怎样为Linux内核打补丁

作者　　：Jesper Juhl，2005年8月
最后更新日期 ：2006-01-05

译者　　：Jeffshia, 2006年8月
email　：tshxiayu@126.com

在Linux内核邮件列表中一个经常被问到的问题就是怎样为Linux内核打一个补丁，或者更具体一点说，存在这么多的主干/分支，一个补丁到底要打在哪个版本的基础内核上。希望这篇文档能够为你解释明白这一点。

除了解释怎样应用以及卸载补丁以外，在这里还提供了一个不同内核树(以及如何为它们打上特定补丁)的简要介绍。

什么是补丁？

一个补丁就是一个文本文档，这个文档包含了在两个不同版本的源代码树之间的变化。
补丁是通过diff应用程序来创建的。

为了正确地打上一个补丁，你需要知道这个补丁是从哪个基础版本产生出来的以及这个补丁将要把目前的源代码树变化到什么新的版本。这些信息或者会出现在补丁文件的原数据中，或者可能从文件名中推断出来。

怎样打补丁和卸载补丁

可以使用patch程序来打一个补丁。patch程序读取一个diff(或者patch)文件，然后把文件中描述的变化内容应用到代码树上。

Linux内核中的补丁是相对于保存内核源代码目录的父目录而生成的。

这就意味着：patch文件中的文件路径包含了它所基于的内核源文件目录的名字(或者像是"a/"和"b/"之类的其它名字)。

由于这很可能和你本地机器上的内核源代码目录的名字不匹配(但是对于查看一个没有标签的补丁所基于的内核版本是非常有用的)。你应该切换到你的内核源代码目录，并且在打补丁的时候去掉patch中文件名字路径的第一个分量(patch命令的-p1参数可以完成这个任务)。

为了卸载掉一个以前已经打上的补丁，使用-R参数来打补丁。

于是，如果你使用如下的命令来打补丁:

```
patch -p1 < ../patch-x.y.z
```

那么你可以像下面这样来卸载掉这个补丁：

```
patch -R -p1 < ../patch-x.y.z
```

我怎样把一个patch/diff文件加到一个patch中去？

这个可以通过很多种不同的方法来办到(像是Linux以及其它的类Unix操作系统一样)。

在下面所有的例子中，我把这个文件(未压缩)以下面的语法通过标准输入加入到patch中:

```
patch -p1 < path/to/patch-x.y.z
```

如果你只是想能够照着下面的例子而并不想知道其它的打补丁的方法的话，那么你就可以不阅读这一节余下的内容了。

也可以通过patch的-i参数来获得文件的名字，就像下面这样：

```
patch -p1 -i path/to/patch-x.y.z
```

如果你的patch文件是使用gzip或者bzip2来压缩的，那么你在使用之前就不必解压缩了，你可以像下面这样把它加入到patch文件中：

```
zcat path/to/patch-x.y.z.gz | patch -p1
bzcat path/to/patch-x.y.z.bz2 | patch -p1
```

如果你想在打补丁之前，手动解压缩这个patch文件，那么你就可以对这个文件进行gunzip或者bunzip2操作---就像下面这样：

```
gunzip patch-x.y.z.gz
bunzip2 patch-x.y.z.bz2
```

这个将会给你一个无格式的文本patch-x.y.z文件，这时候你就可以把这个东西以你喜欢的方式通过标准输入或者是-i参数来加入到patch中。

一些其它的比较好的参数是-s，这个参数使patch保持无输出而不是输出错误，这将阻止错误在屏幕上滚动的速度太快。--dry-run参数使得patch命令仅仅打印出来一个将要打的patch的列表，没有真正的做任何变动。最后，-verbose告诉patch命令打印出正在进行的工作的更多信息。

打补丁时候的常见错误

当用patch命令来打一个补丁的时候，它试图以不同的方法来验证这个文件的完整性。

检查这个文件是一个有效的patch文件并且检查这些被改变周围的代码是不是和提供的

上下文相匹配。这些仅仅是patch所作的两个最基本的完整性检查。

如果patch遇到了一些看起来不正确的事情，那么它有两种选择。它可以拒绝应用这些改变并且异常中断或者它试图找到一个方法来使patch命令仅仅做一些比较小的改变。

一个patch试图修正错误的例子就是：如果所有的上下文都匹配，被改变的行匹配，但是这些行的行号不匹配。这是可能发生的，例如，如果patch在一个文件的中间做了一些改变，但是出于一些原因在文件的开头处一些行被添加了进来或者被删除了。在这种情况下，一切看起来都很好，它只是简单地上下移动一点，这时候patch通常会修正这些行号并且打上这个补丁。

