应用补丁到Linux内核

--------------------

原作：Jesper Juhl，2005年8月

最后更新：2006-01-05

一个很常在Linux内核邮件列表中被问到的问题就是如何应用补丁到内核，或者更明确地说，在众多的分支中，一个补丁应该应用到什么基础内核中去。希望此文档能够为你将这个问题解释清楚。

除了如何应用和撤销补丁外，文档还提供了对不同内核树的简要描述（以及如何应用其相应的补丁的例子）。

什么是补丁？

---

补丁就是包含了在一个源树下两个不同版本间差量的小的文档。补丁是使用`diff'程序创建的。

要正确的应用一个补丁你需要知道它是从什么基生成的以及新版本的补丁会对源树带来什么样的改变。这些都应该出现在补丁文件的元数据中或是可以从文件名中推断出来。

如何应用或撤销一个补丁？

---

你需要通过`patch'程序来应用补丁。补丁程序读取diff（或patch）文件然后对其所描述的源树作出改变。

对于Linux内核的补丁是跟其父目录相关生成的，并保持了内核源目录。

这意味着补丁文件内的文件路径包含着其对应生成的内核源目录的名字（或者一些其他目录名，像“a/”或“b/”）。

既然这就不大可能匹配你的本地机器的内核源目录（但常使用info来查看对应生成的未标记的补丁是什么版本的），你应该将其改成你的内核源目录并且在应用的时候去掉路径文件的文件名中的第一个路径元素（`patch'中的-p1参数可以做到这个效果）。

要复原一个之前应用过的补丁，可以对补丁使用-R参数。

因此，如果你是这样应用一个补丁的：

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

你可以这样撤销：

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

如何增加一个patch/diff文件到`patch'

---

有很多种方式可以进行实现（在Linux或者其他类似Unix的操作系统之上）。

在接下来的例子中，我使用stdin以如下的语法将（未压缩形式的）文件添加到补丁：

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

如果你只是想要能够跟随以下的例子并且不想知道更多的使用补丁的方法，你可以从这个章节开始停止阅读。

补丁也可以通过使用-R参数获得文件名，例如：

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

如果你的补丁文件是使用gzip或者bzip2进行压缩的并且你不想在应用之前解压它，你应该这样添加补丁：

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

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

如果你希望在应用之前手动解压补丁文件（假设以及这样做了），那么你只需要在文件上运行gunzip或者bunzip2命令，如下：

gunzip patch-x.y.z.gz

bunzip2 patch-x.y.z.bz2

这就可以得到一个简单的txt补丁文件x.y.z，然后你就可以通过stdin或者-i命令将补丁添加。

补丁还有其他一些很好的参数，-s可以使得除了发生错误以外都保持静默，这就可以防止由于屏幕滚动过快而产生错误，--dry-run可以让补丁只打印出正在发生的事件列表，但实际上不会做出任何改变，--verbose告知补丁打印所完成的事件的更详细的信息。

打补丁时遇到的常见错误

---

当补丁在应用一个补丁文件时，它会以几种不同的方式来验证文件的完整性。

两个补丁检查完整性的基础方法是：检查文件是否像一个有效的补丁文件；检查部分被修改的代码是否符合补丁所提供的内容。

如果补丁遇到一些问题觉得不太对劲，那它就只有两个选项。它可以拒绝应用改变并且放弃，或是找到一种方式使得补丁以最小改变的形式被应用。

一个关于补丁感觉有些“不对劲”的事情而试图修复的例子：如果所有的内容吻合，被更改的内容吻合，但是行数是不同的。这是可能会发生的，举个例子，如果补丁对文件的中部进行了更改，但由于某些原因，文件开头的少数几行的内容进行了添加或者移除。在这样的情况下除了上下移动了一点点之外一切看起来都很正常，因此补丁只是更改了行数就应用了补丁。

当应用一个被进行稍作修改而应用的补丁是，它都会告知你这个补丁是“模糊”应用的。你应该对这样的改变保持担忧，因为即使补丁做对了也并不总会是对的，这个结果可能在某些时候变成错的。

当补丁遇到不能通过模糊修复的改变时，它就会彻底拒绝并给文件留下一个.rej的扩展名（一个拒绝文件）。你可以通过阅读该文档来发现具体什么改变是不能被应用的，如果你想的话，可以通过手动修改的方式来修复它。

如果你没有应用任何第三方补丁到你的内核源，只有来自kernel.org的，并且以正确的顺序应用了补丁，没有自行更改过源文件，那么你就绝不会从补丁处收到模糊或是拒绝的信息。如果你确实看到了这样的信息，那么有很大的可能性是你的本地源树或是补丁文件在某种程度上崩溃了。在这样的情况下，你可能需要重新下载补丁，若还是没有解决，建议你从kernel.org下载一个全新的树重新开始。

