【2.6.31】内核编译指南

Translated By: openspace
Date : 2009-11-1

linux 内核发行版 2.6.xx <http://kernel.org/>

这是 Linux 内核 2.6 版本的发行注记。请认真阅读，因为这里告诉你所有相关的信息，描述如何安装内核、以及如果出现错误该如何处理。

什么是 Linux？

Linux 是 Unix 操作系统的一个克隆颁布，由 Linus Torvalds 和一些通过网络松散组织起来的黑客门从头开始构建。它的目标是兼容 POSIX 和 Single UNIX Specification。

Linux 具有你可以想象得到的在现在完全成熟的 Unix 上所具有的所有特征，包括真正的多任务、虚拟内存、共享库、按需加载、共享的写时复制可执行程序、合理的内存管理、以及包含 IPv4 和 IPv6 在内的多个网络协议栈。

Linux 在 GNU 通用公共许可证下发布——查看文件 COPYING 获取更多细节。

Linux 可以运行于哪些硬件系统之上？

虽然最初是为基于 32 位 x86 的 PC（386 或者更高）开发的，现在的 Linux 可以运行在（至少）Compaq Alpha AXP、Sun SPARC、UltraSPARC、Motorola 68000、 PowerPC、PowerPC64、ARM、Hitachi SuperH、Cell、IBM S/390、MIPS、HP PA-RISC、Intel IA-64、DEC VAX、AMD x86-64、AXIS CRIS、Xtensa、AVR32 和 Renesas M32R 体系架构的处理器之上。

可以很容易地将 Linux 移植到大多数通用的 32 位和 64 位架构上，只要它们具备基于页面的存储管理单元（PMMU）并移植了 GNU C 编译器（gcc）（属于 GNU Compiler Collection，GCC）。Linux 也可以被移植到不具备 PMMU 的一些架构上，虽然功能会明显地受到一些限制。Linux 也可以移植到自身之上。你可以以用户态应用程序的方式来运行内核——这称为用户态 Linux（UML）。

文档

- 网上有大量关于 Linux 和 UNIX 的电子文档，在这些方面也有大量的书籍。有些是特定于 Linux 的，也有些是通用的 UNIX 相关的。我建议先查看任意 Linux FTP 站点子目录下的有关 LDP（Linux 文档项目）的书籍。本篇不作为讲解系统的文档：实际上有更好的资源可以参考。

- 在 documentation/子目录下有大量的 README 文件：例如，这些文件包含与内核相关的关于一些驱动程序的安装注记。查看 Documentation/00-INDEX 来获取每个文件所包

含信息的列表。请阅读文件 Changes，该文件包含了关于可能导致内核更新失败的问题的信息。

- Documentation/DocBook/子目录下包含适用于内核开发者和用户的一些指南。这些指南可以以多种格式显示：PostScript(.ps)、PDF、HTML、man 手册页，以及其他格式。安装后，执行命令“make psdocs”、“make pdfdocs”、“make htmldocs”或者“make mandocs”可以生成指定格式的文档。

安装内核源码

- 如果你要安装完整的源码，将内核 tar 包放在你拥有访问权限的目录下（例如，你的 home 目录）并解压缩：

```
gzip -cd linux-2.6.XX.tar.gz | tar xvf -
```

或者

```
bzip2 -dc linux-2.6.XX.tar.bz2 | tar xvf -
```

将“XX”替换为最新内核的版本号。

不要使用/usr/src/linux！这里包含（通常不完整）一组由库的头文件使用的内核头文件。这些头文件要与库代码匹配，并且不应该因为任何偶然的 kernel-du-jour 而导致混乱。

- 你还可以通过打补丁在 2.6.xx 版本之间进行升级。补丁以传统的 gzip 格式和新的 bzip2 格式发放。要通过打补丁安装，首先获取所有新的补丁文件，然后进入内核源码（linux-2.6.xx）的顶层目录，执行下列命令：

```
gzip -cd ../patch-2.6.xx.gz | patch -p1
```

或者

```
bzip2 -dc ../patch-2.6.xx.bz2 | patch -p1
```

（按顺序重复处理版本号大于当前源码树的 xx 版本）就可以了。你可以删除备份文件（xxx~或者 xxx.orig），并确保没有失败的补丁（xxx#或者 xxx.rej）。如果有补丁失败，那么就是你或者我在操作时出了错。

