码树会被集成到-mm版本内核中。

下面是目前可用的一些内核源码树的列表：
  通过git管理的源码树：
    - Kbuild开发源码树， Sam Ravnborg <sam@ravnborg.org>
      git.kernel.org:/pub/scm/linux/kernel/git/sam/kbuild.git

    - ACPI开发源码树, Len Brown <len.brown@intel.com>
      git.kernel.org:/pub/scm/linux/kernel/git/lenb/linux-acpi-2.6.git

    - 块设备开发源码树, Jens Axboe <axboe@suse.de>
      git.kernel.org:/pub/scm/linux/kernel/git/axboe/linux-2.6-block.git

    - DRM开发源码树, Dave Airlie <airlied@linux.ie>
      git.kernel.org:/pub/scm/linux/kernel/git/airlied/drm-2.6.git

    - ia64开发源码树, Tony Luck <tony.luck@intel.com>
      git.kernel.org:/pub/scm/linux/kernel/git/aegl/linux-2.6.git

    - ieee1394开发源码树, Jody McIntyre <scjody@modernduck.com>
      git.kernel.org:/pub/scm/linux/kernel/git/scjody/ieee1394.git

    - infiniband开发源码树, Roland Dreier <rolandd@cisco.com>
      git.kernel.org:/pub/scm/linux/kernel/git/roland/infiniband.git

    - libata开发源码树, Jeff Garzik <jgarzik@pobox.com>
      git.kernel.org:/pub/scm/linux/kernel/git/jgarzik/libata-dev.git

    - 网络驱动程序开发源码树, Jeff Garzik <jgarzik@pobox.com>
      git.kernel.org:/pub/scm/linux/kernel/git/jgarzik/netdev-2.6.git

    - pcmcia开发源码树, Dominik Brodowski <linux@dominikbrodowski.net>
      git.kernel.org:/pub/scm/linux/kernel/git/brodo/pcmcia-2.6.git

    - SCSI开发源码树, James Bottomley <James.Bottomley@SteelEye.com>
      git.kernel.org:/pub/scm/linux/kernel/git/jejb/scsi-misc-2.6.git

  使用quilt管理的补丁集：
    - USB, PCI, 驱动程序核心和I2C, Greg Kroah-Hartman <gregkh@suse.de>
      kernel.org/pub/linux/kernel/people/gregkh/gregkh-2.6/
    - x86-64, 部分i386, Andi Kleen <ak@suse.de>
      ftp.firstfloor.org:/pub/ak/x86_64/quilt/

  其他内核源码树可以在http://git.kernel.org的列表中和MAINTAINERS文件里
  找到。

报告bug
----

bugzilla.kernel.org是Linux内核开发者们用来跟踪内核Bug的网站。我们鼓励用
户在这个工具中报告找到的所有bug。如何使用内核bugzilla的细节请访问：
        http://test.kernel.org/bugzilla/faq.html

内核源码主目录中的REPORTING-BUGS文件里有一个很好的模板。它指导用户如何报
告可能的内核bug以及需要提供哪些信息来帮助内核开发者们找到问题的根源。

利用bug报告

练习内核开发技能的最好办法就是修改其他人报告的bug。你不光可以帮助内核变得更加稳定，还可以学会如何解决实际问题从而提高自己的技能，并且让其他开发者感受到你的存在。修改bug是赢得其他开发者赞誉的最好办法，因为并不是很多人都喜欢浪费时间去修改别人报告的bug。

要尝试修改已知的bug，请访问http://bugzilla.kernel.org网址。如果你想获得最新bug的通知，可以订阅bugme-new邮件列表（只有新的bug报告会被寄到这里）或者订阅bugme-janitor邮件列表（所有bugzilla的变动都会被寄到这里）。

        http://lists.osdl.org/mailman/listinfo/bugme-new
        http://lists.osdl.org/mailman/listinfo/bugme-janitors

邮件列表

正如上面的文档所描述，大多数的骨干内核开发者都加入了Linux Kernel邮件列表。如何订阅和退订列表的细节可以在这里找到：
        http://vger.kernel.org/vger-lists.html#linux-kernel
网上很多地方都有这个邮件列表的存档(archive)。可以使用搜索引擎来找到这些存档。比如：
        http://dir.gmane.org/gmane.linux.kernel
在发信之前，我们强烈建议你先在存档中搜索你想要讨论的问题。很多已经被详细讨论过的问题只在邮件列表的存档中可以找到。