让我们再来看一下补丁可能产生的其他的信息。

如果补丁停止并且跳出了“File to patch:”的提示，那么补丁可能找不到需要被打补丁的文件。极大的可能是你忘记使用-p1命令或是处在错误地目录下。还有一种情况是，有一些补丁是需要通过-p0来应用的而不是-p1（阅读补丁文件就能清楚是否属于这种情况——如果是的话，那这就是创造这个补丁的人的错误，并没有很大的影响）。

如果你得到了“Hunk #2 succeeded at 1887 with fuzz 2 (offset 7 lines).”的提示，或是类似于此的信息，那就意味着补丁需要调整改变的位置（在此例子中，需要去除7行使得其能被修复）。

结果文件可能行也可能不行，这取决于文档是否和预期的不同。

这很常发生在当你试图应用一个在和你的内核版本不同的版本之下生成的补丁之时。

如果你收到了“Hunk #3 FAILED at 2387.”的信息，这就意味着补丁不能正确应用并且不能通过模糊处理解决。这会生成一个.rej文件包含导致补丁失败的改变以及一个.orig文件来告诉你不能被改变的原本的内容。

如果是“Reversed (or previously applied) patch detected! Assume -R? [n]”，那么补丁侦测到了一些在补丁内部的改变已经被做出了。

如果你早先已经应用过这个补丁，只是错误地再次进行应用，那你只需要按“n”终止补丁就可以了。如果你已经应用过这个补丁，而你只是想要移除它，却忘了使用-R，那么按“y”就可以移除这个补丁了。这个情况也有可能发生，当补丁的创建者弄错了补丁的源目录和目标目录，那么反过来应用补丁就行了。

类似于“patch: \*\*\*\* unexpected end of file in patch”或是“patch unexpectedly ends in middle of line”的信息，意味着补丁可能对于你想要应用的文件没有意义。可能是由于你下载的已损坏的，或是你的补丁未解压，或是你的补丁文件被邮件客户端或是邮件传输代理在中途损坏了，例如，将很长的一行分成了两行。通常这些警告是很容易修复的，可以通过将分开的两行重新合并即可。

就像我在上面提到的，这些错误是不应该发生的，如果是在将你从kernel.org得到的补丁应用到正确版本且未经修改的源树的情况下。如果你是从kernel.org得到的补丁却仍发生了这些错误，那你就应该考虑可能是你的补丁文件或是源树已损坏，我建议你可以重新下载一个完整的内核数再打补丁。

除了`patch'还有其他选择吗？

---

当然，还有这样一些选择。

你可以使用`interdiff'程序（http://cyberelk.net/tim/patchutils/）来生成一个补丁其代表了两个补丁的差异然后再应用它。

这使得你可以通过一个简单的步骤就去掉一些像2.6.12.2 到 2.6.12.3的东西。interdiff的-z标志甚至可以让你在gzip或bzip2的压缩格式下应用补丁，而不需要使用zcat或是bzcat或是手动解压。

以下是你如何在一个简单的步骤里从2.6.12.2到2.6.12.3：

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

尽管interdiff能够让你省略一两个步骤，但一般来说还是建议你不要这样做，因为interdiff在某些情况下是会出错的。

另一个选择是`ketchup'，它是一个Python脚本，可以自动下载并应用补丁（<http://www.selenic.com/ketchup/>）。

还有这样一些工具：diffstat可以显示这个补丁所做出的总的改变；lsdiff可以显示补丁文件里受影响的文件的一份列表，包括每一个补丁开始的行数（可选）；grepdiff显示被补丁所更改的文件的列表，并且补丁包含给定的正则表达式。

可以从哪里下载补丁？

---

补丁都可以从<http://kernel.org/>下载。大多数的最新的补丁都可以在首页找到链接，但它们都有各自的页面。

2.6.x.y（稳定版）2.6.x版的补丁地址

ftp://ftp.kernel.org/pub/linux/kernel/v2.6/

-rc的补丁地址

ftp://ftp.kernel.org/pub/linux/kernel/v2.6/testing/

-git的补丁地址

ftp://ftp.kernel.org/pub/linux/kernel/v2.6/snapshots/

-mm内核的补丁地址

ftp://ftp.kernel.org/pub/linux/kernel/people/akpm/patches/2.6/

在[ftp.kernel.org](ftp://ftp.kernel.org)你可以使用[ftp.cc.kernel.org](ftp://ftp.cc.kernel.org)，其中cc是国家的代码，这样你就可以从离你地理位置最近的一个镜像网站进行下载，这可以使得你的下载更快，同时，在全球范围内使用更少的带宽，也可以减少kernel.org主服务器上的负载——因此，请尽可能使用镜像。