与 2.6.x 内核所使用的补丁不同的是，用于 2.6.x.y 内核（称为-stable 内核）的补丁不是增量的，而是可以直接合并到基本的 2.6.x 内核中。请查看 Documentation/applying-patches.txt 获取更多信息

另外，脚本 patch-kernel 可以自动化这个过程。它确定当前内核版本，然后合并所有找到的补丁。

```
linux/scripts/patch-kernel linux
```

上面命令的第一个参数是内核源码的位置。当前目录下的补丁被合并入内核，但是可以通过第二个参数指定一个替换目录。

- 如果是通过稳定版的补丁（例如 patch-2.6.xx.y）在不同版本之间升级，注意这些“dot-releases”不是增量的，它们只能用于 2.6.xx 基础内核树。例如，如果基础内核为

2.6.12，而你要应用补丁 2.6.12.3，那么你事先不能应用补丁 2.6.12.1 和 2.6.12.2。同样的，如果使用的内核为 2.6.12.2，而你打算升级到 2.6.12.3，那么你必须在应用补丁 2.6.12.3 之前先撤销 2.6.12.2 补丁（即 patch -R）。

可以阅读 Documentation/applying-patches.txt 获取更多信息。

- 确保不会有旧的.o 文件和依赖存在：

```
cd linux
make mrproper
```

现在你可以正确地安装源码了。

软件要求

编译和运行 2.6.xx 内核需要各种软件包的最新版本。查看 Documentation/Changes 了解对这些软件包最低版本的要求以及如何进行更新。注意，如果使用非常旧的版本的软件包会产生难以跟踪的间接错误，所以不要认为在构建或者操作过程中出现明显的错误时只要更新软件包就可以。

构建存放内核的目录

当编译内核时，缺省地，所有的输出文件将与对应的内核源码文件存放在一起。使用选项“make O=output/dir”允许你指定一个可选目录来存放输出文件（包括.config）。例如：

```
内核源码: /usr/src/linux-2.6.N
构建目录: /home/name/build/kernel
```

要配置和构建内核，执行如下命令：

```
cd /usr/src/linux-2.6.N
make O=/home/name/build/kernel menuconfig
make O=/home/name/build/kernel
sudo make O=/home/name/build/kernel modules_install install
```

请注意：如果使用了选项“O=output/dir”，那么每次调用 make 时都必须使用该选项。

配置内核

不要跳过这一步，即便是针对次版本进行升级。在每个发行版本中都会加入新的配置选项，如果没有正确地设置配置文件，会出现一些奇怪的问题。如果你打算花费尽量少的时间来将你现有的配置用于新版本，使用“make oldconfig”，这样你只需要设置新选项即可。

- 可选的配置命令包括：

```
"make config"      普通文本界面
"make menuconfig"  基于彩色菜单、单选列表和对话框的文本界面
"make xconfig"     基于 X window (Qt)的配置工具
"make gconfig"     基于 X window (Gtk)的配置工具
"make oldconfig"   缺省所有的选项基于现有的./.config 文件的内容，询问用户对新的配置选项的设置
```

"make silentoldconfig"
类似于上一条命令，但是没有将已有的设置显示在屏幕上。此外还更新依赖关系
"make defconfig" 基于 arch/$ARCH/defconfig 或者 arch/$ARCH/configs/${PLATFORM}_defconfig 中的缺省选项配置来创建一个./.config 文件，如何选择依赖于体系结构
"make ${PLATFORM}_defconfig"
基于 arch/$ARCH/configs/${PLATFORM}_defconfig 中的缺省选项配置来创建一个./.config 文件。使用“make help”获取对应体系结构的所有平台的列表
"make allyesconfig" 创建一个选项值尽可能设置为'y'的./.config 文件
"make allmodconfig" 创建一个选项值尽可能设置为'm'的./.config 文件
"make allnoconfig" 创建一个选项值尽可能设置为'n'的./.config 文件
"make randconfig" 创建一个选项值为随机值的./.config 文件

使用 Documentation/kbuild/kconfig.txt 中的 Linux 内核配置工具可以获取更多信息。

关于“make config”：

- 加入无关的驱动程序会让内核臃肿，而且在某些情况下会出问题：探测一个不存在的控制器时可能将驱动程序关联到其他控制器。
- 编译内核时将“Processor type”设置为高于 386 会导致内核不能在 386 机器上工作。内核在启动时会对处理器类型进行检测并停止工作。
- 如果内核编译进了仿真数学协处理器，内核还是会使用模拟的即便有一个物理的协处理器：这时永远不会使用物理的数学协处理器。内核会稍微大一些，但是它能运行于不同的机器上，不管这些机器上是否有物理的数学协处理器。
- 配置“kernel hacking”会导致内核大一些，或者运行会慢一些（或者两者兼有），而且内核会不稳定，因为配置了一些例程会打断出错代码来查找内核的错误（kmalloc()）。因此多数情况下对选项 "development", "experimental", or "debugging" features 要回答'n'。