任何时候，只要patch在打补丁的时候需要改动文件的一些内容，它就会告诉你说：这个补丁打得有点儿混乱。你应该对这些改变保持一些警惕，因为即使补丁很可能被正确地打上了，但是情况并不总是这样，有些情况下结果会是错误的。

当patch命令遇到一个变化而不能进行使用一种模糊的方法进行弥补的时候，它就会彻底地放弃这个动作，并且留下来一个以.rej为扩展名的文件。你可以阅读这个文件来查看到底是什么改变不能进行下去，从而在你愿意的情况下来手动修补它。

如果你的内核源代码上没有应用任何第三方的补丁，只是一些来自kernel.org的补丁，并且你打这些补丁的顺序是正确的，而且你自己没有对这些源文件进行改动过，那么你应该就不会看到一个补丁对这个文件的模糊的改变或者是一些拒绝消息。如果你确实看到了这些消息的话，那么将有非常高的危险性，这说明或者是你的本地的源代码书或者是补丁文件在某些方面被玷污了。在这种情况下，你很可能是应该重新下载这个补丁文件，如果事情仍然还是保持原样的话，那么我建议你去尝试从kernel.org上下载一个完整的新的源代码树。

让我们来看一下补丁可能产生的更多信息。

如果patch命令停下来并且显示一个“ File to patch ”的提示符，那么这个时候patch命令找不到要打补丁的文件。很可能的情况是你忘记指定-p1参数或者你处于一个错误的目录中了。更加不常见的一种情况是，你会发现一些补丁需要使用-p0参数而不是-p1参数来打补丁(阅读这个补丁文件应该能揭示出这些信息--如果是这样的话，这是一种创建补丁文件的人所犯的错误，但是不致命)

如果你得到信息“ Hunk #2 succeeded at 1887 with fuzz 2 (offset 7 lines) ”，或者是一个类似的消息，那就意味着patch命令必须调整改变的位置（在这个例子中，它需要在它想打补丁的地方移动7行来适应这个补丁）。结果得到的文件可能正确也可能不正确，这决定于这个文件与所期望的文件不相同的原因。这常常发生于你所要打的patch产生于一个另外一个内核版本，这个内核版本和你要打的patch所基于的内核版本不同。

如果你得到一个类似于“ Hunk #3 FAILED at 2387 ”之类的消息，那么这就意味着不能正确地打上这个补丁，并且patch程序也不能模糊地通过。这将产生一个导致patch失败的.rej文件并且产生一个.orig文件把一些不能改变的原始内容显示给你。

如果你得到的信息是：“ Reversed (or previously applied) patch detected! Assume -R? [n] ”，那么patch检测到了这个补丁文件中包含的改变已经应用在了目标文件上。如果你确实已经在此之前打了这个补丁并且重新打补丁遇到了错误，那么你可以简单地选择[n]o并且终止这次补丁动作。如果你之前打了补丁并且想得到打补丁以前的版本，但是忘记了指定-R参数，那么你在这里可以回答[y]es来使用patch

为你恢复它。这也可能发生在补丁文件的创建者在创建补丁文件的时候倒置了源文件和目标目录的位置，在这种情况下从patch中revert实际上是打上了这个补丁。

一个类似于“ patch: **** unexpected end of file in patch ”或者“ patch unexpectedly ends in middle of line ”的消息意味着patch命令对你加入到它之中的文件觉得没有意义。或者是你的下载被打断了，你试图打上一个没有压缩的patch文件，而事前并没有解压缩它，或者是你使用的补丁文件在传输的某个地方被一个邮件客户端或者一个邮件传输代理给损坏了。例如，通过把一个长行分成两行。通常情况下这些警告可以通过把这两个被分开的行合并起来来解决。

就像我上面提到的那样，如果你打的是从kernel.org得来的补丁到一个正确的版本，并且你没有修改过源代码树，这些错误将从来不会发生。因此如果你从kernel.org来的补丁上得来了这些错误，那么你应该很可能认为或者是你的补丁文件或者是源代码树损坏掉了。我建议你重新开始下载一个完整的内核树以及你想要打的补丁。