2.6.x内核

---

这些是Linux发布的基础稳定的版本，数字越大表明是越新发布的。

如果退化或是其他严重错误被发现，那么一个-stable修复补丁会在这里的最上方被发布（见下）。一旦一个新的2.6.x的基础内核发布，就会有一个补丁关于之前的2.6.x的和新的之间的差异。

要应用一个补丁从2.6.11升级到2.6.12，你需要按如下操作（需注意这样的补丁只能应用在2.6.x的内核之上而不能应用在2.6.x.y上——如果你需要从2.6.x.y升级到2.6.x+1，那么你需要先撤销2.6.x.y的补丁）。

下面是一个例子:

# 从 2.6.11 到 2.6.12

$ cd ~/linux-2.6.11 # 进入内核源目录

$ patch -p1 < ../patch-2.6.12 # 应用 2.6.12 补丁

$ cd ..

$ mv linux-2.6.11 linux-2.6.12 # 重命名源目录

# moving from 2.6.11.1 to 2.6.12

$ cd ~/linux-2.6.11.1 # 转换到内核源目录

$ patch -p1 -R < ../patch-2.6.11.1 # 撤销 2.6.11.1 补丁

# 源目录现在为 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 # 重命名源目录

2.6.x.y内核

---

4位数字版本的内核是-stable版的。它们包含为安全问题或是在2.6.x内核中发现的显著退化而产生的小的重要修复。

这推荐给那些需要最新稳定版本却没有兴趣帮助测试开发/实验版本内核的用户。

如果目前没有2.6.x.y版本的内核，那么数字最大的2.6.x版的就是当前的稳定内核版本。

注意：-stable团队通常会做出和最新主线发布相应的渐进式补丁，以下仅涉及非渐进式的。渐进式的可以在这里找到：

ftp://ftp.kernel.org/pub/linux/kernel/v2.6/incr/

这些补丁是非渐进式的意味着2.6.12.3的补丁是不能应用在2.6.12.2的内核源之上的，而是在2.6.12的内核源。因此，想要应用2.6.12.3的补丁，你需要回退2.6.12.2的补丁（你就会剩下一个基础的2.6.12的内核源），然后再应用新的2.6.12.3的补丁。

以下是一个小例子：

$ cd ~/linux-2.6.12.2 # 进入内核源目录

$ patch -p1 -R < ../patch-2.6.12.2 #移除 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 # 重命名内核源目录

-rc内核

---

这些是候选发布版。这些开发版内核是由Linux发布的，当他认为当前的git（内核的源管理工具）树已经处于一个相当正常的状态之下，已适合进行测试了。

这些内核是不稳定的，当你准备运行它的时候就应该预想到其可能不时发生的崩溃。但这仍然是当前主开发版中最稳定的，也会最终成为下一个稳定内核，因此能有很多人来对它进行测试是很重要的。

这对于那些想要帮助测试开发版内核但并不想要运行一些太过实验性质的东西（这样的人可以参考接下来的关于-git和-mm的章节）的人们来说是一个很好的选项。

-rc补丁是非渐进式的，它们应用在2.6.x内核，就像之前描述的2.6.x.y补丁一样。-rcN之前的内核版本的后缀表示了-rc内核最终会变成的内核版本。因此，2.6.13-rc5意味着这是为2.6.13发布的第五个候选，并且这个补丁应该应用在2.6.12的内核源之上。

下面是如何使用这些补丁的三个例子：

#第一个例子是从 2.6.12 到 2.6.13-rc3

$ cd ~/linux-2.6.12 # 进入2.6.12 源目录

$ patch -p1 < ../patch-2.6.13-rc3 # 应用 2.6.13-rc3 补丁

$ cd ..