编译内核

- 确认安装了最新的 gcc 3.2。关于更多信息，查看 Documentation/Changes。

  请注意：你仍然可以在新内核上运行用户态程序 a.out。

- 执行“make”构建一个压缩的内核映像。如果已经安装了 lilo 来配合内核的 makefile 文件，可以执行“make install”；但是最好先检查一下 lilo 的设置。

  要进行实际的安装必须有 root 权限，但是正常的构建不会有此要求。root 自有它的用武之处。

- 如果将内核的一部分配置为`modules`，那么还需要执行命令“make modules_install”。
- 输出内核编译/构建的详细信息：

通常内核构建系统以沉默模式运行（不是完全沉默）。但是，有时你或者内核开发人员需要查看执行的编译、链接或者其他命令。要是这样，使用“verbose”构建模式。在“make”命令中插入“V=1”即可。例如：

```
make V=1 all
```

要让构建系统输出某个目标重新构建的理由，使用“V=2”。缺省为“V=0”。

- 保存一个内核的副本，以免出现错误。对于开发版本来说这很重要，因为新版本包含一些没有调试过的代码。

  同时务必保存对应版本的模块的副本。如果安装跟当前工作的内核版本号相同的新内核，在执行“make modules_install”之前构造模块目录的备份。还有一种方式是，在编译内核之前，使用内核配置选项“LOCALVERSION”在工作内核版本后添加一个唯一的后缀。可以在菜单“General Setup”设置 LOCALVERSION。

- 要启动新内核，需要将新内核映像（例如，编译后得到的.../linux/arch/i386/boot/bzImage）复制到放置工作内核的地方。

- 已经不支持在不使用 bootloader 例如 LILO 的情况下而直接从软盘启动内核。

  如果从硬盘启动 Linux，可能你使用的是 LILO，而 LILO 使用的内核映像由文件/etc/lilo.conf 指定。通常内核映像文件为/vmlinuz、/boot/vmlinuz、/bzImage 或者/boot/bzImage。要使用新内核，先保存旧映像文件的一个副本，然后用新映像覆盖旧的映像文件。接下来，必须返回到 LILO 更新加载的映像！！如果不这样做，你将无法启动新的内核映像。

  通常重新安装 LILO 需要运行/sbin/lilo。可以编辑/etc/lilo.conf 来指定旧的内核映像（例如/vmlinux.old），以备新内核不能工作。查看 LILO 文档获取更多信息。

  重新安装 LILO 后，可以动手了。关闭系统，重启，使用新内核!

  如果要改变内核映像中的缺省根设备、视频模式、ramdisk 大小等等，使用程序‘rdev’（或者适当的时候使用 LILO 的引导选项）。改变这些参数不需要重新编译内核。

- 重启进入新内核，享受新的旅程。

如果出现错误

- 如果错误可能是内核 bug 引起的，请查看 MAINTAINERS，看看是否有专人负责出问题的内核部分。若是没有，那么接下来最好把它们发给我（torvalds@linux-foundation.org），也可以发到其它相关的邮件列表或者新闻组。

- 在 bug 报告中，请说明内核的详细信息、如何重现问题、以及你所使用的系统配置（用你的常识描述）。如果是一个新发现的问题，说明这一点，如果是一个已有的问题，请尽量在第一次发现时告诉我。

- 如果 bug 在屏幕上或者系统日志中产生一条类似这样的消息

```
unable to handle kernel paging request at address C0000010
Oops: 0002
```

```
EIP:   0010:XXXXXXXX
eax: xxxxxxxx   ebx: xxxxxxxx   ecx: xxxxxxxx   edx: xxxxxxxx
esi: xxxxxxxx   edi: xxxxxxxx   ebp: xxxxxxxx
ds: xxxx  es: xxxx  fs: xxxx  gs: xxxx
Pid: xx, process nr: xx
xx xx xx xx xx xx xx xx xx xx
```