有patch的替代品么？

是的，有一些替代品。

你可以使用“ interdiff ”程序(http://cyberelk.net/tim/patchutils/)来产生一个文件来表示两个补丁文件之间之间的不同，然后打上这个文件。这可以使你从一个像2.6.12.2的版本一步就可以到达版本2.6.12.3。-z标志甚至可以使你送给interdiff一些使用gzip或者bzip2压缩的格式文件而不用使用zcat或者bzcat或者手动解压缩。

下面展示了你可以怎样一步就可以从2.6.12.2到达2.6.12.3

```
interdiff -z ../patch-2.6.12.2.bz2 ../patch-2.6.12.3.gz | patch -p1
```

尽管interdiff可以节省一到两步，但是我还是建议你通常情况下应该做这些附加的步骤，这是由于在某些情况下interdiff会把事情弄糟。

另外的一个替代品是“ ketchup ”，是一个使用python写的脚本，这个脚本可以自动下载和打上一些补丁(http://www.selenic.com/ketchup/).

另外一些比较好的工具是diffstat,可以显示patch所作的所有改变的总结；lsdiff，可以显示在一个patch文件中受影响的文件的简短列表，以及(可选)每一个补丁的行号码；grepdiff，可以显示在补丁包含给定的正则表达式的时候显示一个被补丁文件修改的文件的列表。

我可以从哪儿下载这些补丁？

补丁文件可以从http://kernel.org/来获得
最近的补丁文件可以从首页的链接中得到，但是他们也有自己的特定的主页。

2.6.x.y(-稳定)以及2.6.x补丁位于：
ftp://ftp.kernel.org/pub/linux/kernel/v2.6/

-rc补丁位于：
ftp://ftp.kernel.org/pub/linux/kernel/v2.6/testing/

-git补丁位于：
ftp://ftp.kernel.org/pub/linux/kernel/v2.6/snapshots/

-mm内核补丁位于：
ftp://ftp.kernel.org/pub/linux/kernel/people/akpm/patches/2.6/

在ftp.kernel.org中，你可以使用ftp.cc.kernel.org,这里cc是一个国家的代码。这样你就可以从一个
在地理上离你比较近的位置的镜像站点下载，从而使你获得较快的下载速度，全局上更少的带宽以及
对于kernel.org主服务器的更小的压力---这是比较好的事情，所以你还是在可能的时候使用这些镜像
站点。

2.6.x内核

这是Linus发布的基础稳定版本.发布的最高版本是最新的。

如果发现了冲突或者严重的瑕疵，那么在这个基础上，一个-stable的修正补丁就会被发布
出来(参见下面)。一旦一个新的2.6.x的基础内核发布出来，就可以得到一个测试版本的补丁
，这个补丁基于先前的2.6.x版本内核和这个新的内核。

为了应用一个从2.6.11到2.6.12的补丁，你最好按照下面来做(注意这些补丁不能应用于2.6.x.y的内核,
而是应用在2.6.x的基础内核---如果你需要从2.6.x.y到2.6.x+1,那么你首先需要卸载掉2.6.x.y的补丁)

下面是一些例子：

```
#从2.6.11到2.6.12
$ cd ~/linux-2.6.11                 # 切换到内核源代码目录
$ patch -p1 < ../patch-2.6.12       # 应用2.6.12补丁
$ cd ..
$ mv linux-2.6.11 linux-2.6.12      # 重命名源代码目录

# moving from 2.6.11.1 to 2.6.12
$ cd ~/linux-2.6.11.1               # 切换到内核源代码目录
$ patch -p1 -R < ../patch-2.6.11.1  # 恢复出来2.6.11.1
                                    # 源代码目录现在是2.6.11
$ patch -p1 < ../patch-2.6.12       # 应用新的2.6.12补丁
```

```
$ cd ..
$ mv linux-2.6.11.1 linux-2.6.12      # 重命名源代码目录
```

2.6.x.y内核

带有四位数字版本号的内核是-stable的内核。他们包含了对一个给定的2.6.x内核的一些安全问题以及发现的重要的退化的修复。

对于那些想要最近的稳定内核并且对于测试开发中的试验性的版本没有兴趣的用户来说，我们推荐这个分支。

如果没有可用的2.6.x.y内核，那么最高数字的2.6.x内核是目前的稳定内核。

注意：维护稳定内核的团队通常会做一些增量的补丁，就像是基于最近的主流版本发布的补丁一样。但是在下面我仅仅说明了非增量的情况。那些增量式的版本可以在下面的ftp处找到: ftp://ftp.kernel.org/pub/linux/kernel/v2.6/incr/.