$ mv linux-2.6.12 linux-2.6.13-rc3 # 重命名源目录

# 从 2.6.13-rc3 到 2.6.13-rc5

$ cd ~/linux-2.6.13-rc3 # 进入 2.6.13-rc3 目录

$ patch -p1 -R < ../patch-2.6.13-rc3 # 移除 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 #重命名源目录

# 最后尝试从2.6.12.3 到 2.6.13-rc5

$ cd ~/linux-2.6.12.3 #进入内核源目录

$ patch -p1 -R < ../patch-2.6.12.3 # 移除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 # 重命名内核源目录

-git内核

---

这些是Linux内核数的日常快照（在git中进行管理，因此得名）。

这些补丁通常是每天发布的，代表了Linux树的当前的状态。他们比-rc内核更具实验性质，因为他们是自动生成的而根本没有经过完整性的检查。

-git补丁是非渐进式的，可以被应用到一个基础2.6.x内核或一个2.6.x-rc内核——这你可以从其名字看出。2.6.12-git1补丁应用到2.6.12内核源，而2.6.13-rc3-git2补丁应用到2.6.13-rc3内核源。

以下是一些简单的如何应用这些补丁的例子：

# 从 2.6.12 到 2.6.12-git1

$ cd ~/linux-2.6.12 # 进入内核源目录

$ 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 # 重命名内核源目录

# 从 2.6.12-git1到2.6.13-rc2-git3

$ cd ~/linux-2.6.12-git1 # 进入内核源目录

$ patch -p1 -R < ../patch-2.6.12-git1 #移除 2.6.12-git1 补丁

# 现在就回到了 2.6.12 内核

$ patch -p1 < ../patch-2.6.13-rc2 # 应用2.6.13-rc2 补丁

# 内核现在就是2.6.13-rc2

$ patch -p1 < ../patch-2.6.13-rc2-git3 # 应用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 # 重命名源目录

-mm内核

---

这些是由Andrew Morton发布的实验性质的内核。

-mm树是作为新要素及实验性补丁的类似于试验场的存在。一旦一个补丁在-mm通过一段时间体现了其价值所在，那么Andrew就会将他推送给Linux以加入到主线中。

尽管鼓励补丁通过-mm树进入到Linux，但这并不是强制的。子系统维护人员（或个人）有时候直接将他们的补丁推送到Linux，即使他们已经在-mm合并然后测试过了（甚至有时候根本没有在-mm进行预先的测试）。

通常情况下，你应该使你的补丁通过-mm进入到主线以确保最大程度的测试。

这个版本是在持续的变化中的并且含有许多实验性质的要素，因为大量的调试补丁是不适合主线的，这也是在整篇文档中最实验性的版本。

这些内核是不适合在那些要求稳定的系统中使用的，并且比运行其他版本的风险性更高（确保你做好了最新的备份——这对于任何实验性质的内核来说都是很重要的，更何况目前的是-mm内核）。

除了所有其他实验性质的补丁外，这些内核还包含了主线-git内核所发布的之中的任何改变。

测试-mm内核是很重要的，其意义在于，在将改变合并到更稳定的Linux树之前，要去除退化，死机，数据崩溃漏洞，构造损坏（以及所有其他漏洞）。但-mm的测试者应该意识到这个树中的损坏是比任何其他的树更为常见的。

-mm内核没有固定的发布进度。但通常少量的-mm内核会在-rc内核（通常为1到3）之间进行发布。-mm内核应用在基础的2.6.x内核（当没有-rc版本内核发布的时候）或是Linux的-rc内核。

下面是一些应用-mm补丁的例子：

# 从 2.6.12 到 2.6.12-mm1

$ cd ~/linux-2.6.12 # 进入 2.6.12 源目录

$ patch -p1 < ../2.6.12-mm1 # 应用 2.6.12-mm1补丁

$ cd ..

$ mv linux-2.6.12 linux-2.6.12-mm1 # 合适地重命名源

# 从 2.6.12-mm1 到 2.6.13-rc3-mm3

$ cd ~/linux-2.6.12-mm1

$ patch -p1 -R < ../2.6.12-mm1 # 移除 2.6.12-mm1 补丁

# 现在为 2.6.12 源

$ patch -p1 < ../patch-2.6.13-rc3 # 应用2.6.13-rc3 补丁

# 现在为 2.6.13-rc3 源

$ patch -p1 < ../2.6.13-rc3-mm3 #应用 2.6.13-rc3-mm3 源

$ cd ..

$ mv linux-2.6.12-mm1 linux-2.6.13-rc3-mm3 # 重命名源目录

以上就是对于一系列的不同内核树的解释。希望你现在能够知道如何应用不同的补丁并且能帮助测试内核。

感谢Randy Dunlap，Rolf Eike Beer，Linus Torvalds，Bodo Eggert，Johannes Stezenbach, Grant Coady，Pavel Machek以及其他我可能遗漏的人们对此文档的审核及贡献。

Applying Patches To The Linux Kernel

------------------------------------

Original by: Jesper Juhl, August 2005

Last update: 2006-01-05

A frequently asked question on the Linux Kernel Mailing List is how to apply

a patch to the kernel or, more specifically, what base kernel a patch for

one of the many trees/branches should be applied to. Hopefully this document

will explain this to you.

