Chinese translated version of Documentation/SubmittingPatches

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 Contact the Chinese maintainer if this translation is outdated or if there is a problem with the translation.

Chinese maintainer: TripleX Chung <triplex@zh-kernel.org>

Documentation/SubmittingPatches 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。

中文版维护者： 钟宇 TripleX Chung <triplex@zh-kernel.org>
中文版翻译者： 钟宇 TripleX Chung <triplex@zh-kernel.org>
中文版校译者： 李阳 Li Yang <leo@zh-kernel.org>
王聪 Wang Cong <xiyou.wangcong@gmail.com>

以下为正文

如何让你的改动进入内核
或者
获得亲爱的 Linus Torvalds 的关注和处理

对于想要将改动提交到 Linux 内核的个人或者公司来说，如果不熟悉“规矩”，提交的流程会让人畏惧。本文档收集了一系列建议，这些建议可以大大的提高你的改动被接受的机会。
阅读 Documentation/SubmitChecklist 来获得在提交代码前需要检查的项目的列表。如果你在提交一个驱动程序，那么同时阅读一下 Documentation/SubmittingDrivers 。

第一节 - 创建并发送你的改动
---

1) "diff -up"

使用 "diff -up" 或者 "diff -uprN" 来创建补丁。

所有内核的改动，都是以补丁的形式呈现的，补丁由 diff(1) 生成。创建补丁的时候，要确认它是以 "unified diff" 格式创建的，这种格式由 diff(1) 的 '-u' 参数生成。而且，请使用 '-p' 参数，那样会显示每个改动所在的C函数，使得产生的补丁容易读得多。补丁应该基于内核源代码树的根目录，而不是里边的任何子目录。
为一个单独的文件创建补丁，一般来说这样做就够了：

```
SRCTREE= linux-2.6
MYFILE=  drivers/net/mydriver.c
```

```
cd $SRCTREE
cp $MYFILE $MYFILE.orig
vi $MYFILE      # make your change
cd ..
diff -up $SRCTREE/$MYFILE{.orig,} > /tmp/patch
```

为多个文件创建补丁，你可以解开一个没有修改过的内核源代码树，然后和你自己的代码树之间做 diff 。例如：

```
MYSRC= /devel/linux-2.6

tar xvfz linux-2.6.12.tar.gz
mv linux-2.6.12 linux-2.6.12-vanilla
diff -uprN -X linux-2.6.12-vanilla/Documentation/dontdiff \
        linux-2.6.12-vanilla $MYSRC > /tmp/patch
```

"dontdiff" 是内核在编译的时候产生的文件的列表，列表中的文件在 diff(1) 产生的补丁里会被跳过。"dontdiff" 文件被包含在2.6.12和之后版本的内核源代码树中。对于更早的内核版本，你可以从
<http://www.xenotime.net/linux/doc/dontdiff> 获取它。
确定你的补丁里没有包含任何不属于这次补丁提交的额外文件。记得在用diff(1)生成补丁之后，审阅一次补丁，以确保准确。
如果你的改动很散乱，你应该研究一下如何将补丁分割成独立的部分，将改动分割成一系列合乎逻辑的步骤。这样更容易让其他内核开发者审核，如果你想你的补丁被接受，这是很重要的。下面这些脚本能够帮助你做这件事情：
Quilt:
http://savannah.nongnu.org/projects/quilt

Andrew Morton 的补丁脚本:
http://www.zip.com.au/~akpm/linux/patches/
作为这些脚本的替代，quilt 是值得推荐的补丁管理工具(看上面的链接)。

2)描述你的改动。
描述你的改动包含的技术细节。

要多具体就写多具体。最糟糕的描述可能是像下面这些语句：“更新了某驱动程序”，“修正了某驱动程序的bug”，或者“这个补丁包含了某子系统的修改，请使用。”

如果你的描述开始变长，这表示你也许需要拆分你的补丁了，请看第3小节，继续。

3)拆分你的改动

将改动拆分，逻辑类似的放到同一个补丁文件里。

例如，如果你的改动里同时有bug修正和性能优化，那么把这些改动才分到两个或者更多的补丁文件中。如果你的改动包含对API的修改，并且修改了驱动程序来适应这些新的API，那么把这些修改分成两个补丁。

另一方面，如果你将一个单独的改动做成多个补丁文件，那么将它们合并成一个单独的补丁文件。这样一个逻辑上单独的改动只被包含在一个补丁文件里。

如果有一个补丁依赖另外一个补丁来完成它的改动，那没问题。简单的在你的补丁描述里指出“这个补丁依赖某补丁”就好了。

如果你不能将补丁浓缩成更少的文件，那么每次大约发送出15个，然后等待审查和整合。

4)选择 e-mail 的收件人

看一遍 MAINTAINERS 文件和源代码，看看你所的改动所在的内核子系统有没有指定的维护者。如果有，给他们发e-mail。

如果没有找到维护者，或者维护者没有反馈，将你的补丁发送到内核开发者主邮件列表 linux-kernel@vger.kernel.org。大部分的内核开发者都跟踪这个邮件列表，可以评价你的改动。