这些补丁不是增量式的，意味着例如对于2.6.12.3补丁不能应用于2.6.12.2的内核源代码上去，但是可以应用在2.6.12内核代码上。
因此，为了为了把2.6.12.3的补丁应用到你使用的2.6.12.2的内核源代码上，你不得不卸载掉2.5.12.2补丁(因此你可以得到一个基础的2.6.12的内核源代码)，并且应用新的2.6.12.3补丁。

下面是一个小例子：

```
$ cd ~/linux-2.6.12.2               # 切换到内核源代码目录
$ patch -p1 -R < ../patch-2.6.12.2    # 回归2.6.12.2补丁
$ patch -p1 < ../patch-2.6.12.3       # 应用新的2.6.12.3补丁
$ cd ..
$ mv linux-2.6.12.2 linux-2.6.12.3    # 重新命名内核源代码目录
```

-rc内核

这些是候选的发布内核。当Linus认为目前的git(内核的源代码管理工具)内核树处于一个健全的稳定状态足以用来测试的时候，而发布的开发内核。

这些内核是不稳定的，如果你试着运行他们应该会想到可能会不时地有问题出现。
但是这是主开发分支上的最稳定的内核，并且最终会变成下一个稳定的内核。因此让尽可能多的人来测试它就显得格外重要。

对于那些想帮忙测试开发中的内核但是又不想跑那些试验性的东西的人来说，这将是
一个非常好的分支。(这样的人应该参照下面的关于-git和-mm内核的部分)

-rc补丁是非增量式的，他们应用于2.6.x内核上，就像上面描述的2.6.x.y内核一样。在-rcN
后缀之前的内核版本号代表了这个-rc的内核最终会变成的内核版本。

因此，2.6.13-rc5意思是这是2.6.13内核的第五个候选的发布版本，并且这个补丁应该打在
2.6.12的内核源代码上。

下面是3个关于怎样打这些补丁的例子：

```
# 首先是一个从2.6.12到2.6.13-rc3的例子

$ cd ~/linux-2.6.12               # 切换到2.6.12的源代码目录
$ patch -p1 < ../patch-2.6.13-rc3     # 打上2.6.13-rc3的补丁
$ cd ..
$ mv linux-2.6.12 linux-2.6.13-rc3    # 重新命名源代码目录

# 现在从2.6.13-rc3迁移到2.6.13-rc5

$ cd ~/linux-2.6.13-rc3           # 切换到2.6.12的源代码目录
$ patch -p1 -R < ../patch-2.6.13-rc3  # 卸载掉2.6.13-rc3补丁
$ patch -p1 < ../patch-2.6.13-rc5     # 应用新的2.6.13-rc5补丁
$ cd ..
$ mv linux-2.6.13-rc3 linux-2.6.13-rc5 # 重新命名源代码目录

# 最后让我们试着从2.6.12.3到2.6.13-rc5

$ cd ~/linux-2.6.12.3             # 切换到内核源代码目录
$ patch -p1 -R < ../patch-2.6.12.3    # 回返2.6.12.3补丁
$ patch -p1 < ../patch-2.6.13-rc5     # 应用新的2.6.13-rc5补丁
$ cd ..
$ mv linux-2.6.12.3 linux-2.6.13-rc5  # 重新命名源代码目录
```

-git内核

这些是每天Linus的内核树的快照(在一个git仓库中管理着，因此得名)。

这些补丁通常每天都发布而且代表了的Linus的内核树的当前状态，由于它们是自动产生的
甚至没有任何一个光标的骚动来看它们是不是健全的，所以它们比-rc内核更具有试验性。

-git补丁不是增量的，它们或者是应用在2.6.x内核上或者是应用在一个基础的
2.6.x-rc内核上---这一点你可以从他们的名字上看出来。一个名字是2.6.12-git1的
补丁应用在2.6.12内核源代码上，一个名字为2.6.13-rc3-git2的补丁应用在2.6.13-rc3
的内核源代码上。