In addition to explaining how to apply and revert patches, a brief

description of the different kernel trees (and examples of how to apply

their specific patches) is also provided.

What is a patch?

---

A patch is a small text document containing a delta of changes between two

different versions of a source tree. Patches are created with the `diff'

program.

To correctly apply a patch you need to know what base it was generated from

and what new version the patch will change the source tree into. These

should both be present in the patch file metadata or be possible to deduce

from the filename.

How do I apply or revert a patch?

---

You apply a patch with the `patch' program. The patch program reads a diff

(or patch) file and makes the changes to the source tree described in it.

Patches for the Linux kernel are generated relative to the parent directory

holding the kernel source dir.

This means that paths to files inside the patch file contain the name of the

kernel source directories it was generated against (or some other directory

names like "a/" and "b/").

Since this is unlikely to match the name of the kernel source dir on your

local machine (but is often useful info to see what version an otherwise

unlabeled patch was generated against) you should change into your kernel

source directory and then strip the first element of the path from filenames

in the patch file when applying it (the -p1 argument to `patch' does this).

To revert a previously applied patch, use the -R argument to patch.

So, if you applied a patch like this:

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

You can revert (undo) it like this:

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

How do I feed a patch/diff file to `patch'?

---

This (as usual with Linux and other UNIX like operating systems) can be

done in several different ways.

In all the examples below I feed the file (in uncompressed form) to patch

via stdin using the following syntax:

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

If you just want to be able to follow the examples below and don't want to

know of more than one way to use patch, then you can stop reading this

section here.

Patch can also get the name of the file to use via the -i argument, like

this:

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

If your patch file is compressed with gzip or bzip2 and you don't want to

uncompress it before applying it, then you can feed it to patch like this

instead:

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

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

If you wish to uncompress the patch file by hand first before applying it

(what I assume you've done in the examples below), then you simply run

gunzip or bunzip2 on the file -- like this:

gunzip patch-x.y.z.gz

bunzip2 patch-x.y.z.bz2

Which will leave you with a plain text patch-x.y.z file that you can feed to

patch via stdin or the -i argument, as you prefer.

A few other nice arguments for patch are -s which causes patch to be silent

except for errors which is nice to prevent errors from scrolling out of the

screen too fast, and --dry-run which causes patch to just print a listing of

what would happen, but doesn't actually make any changes. Finally --verbose

tells patch to print more information about the work being done.

Common errors when patching

---

When patch applies a patch file it attempts to verify the sanity of the

file in different ways.

Checking that the file looks like a valid patch file and checking the code

around the bits being modified matches the context provided in the patch are

just two of the basic sanity checks patch does.

If patch encounters something that doesn't look quite right it has two

options. It can either refuse to apply the changes and abort or it can try

to find a way to make the patch apply with a few minor changes.

One example of something that's not 'quite right' that patch will attempt to

fix up is if all the context matches, the lines being changed match, but the

line numbers are different. This can happen, for example, if the patch makes

a change in the middle of the file but for some reasons a few lines have

been added or removed near the beginning of the file. In that case

everything looks good it has just moved up or down a bit, and patch will

usually adjust the line numbers and apply the patch.

Whenever patch applies a patch that it had to modify a bit to make it fit

it'll tell you about it by saying the patch applied with 'fuzz'.

You should be wary of such changes since even though patch probably got it

right it doesn't /always/ get it right, and the result will sometimes be

wrong.

When patch encounters a change that it can't fix up with fuzz it rejects it

outright and leaves a file with a .rej extension (a reject file). You can

read this file to see exactly what change couldn't be applied, so you can

go fix it up by hand if you wish.

If you don't have any third-party patches applied to your kernel source, but

only patches from kernel.org and you apply the patches in the correct order,

and have made no modifications yourself to the source files, then you should

never see a fuzz or reject message from patch. If you do see such messages

anyway, then there's a high risk that either your local source tree or the

patch file is corrupted in some way. In that case you should probably try

re-downloading the patch and if things are still not OK then you'd be advised

to start with a fresh tree downloaded in full from kernel.org.

Let's look a bit more at some of the messages patch can produce.

If patch stops and presents a "File to patch:" prompt, then patch could not

find a file to be patched. Most likely you forgot to specify -p1 or you are

in the wrong directory. Less often, you'll find patches that need to be

applied with -p0 instead of -p1 (reading the patch file should reveal if

this is the case -- if so, then this is an error by the person who created

the patch but is not fatal).