每次不要发送超过15个补丁到 vger 邮件列表！！！

Linus Torvalds 是决定改动能否进入 Linux 内核的最终裁决者。他的 e-mail 地址是 <torvalds@linux-foundation.org> 。他收到的 e-mail 很多，所以一般的说，最好别给他发 e-mail。

那些修正bug，“显而易见”的修改或者是类似的只需要很少讨论的补丁可以直接发送或者CC给Linus。那些需要讨论或者没有很清楚的好处的补丁，一般先发送到linux-kernel邮件列表。只有当补丁被讨论得差不多了，才提交给Linus。

5)选择CC(e-mail 抄送)列表

除非你有理由不这样做，否则CC linux-kernel@vger.kernel.org。

除了 Linus 之外，其他内核开发者也需要注意到你的改动，这样他们才能评论你的改动并提供代码审查和建议。linux-kernel 是 Linux 内核开发者主邮件列表。其它的邮件列表为特定的子系统提供服务，比如 USB，framebuffer 设备，虚拟文件系统，SCSI 子系统，等等。查看 MAINTAINERS 文件来获得和你的改动有关的邮件列表。

Majordomo lists of VGER.KERNEL.ORG at:
        <http://vger.kernel.org/vger-lists.html>

如果改动影响了用户空间和内核之间的接口，请给 MAN-PAGES 的维护者（列在 MAITAINERS 文件里的）发送一个手册页（man-pages）补丁，或者至少通知一下改变，让一些信息有途径进入手册页。

即使在第四步的时候，维护者没有作出回应，也要确认在修改他们的代码的时候，一直将维护者拷贝到CC列表中。

对于小的补丁，你也许会CC到 Adrian Bunk 管理的搜集琐碎补丁的邮件列表 (Trivial Patch Monkey)trivial@kernel.org，那里专门收集琐碎的补丁。下面这样的补丁会被看作“琐碎的”补丁：
  文档的拼写修正。
  修正会影响到 grep(1) 的拼写。
  警告信息修正(频繁的打印无用的警告是不好的。)
  编译错误修正（代码逻辑的确是对的，只是编译有问题。）
  运行时修正（只要真的修正了错误。）
  移除使用了被废弃的函数/宏的代码(例如 check_region。)
  联系方式和文档修正。
  用可移植的代码替换不可移植的代码（即使在体系结构相关的代码中，既然有人拷贝，只要它是琐碎的）
  任何文件的作者/维护者对该文件的改动（例如 patch monkey 在重传模式下）

URL: <http://www.kernel.org/pub/linux/kernel/people/bunk/trivial/>

(译注，关于“琐碎补丁”的一些说明：因为原文的这一部分写得比较简单，所以不得不违例写一下译注。"trivial"这个英文单词的本意是“琐碎的，不重要的。”但是在这里有稍微有一些变化，例如对一些明显的NULL指针的修正，属于运行时修正，会被归类到琐碎补丁里。虽然NULL指针的修正很重要，但是这样的修正往往很小而且很容易得到检验，所以也被归入琐碎补丁。琐碎补丁更精确的归类应该是“simple, localized & easy to verify”，也就是说简单的，局部的和易于检验的。trivial@kernel.org邮件列表的目的是针对这样的补丁，为提交者提供一个中心，来降低提交的门槛。)

6)没有 MIME 编码，没有链接，没有压缩，没有附件，只有纯文本。

Linus 和其他的内核开发者需要阅读和评论你提交的改动。对于内核开发者来说，可以“引用”你的改动很重要，使用一般的 e-mail 工具，他们就可以在你的代码的任何位置添加评论。

因为这个原因，所有的提交的补丁都是 e-mail 中“内嵌”的。
警告：如果你使用剪切-粘贴你的补丁，小心你的编辑器的自动换行功能破坏你的补丁。

不要将补丁作为 MIME 编码的附件，不管是否压缩。很多流行的 e-mail 软件不是任何时候都将 MIME 编码的附件当作纯文本发送的，这会使得别人无法在你的代码中加评论。另外，MIME 编码的附件会让 Linus 多花一点时间来处理，这就降低了你的改动被接受的可能性。