这里是一些怎样打这些补丁的例子：

```
# 从2.6.12迁移到2.6.12-git1

$ cd ~/linux-2.6.12                 # 切换到内核源代码目录
$ patch -p1 < ../patch-2.6.12-git1     # 应用2.6.12-git1补丁
$ cd ..
$ mv linux-2.6.12 linux-2.6.12-git1    # 重新命名内核源代码目录

# 从2.6.12-git1迁移到2.6.13-rc2-git3

$ cd ~/linux-2.6.12-git1            # 切换到内核源代码目录
$ patch -p1 -R < ../patch-2.6.12-git1  # 回返2.6.12-git1补丁
                            # 我们现在有了一个2.6.12内核
$ patch -p1 < ../patch-2.6.13-rc2      # 打上2.6.13-rc2补丁
                            # 内核现在是2.6.13-rc2
$ patch -p1 < ../patch-2.6.13-rc2-git3  # 打上2.6.13-rc2-git3补丁
                            # 内核现在是2.6.13-rc2-git3
$ cd ..
$ mv linux-2.6.12-git1 linux-2.6.13-rc2-git3   # 重新命名内核源代码目录
```

-mm内核

这是Andrew Morton发布的实验性的内核

-mm树作为一个新特性和实验性的补丁的实验场。一旦一个补丁在-mm中经过一段时间被证明有价值，为了使它能包含在主流内核中，Andrew就会把它推给Linus。

尽管鼓励的方法是通过-mm树把补丁推给Linus，这个步骤并不是总被实行。子系统的维护者(或者个人)有些时候直接把补丁推给Linus，尽管(或者之后)它们已经被它并到了-mm中并得到了测试(或者有些时候并没有事前在-mm中得到测试)。

通常情况下你应该尽力使你的补丁通过-mm中最大程度测试后再到达主流内核中。

这个分支是一个持续的变化并且包含了一些实验性的特征，很多正在debug的补丁并不适合于主流的内核等等。这个分支是这个文档中描述的最具有试验性的分支。

这些内核内核不适合于应用在要求稳定的系统上面，并且在运行中比其他任何的分支都可能承担更大的风险(确信你有最新的备份---这对于任何实验性的内核都适用，但是对于-mm更是这样)。

这些内核除了包含所有的试验性的补丁以外，它们还包含了在主流-git内核发布的时候任何可用的改变。

对-mm内核测试会得到极大的赏识，因为这个分支的总的目的就是为了在改变被加到更加稳定的主流的Linus内核树之前，消除退化、死机、数据失败bug、build失败(以及任何通常意义上的bug)。

但是-mm的测试者应该清醒地认识到这个源代码树中的失败会比其他任何树中的都要普遍。

-mm内核并不会以一个固定的时间发布，但是通常一些-mm内核会在每一个-rc内核(通常1到3个)发布的中间。-mm内核或者是应用于一个基础的2.6.x内核(当还没有-rc内核发布的时候)或者应用于一个Linus -rc的内核。

这里有一个打-mm补丁的例子

```
# 从2.6.12到2.6.12-mm1

$ cd ~/linux-2.6.12                # 切换到2.6.12的源文件目录
$ patch -p1 < ../2.6.12-mm1          # 打一个2.6.12-mm1的补丁
$ cd ..
$ mv linux-2.6.12 linux-2.6.12-mm1     # 重新正确命名这个源文件


# 从2.6.12-mm1到2.6.13-rc3-mm3

$ cd ~/linux-2.6.12-mm1
$ patch -p1 -R < ../2.6.12-mm1        # 卸载掉2.6.12-mm1补丁
                          # 现在我们得到了一个2.6.12的源文件
$ patch -p1 < ../patch-2.6.13-rc3      # 打一个2.6.13-rc3的补丁
                          # 我们现在得到一个2.6.13-rc3的源文件
$ patch -p1 < ../2.6.13-rc3-mm3       # 打一个2.6.13-rc3-mm3的补丁
$ cd ..
$ mv linux-2.6.12-mm1 linux-2.6.13-rc3-mm3     # 重新命名源文件目录
```

上面总结了不同内核树的一些解释。我希望你已经明白了怎样打不同的补丁并且对你测试内核有所帮助。

致谢列表：
Randy Dunlap, Rolf Eike Beer, Linus Torvalds, Bodo Eggert,
Johannes Stezenbach, Grant Coady, Pavel Machek，还有一些人我可能忘记了他们的名字，
但是他们对这篇文档进行了评论或者贡献。