If you get "Hunk #2 succeeded at 1887 with fuzz 2 (offset 7 lines)." or a

message similar to that, then it means that patch had to adjust the location

of the change (in this example it needed to move 7 lines from where it

expected to make the change to make it fit).

The resulting file may or may not be OK, depending on the reason the file

was different than expected.

This often happens if you try to apply a patch that was generated against a

different kernel version than the one you are trying to patch.

If you get a message like "Hunk #3 FAILED at 2387.", then it means that the

patch could not be applied correctly and the patch program was unable to

fuzz its way through. This will generate a .rej file with the change that

caused the patch to fail and also a .orig file showing you the original

content that couldn't be changed.

If you get "Reversed (or previously applied) patch detected! Assume -R? [n]"

then patch detected that the change contained in the patch seems to have

already been made.

If you actually did apply this patch previously and you just re-applied it

in error, then just say [n]o and abort this patch. If you applied this patch

previously and actually intended to revert it, but forgot to specify -R,

then you can say [y]es here to make patch revert it for you.

This can also happen if the creator of the patch reversed the source and

destination directories when creating the patch, and in that case reverting

the patch will in fact apply it.

A message similar to "patch: \*\*\*\* unexpected end of file in patch" or "patch

unexpectedly ends in middle of line" means that patch could make no sense of

the file you fed to it. Either your download is broken, you tried to feed

patch a compressed patch file without uncompressing it first, or the patch

file that you are using has been mangled by a mail client or mail transfer

agent along the way somewhere, e.g., by splitting a long line into two lines.

Often these warnings can easily be fixed by joining (concatenating) the

two lines that had been split.

As I already mentioned above, these errors should never happen if you apply

a patch from kernel.org to the correct version of an unmodified source tree.

So if you get these errors with kernel.org patches then you should probably

assume that either your patch file or your tree is broken and I'd advise you

to start over with a fresh download of a full kernel tree and the patch you

wish to apply.

Are there any alternatives to `patch'?

---

Yes there are alternatives.

You can use the `interdiff' program (http://cyberelk.net/tim/patchutils/) to

generate a patch representing the differences between two patches and then

apply the result.

This will let you move from something like 2.6.12.2 to 2.6.12.3 in a single

step. The -z flag to interdiff will even let you feed it patches in gzip or

bzip2 compressed form directly without the use of zcat or bzcat or manual

decompression.

Here's how you'd go from 2.6.12.2 to 2.6.12.3 in a single step:

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

Although interdiff may save you a step or two you are generally advised to

do the additional steps since interdiff can get things wrong in some cases.

Another alternative is `ketchup', which is a python script for automatic

downloading and applying of patches (http://www.selenic.com/ketchup/).

Other nice tools are diffstat, which shows a summary of changes made by a

patch; lsdiff, which displays a short listing of affected files in a patch

file, along with (optionally) the line numbers of the start of each patch;

and grepdiff, which displays a list of the files modified by a patch where

the patch contains a given regular expression.

Where can I download the patches?

---

The patches are available at http://kernel.org/

Most recent patches are linked from the front page, but they also have

specific homes.

The 2.6.x.y (-stable) and 2.6.x patches live at

ftp://ftp.kernel.org/pub/linux/kernel/v2.6/

The -rc patches live at

ftp://ftp.kernel.org/pub/linux/kernel/v2.6/testing/

The -git patches live at

ftp://ftp.kernel.org/pub/linux/kernel/v2.6/snapshots/

The -mm kernels live at

ftp://ftp.kernel.org/pub/linux/kernel/people/akpm/patches/2.6/

In place of ftp.kernel.org you can use ftp.cc.kernel.org, where cc is a

country code. This way you'll be downloading from a mirror site that's most

likely geographically closer to you, resulting in faster downloads for you,

less bandwidth used globally and less load on the main kernel.org servers --

these are good things, so do use mirrors when possible.

The 2.6.x kernels

---

These are the base stable releases released by Linus. The highest numbered

release is the most recent.

If regressions or other serious flaws are found, then a -stable fix patch

will be released (see below) on top of this base. Once a new 2.6.x base

kernel is released, a patch is made available that is a delta between the

previous 2.6.x kernel and the new one.

To apply a patch moving from 2.6.11 to 2.6.12, you'd do the following (note

that such patches do \*NOT\* apply on top of 2.6.x.y kernels but on top of the

base 2.6.x kernel -- if you need to move from 2.6.x.y to 2.6.x+1 you need to

first revert the 2.6.x.y patch).