警告：一些邮件软件，比如 Mozilla 会将你的信息以如下格式发送：
---- 邮件头 ----
Content-Type: text/plain; charset=us-ascii; format=flowed
---- 邮件头 ----
问题在于 “format=flowed” 会让接收端的某些邮件软件将邮件中的制表符替换成空格以及做一些类似的替换。这样，你发送的时候看起来没问题的补丁就被破坏了。

要修正这个问题，只需要将你的 mozilla 的 defaults/pref/mailnews.js 文件里的
pref("mailnews.send_plaintext_flowed", false); // RFC 2646=======
修改成
pref("mailnews.display.disable_format_flowed_support", true);
就可以了。

7) e-mail 的大小

给 Linus 发送补丁的时候，永远按照第6小节说的做。

大的改动对邮件列表不合适，对某些维护者也不合适。如果你的补丁，在不压缩的情况下，超过了40kB，那么你最好将补丁放在一个能通过 internet 访问的服务器上，然后用指向你的补丁的 URL 替代。

8) 指出你的内核版本

在标题和在补丁的描述中，指出补丁对应的内核的版本，是很重要的。

如果补丁不能干净的在最新版本的内核上打上，Linus 是不会接受它的。

9) 不要气馁，继续提交。

当你提交了改动以后，耐心地等待。如果 Linus 喜欢你的改动并且同意它，那么它将在下一个内核发布版本中出现。

然而，如果你的改动没有出现在下一个版本的内核中，可能有若干原因。减少那些原因，修正错误，重新提交更新后的改动，是你自己的工作。

Linus不给出任何评论就“丢弃”你的补丁是常见的事情。在系统中这样的事情很平常。如果他没有接受你的补丁，也许是由于以下原本：
* 你的补丁不能在最新版本的内核上干净的打上。
* 你的补丁在 linux-kernel 邮件列表中没有得到充分的讨论。
* 风格问题（参照第2小节）
* 邮件格式问题（重读本节）
* 你的改动有技术问题。
* 他收到了成吨的 e-mail，而你的在混乱中丢失了。
* 你让人为难。

有疑问的时候，在 linux-kernel 邮件列表上请求评论。

10）在标题上加上 PATCH 的字样

Linus 和 linux-kernel 邮件列表的 e-mail 流量都很高，一个通常的约定是标题行以 [PATCH] 开头。这样可以让 Linus 和其他内核开发人员可以从 e-mail 的讨论中很轻易的将补丁分辨出来。

11）为你的工作签名

为了加强对谁做了何事的追踪，尤其是对那些透过好几层的维护者的补丁，我们建议在发送出去的补丁上加一个 “sign-off” 的过程。

"sign-off" 是在补丁的注释的最后的简单的一行文字，认证你编写了它或者其他人有权力将它作为开放源代码的补丁传递。规则很简单：如果你能认证如下信息：

```
开发者来源证书 1.1
对于本项目的贡献，我认证如下信息：
（a）这些贡献是完全或者部分的由我创建，我有权利以文件中指出
的开放源代码许可证提交它；或者
（b）这些贡献基于以前的工作，据我所知，这些以前的工作受恰当的开放
源代码许可证保护，而且，根据许可证，我有权提交修改后的贡献，
无论是完全还是部分由我创造，这些贡献都使用同一个开放源代码许可证
（除非我被允许用其它的许可证），正如文件中指出的；或者
（c）这些贡献由认证（a），（b）或者（c）的人直接提供给我，而
且我没有修改它。
（d）我理解并同意这个项目和贡献是公开的，贡献的记录（包括我
一起提交的个人记录，包括 sign-off ）被永久维护并且可以和这个项目
或者开放源代码的许可证同步地再发行。
那么加入这样一行：
Signed-off-by: Random J Developer <random@developer.example.org>
```

使用你的真名（抱歉，不能使用假名或者匿名。）

有人在最后加上标签。现在这些东西会被忽略，但是你可以这样做，来标记公司内部的过程，或者只是指出关于 sign-off 的一些特殊细节。

12）标准补丁格式

标准的补丁，标题行是：
    Subject: [PATCH 001/123] 子系统:一句话概述

标准补丁的信体存在如下部分：

- 一个 "from" 行指出补丁作者。

- 一个空行

- 说明的主体，这些说明文字会被拷贝到描述该补丁的永久改动记录里。

- 一个由"---"构成的标记行

- 不合适放到改动记录里的额外的注解。

- 补丁本身（diff 输出）

标题行的格式，使得对标题行按字母序排序非常的容易 - 很多 e-mail 客户端都可以支持 - 因为序列号是用零填充的，所以按数字排序和按字母排序是一样的。

e-mail 标题中的“子系统”标识哪个内核子系统将被打补丁。

e-mail 标题中的“一句话概述”扼要的描述 e-mail 中的补丁。“一句话概述”不应该是一个文件名。对于一个补丁系列（“补丁系列”指一系列的多个相关补丁），不要对每个补丁都使用同样的“一句话概述”。