大多数内核子系统也有自己独立的邮件列表来协调各自的开发工作。从MAINTAINERS文件中可以找到不同话题对应的邮件列表。

很多邮件列表架设在kernel.org服务器上。这些列表的信息可以在这里找到：
        http://vger.kernel.org/vger-lists.html

在使用这些邮件列表时，请记住保持良好的行为习惯。下面的链接提供了与这些列表（或任何其它邮件列表）交流的一些简单规则，虽然内容有点滥竽充数。
        http://www.albion.com/netiquette/

当有很多人回复你的邮件时，邮件的抄送列表会变得很长。请不要将任何人从抄送列表中删除，除非你有足够的理由这么做。也不要只回复到邮件列表。请习惯于同一封邮件接收两次（一封来自发送者一封来自邮件列表），而不要试图通过添加一些奇特的邮件头来解决这个问题，人们不会喜欢的。

记住保留你所回复内容的上下文和源头。在你回复邮件的顶部保留“某某某说到……”这几行。将你的评论加在被引用的段落之间而不要放在邮件的顶部。

如果你在邮件中附带补丁，请确认它们是可以直接阅读的纯文本（如Documentation/SubmittingPatches文档中所述）。内核开发者们不希望遇到附件或者被压缩了的补丁。只有这样才能保证他们可以直接评论你的每行代码。请确保你使用的邮件发送程序不会修改空格和制表符。一个防范性的测试方法是先将邮件发送给自己，然后自己尝试是否可以顺利地打上收到的补丁。如果测试不成功，请调整或者更换你的邮件发送程序直到它正确工作为止。

总而言之，请尊重其他的邮件列表订阅者。

同内核社区合作

内核社区的目标就是提供尽善尽美的内核。所以当你提交补丁期望被接受进内核的时候，它的技术价值以及其他方面都将被评审。那么你可能会得到什么呢？
- 批评
- 评论
- 要求修改
- 要求证明修改的必要性
- 沉默

要记住，这些是把补丁放进内核的正常情况。你必须学会听取对补丁的批评和评论，从技术层面评估它们，然后要么重写你的补丁要么简明扼要地论证修改是不必要的。如果你发的邮件没有得到任何回应，请过几天后再试一次，因为有时信件会湮没在茫茫信海中。

你不应该做的事情：
- 期望自己的补丁不受任何质疑就直接被接受
- 翻脸
- 忽略别人的评论
- 没有按照别人的要求做任何修改就重新提交

在一个努力追寻最好技术方案的社区里，对于一个补丁有多少好处总会有不同的见解。你必须要抱着合作的态度，愿意改变自己的观点来适应内核的风格。或者至少愿意去证明你的想法是有价值的。记住，犯错误是允许的，只要你愿意朝着正确的方案去努力。

如果你的第一个补丁换来的是一堆修改建议，这是很正常的。这并不代表你的补丁不会被接受，也不意味着有人和你作对。你只需要改正所有提出的问题然后重新发送你的补丁。

内核社区和公司文化的差异

内核社区的工作模式同大多数传统公司开发队伍的工作模式并不相同。下面这些例子，可以帮助你避免某些可能发生问题：

用这些话介绍你的修改提案会有好处：
- 它同时解决了多个问题
- 它删除了2000行代码
- 这是补丁，它已经解释了我想要说明的
- 我在5种不同的体系结构上测试过它……
- 这是一系列小补丁用来……
- 这个修改提高了普通机器的性能……

应该避免如下的说法：
- 我们在AIX/ptx/Solaris就是这么做的，所以这么做肯定是好的……
- 我做这行已经20年了，所以……
- 为了我们公司赚钱考虑必须这么做
- 这是我们的企业产品线所需要的
- 这里是描述我观点的1000页设计文档
- 这是一个5000行的补丁用来……
- 我重写了现在乱七八糟的代码，这就是……
- 我被规定了最后期限，所以这个补丁需要立刻被接受