Here are some examples:

# moving from 2.6.11 to 2.6.12

$ cd ~/linux-2.6.11 # change to kernel source dir

$ patch -p1 < ../patch-2.6.12 # apply the 2.6.12 patch

$ cd ..

$ mv linux-2.6.11 linux-2.6.12 # rename source dir

# moving from 2.6.11.1 to 2.6.12

$ cd ~/linux-2.6.11.1 # change to kernel source dir

$ patch -p1 -R < ../patch-2.6.11.1 # revert the 2.6.11.1 patch

# source dir is now 2.6.11

$ patch -p1 < ../patch-2.6.12 # apply new 2.6.12 patch

$ cd ..

$ mv linux-2.6.11.1 linux-2.6.12 # rename source dir

The 2.6.x.y kernels

---

Kernels with 4-digit versions are -stable kernels. They contain small(ish)

critical fixes for security problems or significant regressions discovered

in a given 2.6.x kernel.

This is the recommended branch for users who want the most recent stable

kernel and are not interested in helping test development/experimental

versions.

If no 2.6.x.y kernel is available, then the highest numbered 2.6.x kernel is

the current stable kernel.

note: the -stable team usually do make incremental patches available as well

as patches against the latest mainline release, but I only cover the

non-incremental ones below. The incremental ones can be found at

ftp://ftp.kernel.org/pub/linux/kernel/v2.6/incr/

These patches are not incremental, meaning that for example the 2.6.12.3

patch does not apply on top of the 2.6.12.2 kernel source, but rather on top

of the base 2.6.12 kernel source .

So, in order to apply the 2.6.12.3 patch to your existing 2.6.12.2 kernel

source you have to first back out the 2.6.12.2 patch (so you are left with a

base 2.6.12 kernel source) and then apply the new 2.6.12.3 patch.

Here's a small example:

$ cd ~/linux-2.6.12.2 # change into the kernel source dir

$ patch -p1 -R < ../patch-2.6.12.2 # revert the 2.6.12.2 patch

$ patch -p1 < ../patch-2.6.12.3 # apply the new 2.6.12.3 patch

$ cd ..

$ mv linux-2.6.12.2 linux-2.6.12.3 # rename the kernel source dir

The -rc kernels

---

These are release-candidate kernels. These are development kernels released

by Linus whenever he deems the current git (the kernel's source management

tool) tree to be in a reasonably sane state adequate for testing.

These kernels are not stable and you should expect occasional breakage if

you intend to run them. This is however the most stable of the main

development branches and is also what will eventually turn into the next

stable kernel, so it is important that it be tested by as many people as

possible.

This is a good branch to run for people who want to help out testing

development kernels but do not want to run some of the really experimental

stuff (such people should see the sections about -git and -mm kernels below).

The -rc patches are not incremental, they apply to a base 2.6.x kernel, just

like the 2.6.x.y patches described above. The kernel version before the -rcN

suffix denotes the version of the kernel that this -rc kernel will eventually

turn into.

So, 2.6.13-rc5 means that this is the fifth release candidate for the 2.6.13

kernel and the patch should be applied on top of the 2.6.12 kernel source.

Here are 3 examples of how to apply these patches:

# first an example of moving from 2.6.12 to 2.6.13-rc3

$ cd ~/linux-2.6.12 # change into the 2.6.12 source dir

$ patch -p1 < ../patch-2.6.13-rc3 # apply the 2.6.13-rc3 patch

$ cd ..

$ mv linux-2.6.12 linux-2.6.13-rc3 # rename the source dir

# now let's move from 2.6.13-rc3 to 2.6.13-rc5

$ cd ~/linux-2.6.13-rc3 # change into the 2.6.13-rc3 dir

$ patch -p1 -R < ../patch-2.6.13-rc3 # revert the 2.6.13-rc3 patch

$ patch -p1 < ../patch-2.6.13-rc5 # apply the new 2.6.13-rc5 patch

$ cd ..

$ mv linux-2.6.13-rc3 linux-2.6.13-rc5 # rename the source dir

# finally let's try and move from 2.6.12.3 to 2.6.13-rc5

$ cd ~/linux-2.6.12.3 # change to the kernel source dir

$ patch -p1 -R < ../patch-2.6.12.3 # revert the 2.6.12.3 patch

$ patch -p1 < ../patch-2.6.13-rc5 # apply new 2.6.13-rc5 patch

$ cd ..

$ mv linux-2.6.12.3 linux-2.6.13-rc5 # rename the kernel source dir

The -git kernels

---

These are daily snapshots of Linus' kernel tree (managed in a git

repository, hence the name).