记住 e-mail 的“一句话概述”会成为该补丁的全局唯一标识。它会蔓延到 git 的改动记录里。然后“一句话概述”会被用在开发者的讨论里，用来指代这个补丁。用户将希望通过 google 来搜索"一句话概述"来找到那些讨论这个补丁的文章。

一些标题的例子：

    Subject: [patch 2/5] ext2: improve scalability of bitmap searching
    Subject: [PATCHv2 001/207] x86: fix eflags tracking

"from" 行是信体里的最上面一行，具有如下格式：
        From: Original Author <author@example.com>

"from" 行指明在永久改动日志里，谁会被确认为作者。如果没有 "from" 行，那么邮件头里的 "From: " 行会被用来决定改动日志中的作者。

说明的主题将会被提交到永久的源代码改动日志里，因此对那些早已经不记得和这个补丁相关的讨论细节的有能力的读者来说，是有意义的。

"---" 标记行对于补丁处理工具要找到哪里是改动日志信息的结束，是不可缺少的。

对于 "---" 标记之后的额外注解，一个好的用途就是用来写 diffstat，用来显示修改了什么文件和每个文件都增加和删除了多少行。diffstat 对于比较大的补丁特别有用。其余那些只是和时刻或者开发者相关的注解，不合适放到永久的改动日志里的，也应该放这里。
使用 diffstat的选项 "-p 1 -w 70" 这样文件名就会从内核源代码树的目录开始，不会占用太宽的空间（很容易适合80列的宽度，也许会有一些缩进。）

在后面的参考资料中能看到适当的补丁格式的更多细节。

第二节 提示，建议和诀窍

本节包含很多和提交到内核的代码有关的通常的"规则"。事情永远有例外...但是你必须真的有好的理由这样做。你可以把本节叫做Linus的计算机科学入门课。

1) 读 Document/CodingStyle

Nuff 说过，如果你的代码和这个偏离太多，那么它有可能会被拒绝，没有更多的审查，没有更多的评价。

2) #ifdef 是丑陋的
混杂了 ifdef 的代码难以阅读和维护。别这样做。作为替代，将你的 ifdef 放在头文件里，有条件地定义 "static inline" 函数，或者宏，在代码里用这些东西。让编译器把那些"空操作"优化掉。

一个简单的例子，不好的代码：

```
    dev = alloc_etherdev (sizeof(struct funky_private));
    if (!dev)
        return -ENODEV;
    #ifdef CONFIG_NET_FUNKINESS
    init_funky_net(dev);
    #endif
```

清理后的例子:

```
(头文件里)
    #ifndef CONFIG_NET_FUNKINESS
    static inline void init_funky_net (struct net_device *d) {}
    #endif

(代码文件里)
    dev = alloc_etherdev (sizeof(struct funky_private));
    if (!dev)
        return -ENODEV;
    init_funky_net(dev);
```

3) 'static inline' 比宏好

Static inline 函数相比宏来说，是好得多的选择。Static inline 函数提供了类型安全，没有长度限制，没有格式限制，在 gcc 下开销和宏一样小。

宏只在 static inline 函数不是最优的时候[在 fast paths 里有很少的独立的案例]，或者不可能用 static inline 函数的时候[例如字符串分配]。
应该用 'static inline' 而不是 'static __inline__', 'extern inline' 和 'extern __inline__' 。

4) 不要过度设计

不要试图预计模糊的未来事情，这些事情也许有用也许没有用："让事情尽可能的简单，而不是更简单"。

第三节 参考文献

Andrew Morton, "The perfect patch" (tpp).
  <http://www.zip.com.au/~akpm/linux/patches/stuff/tpp.txt>

Jeff Garzik, "Linux kernel patch submission format".
  <http://linux.yyz.us/patch-format.html>

Greg Kroah-Hartman, "How to piss off a kernel subsystem maintainer".
  <http://www.kroah.com/log/2005/03/31/>
  <http://www.kroah.com/log/2005/07/08/>
  <http://www.kroah.com/log/2005/10/19/>
  <http://www.kroah.com/log/2006/01/11/>

NO!!!! No more huge patch bombs to linux-kernel@vger.kernel.org people!
  <http://marc.theaimsgroup.com/?l=linux-kernel&m=112112749912944&w=2>

Kernel Documentation/CodingStyle:
  <http://sosdg.org/~coywolf/lxr/source/Documentation/CodingStyle>

Linus Torvalds's mail on the canonical patch format:
  <http://lkml.org/lkml/2005/4/7/183>

--