另外一个内核社区与大部分传统公司的软件开发队伍不同的地方是无法面对面地交

流。使用电子邮件和IRC聊天工具做为主要沟通工具的一个好处是性别和种族歧视将会更少。Linux内核的工作环境更能接受妇女和少数族群，因为每个人在别人眼里只是一个邮件地址。国际化也帮助了公平的实现，因为你无法通过姓名来判断人的性别。男人有可能叫李丽，女人也有可能叫王刚。大多数在Linux内核上工作过并表达过看法的女性对在linux上工作的经历都给出了正面的评价。

对于一些不习惯使用英语的人来说，语言可能是一个引起问题的障碍。在邮件列表中要正确地表达想法必需良好地掌握语言，所以建议你在发送邮件之前最好检查一下英文写得是否正确。

拆分修改

Linux内核社区并不喜欢一下接收大段的代码。修改需要被恰当地介绍、讨论并且拆分成独立的小段。这几乎完全和公司中的习惯背道而驰。你的想法应该在开发最开始的阶段就让大家知道，这样你就可以及时获得对你正在进行的开发的反馈。这样也会让社区觉得你是在和他们协作，而不是仅仅把他们当作倾销新功能的对象。无论如何，你不要一次性地向邮件列表发送50封信，你的补丁序列应该永远用不到这么多。

将补丁拆开的原因如下：

1）小的补丁更有可能被接受，因为它们不需要太多的时间和精力去验证其正确性。一个5行的补丁，可能在维护者看了一眼以后就会被接受。而500行的补丁则需要数个小时来审查其正确性（所需时间随补丁大小增加大约呈指数级增长）。

   当出了问题的时候，小的补丁也会让调试变得非常容易。一个一个补丁地回溯将会比仔细剖析一个被打上的大补丁（这个补丁破坏了其他东西）容易得多。

2）不光发送小的补丁很重要，在提交之前重新编排、化简（或者仅仅重新排列）补丁也是很重要的。

这里有内核开发者Al Viro打的一个比方：

> “想象一个老师正在给学生批改数学作业。老师并不希望看到学生为了得到正确解法所进行的尝试和产生的错误。他希望看到的是最干净最优雅的解答。好学生了解这点，绝不会把最终解决之前的中间方案提交上去。”
>
> 内核开发也是这样。维护者和评审者不希望看到一个人在解决问题时的思考过程。他们只希望看到简单和优雅的解决方案。

直接给出一流的解决方案，和社区一起协作讨论尚未完成的工作，这两者之间似乎很难找到一个平衡点。所以最好尽早开始收集有利于你进行改进的反馈；同时也要保证修改分成很多小块，这样在整个项目都准备好被包含进内核之前，其中的一部分可能会先被接收。

必须了解这样做是不可接受的：试图将未完成的工作提交进内核，然后再找时间修复。

证明修改的必要性

除了将补丁拆成小块，很重要的一点是让Linux社区了解他们为什么需要这样修改。你必须证明新功能是有人需要的并且是有用的。

记录修改

当你发送补丁的时候，需要特别留意邮件正文的内容。因为这里的信息将会做为补丁的修改记录(ChangeLog)，会被一直保留以备大家查阅。它需要完全地描述补丁，包括：

- 为什么需要这个修改
- 补丁的总体设计
- 实现细节
- 测试结果

想了解它具体应该看起来像什么，请查阅以下文档中的“ChangeLog”章节：

“The Perfect Patch”
http://www.zip.com.au/~akpm/linux/patches/stuff/tpp.txt

这些事情有时候做起来很难。要在任何方面都做到完美可能需要好几年时间。这是一个持续提高的过程，它需要大量的耐心和决心。只要不放弃，你一定可以做到。很多人已经做到了，而他们都曾经和现在的你站在同样的起点上。

感谢Paolo Ciarrocchi允许“开发流程”部分基于他所写的文章(http://linux.tar.bz/articles/2.6-development_process)，感谢Randy Dunlap和Gerrit Huizenga完善了应该说和不该说的列表。感谢Pat Mochel, Hanna Linder, Randy Dunlap, Kay Sievers, Vojtech Pavlik, Jan Kara, Josh Boyer, Kees Cook, Andrew Morton, Andi Kleen, Vadim Lobanov, Jesper Juhl, Adrian Bunk, Keri Harris, Frans Pop, David A. Wheeler, Junio Hamano, Michael Kerrisk和Alex Shepard的评审、建议和贡献。没有他们的帮助，这篇文档是不可能完成的。

英文版维护者： Greg Kroah-Hartman <greg@kroah.com>