These patches are usually released daily and represent the current state of

Linus's tree. They are more experimental than -rc kernels since they are

generated automatically without even a cursory glance to see if they are

sane.

-git patches are not incremental and apply either to a base 2.6.x kernel or

a base 2.6.x-rc kernel -- you can see which from their name.

A patch named 2.6.12-git1 applies to the 2.6.12 kernel source and a patch

named 2.6.13-rc3-git2 applies to the source of the 2.6.13-rc3 kernel.

Here are some examples of how to apply these patches:

# moving from 2.6.12 to 2.6.12-git1

$ cd ~/linux-2.6.12 # change to the kernel source dir

$ patch -p1 < ../patch-2.6.12-git1 # apply the 2.6.12-git1 patch

$ cd ..

$ mv linux-2.6.12 linux-2.6.12-git1 # rename the kernel source dir

# moving from 2.6.12-git1 to 2.6.13-rc2-git3

$ cd ~/linux-2.6.12-git1 # change to the kernel source dir

$ patch -p1 -R < ../patch-2.6.12-git1 # revert the 2.6.12-git1 patch

# we now have a 2.6.12 kernel

$ patch -p1 < ../patch-2.6.13-rc2 # apply the 2.6.13-rc2 patch

# the kernel is now 2.6.13-rc2

$ patch -p1 < ../patch-2.6.13-rc2-git3 # apply the 2.6.13-rc2-git3 patch

# the kernel is now 2.6.13-rc2-git3

$ cd ..

$ mv linux-2.6.12-git1 linux-2.6.13-rc2-git3 # rename source dir

The -mm kernels

---

These are experimental kernels released by Andrew Morton.

The -mm tree serves as a sort of proving ground for new features and other

experimental patches.

Once a patch has proved its worth in -mm for a while Andrew pushes it on to

Linus for inclusion in mainline.

Although it's encouraged that patches flow to Linus via the -mm tree, this

is not always enforced.

Subsystem maintainers (or individuals) sometimes push their patches directly

to Linus, even though (or after) they have been merged and tested in -mm (or

sometimes even without prior testing in -mm).

You should generally strive to get your patches into mainline via -mm to

ensure maximum testing.

This branch is in constant flux and contains many experimental features, a

lot of debugging patches not appropriate for mainline etc., and is the most

experimental of the branches described in this document.

These kernels are not appropriate for use on systems that are supposed to be

stable and they are more risky to run than any of the other branches (make

sure you have up-to-date backups -- that goes for any experimental kernel but

even more so for -mm kernels).

These kernels in addition to all the other experimental patches they contain

usually also contain any changes in the mainline -git kernels available at

the time of release.

Testing of -mm kernels is greatly appreciated since the whole point of the

tree is to weed out regressions, crashes, data corruption bugs, build

breakage (and any other bug in general) before changes are merged into the

more stable mainline Linus tree.

But testers of -mm should be aware that breakage in this tree is more common

than in any other tree.

The -mm kernels are not released on a fixed schedule, but usually a few -mm

kernels are released in between each -rc kernel (1 to 3 is common).

The -mm kernels apply to either a base 2.6.x kernel (when no -rc kernels

have been released yet) or to a Linus -rc kernel.

Here are some examples of applying the -mm patches:

# moving from 2.6.12 to 2.6.12-mm1

$ cd ~/linux-2.6.12 # change to the 2.6.12 source dir

$ patch -p1 < ../2.6.12-mm1 # apply the 2.6.12-mm1 patch

$ cd ..

$ mv linux-2.6.12 linux-2.6.12-mm1 # rename the source appropriately

# moving from 2.6.12-mm1 to 2.6.13-rc3-mm3

$ cd ~/linux-2.6.12-mm1

$ patch -p1 -R < ../2.6.12-mm1 # revert the 2.6.12-mm1 patch

# we now have a 2.6.12 source

$ patch -p1 < ../patch-2.6.13-rc3 # apply the 2.6.13-rc3 patch

# we now have a 2.6.13-rc3 source

$ patch -p1 < ../2.6.13-rc3-mm3 # apply the 2.6.13-rc3-mm3 patch

$ cd ..

$ mv linux-2.6.12-mm1 linux-2.6.13-rc3-mm3 # rename the source dir

This concludes this list of explanations of the various kernel trees.

I hope you are now clear on how to apply the various patches and help testing

the kernel.

Thank you's to Randy Dunlap, Rolf Eike Beer, Linus Torvalds, Bodo Eggert,

Johannes Stezenbach, Grant Coady, Pavel Machek and others that I may have

forgotten for their reviews and contributions to this document.