或者类似的内核调试信息，请准确地复制下来。在你看来，dump 可能是难于理解的，但是它包含的信息有助于调试错误。dump 信息上面的文字描述也很重要：它描述了为什么内核会 dump（在上面的例子中是由于一个错误的内核指针）。如何利用 dump 信息请参考 Documentation/oops-tracing.txt。

- 如果编译内核时使用了 CONFIG_KALLSYMS，那么可以将 dump 原样发送，否则必须使用程序“ksymoops”将 dump 信息变得更容易使用（通常建议编译时加入 CONFIG_KALLSYMS 选项） 。该工具可以从 ftp://ftp.<country>.kernel.org/pub/linux/utils/kernel/ksymoops/下载。你还可以手工查看 dump 信息。

- 在调试类似上面的 dump 信息时，如果能弄清楚 EIP 的值是什么意思会非常有帮助。类似这样的 16 进制值并不能给我或者其他人提供帮助：它依赖于你自己的内核的设置。你需要做的是拿掉 EIP 那行的 16 进制值（忽略“0010:”），然后查找内核的名字列表，看看偏移地址属于哪个内核函数。

  要找到内核函数的名字，需要到产生错误的内核的二进制映像中查找。即‘linux/vmlinux’。要抽取名字列表来匹配内核崩溃时的 EIP，执行下面的命令：

```
nm vmlinux | sort | less
```

  这样你会得到一个升序排列的内核地址列表，可以在该列表中找到包含偏移地址的函数。注意内核调试信息给出的地址不需要与函数地址完全匹配（实际上很少会这样），所以不能使用‘grep’：但是列表中会给出每个内核函数的起始点，因此可以查找起始地址低于你给出的偏移地址而其后又有高于偏移地址的函数的函数就可以了。实际上，一个好的做法是在问题报告中包含一些“context”，即加入对应位置前后的一些代码。

  如果由于某种原因你无法进行上述操作（使用了预编译内核映像或者类似的映像文件），那么尽可能详细的告诉我你的系统配置。请参考 REPORTING-BUGS 获取更多细节信息。

- 还有一种方法，对工作内核使用 gdb。（只读；例如你不能改变值或者设置断点）要这样做，首先编译内核时使用-g 选项；调整 arch/i386/Makefile，然后执行“make clean”。还需要设置选项 CONFIG_PROC_FS（通过“make config”）。

  重启到新内核后，执行“gdb vmlinux /proc/kcore”。这时你可以使用所有常用的 gdb 命令。查看系统崩溃位置的命令是“l *0xXXXXXXXX”。（将 XXX 替换为 EIP 的值。）

  对于非运行态的内核使用 gdb 会失败，因为 gdb（错误地）忽视了内核编译的起始偏移量。

【原文】

```
1            Linux kernel release 2.6.xx <http://kernel.org/>
2
3       These are the release notes for Linux version 2.6.  Read them carefully,
4       as they tell you what this is all about, explain how to install the
5       kernel, and what to do if something goes wrong.
6
7       WHAT IS LINUX?
8
9         Linux is a clone of the operating system Unix, written from scratch by
10        Linus Torvalds with assistance from a loosely-knit team of hackers across
11        the Net. It aims towards POSIX and Single UNIX Specification compliance.
12
13        It has all the features you would expect in a modern fully-fledged Unix,
14        including true multitasking, virtual memory, shared libraries, demand
15        loading, shared copy-on-write executables, proper memory management,
16        and multistack networking including IPv4 and IPv6.
17
18        It is distributed under the GNU General Public License - see the
19        accompanying COPYING file for more details.
20
21      ON WHAT HARDWARE DOES IT RUN?
22
23        Although originally developed first for 32-bit x86-based PCs (386 or higher),
24        today Linux also runs on (at least) the Compaq Alpha AXP, Sun SPARC and
25        UltraSPARC, Motorola 68000, PowerPC, PowerPC64, ARM, Hitachi SuperH, Cell,
26        IBM S/390, MIPS, HP PA-RISC, Intel IA-64, DEC VAX, AMD x86-64, AXIS CRIS,
27        Xtensa, AVR32 and Renesas M32R architectures.
28
29        Linux is easily portable to most general-purpose 32- or 64-bit architectures
30        as long as they have a paged memory management unit (PMMU) and a port of the
31        GNU C compiler (gcc) (part of The GNU Compiler Collection, GCC). Linux has
32        also been ported to a number of architectures without a PMMU, although
33        functionality is then obviously somewhat limited.
34        Linux has also been ported to itself. You can now run the kernel as a
35        userspace application - this is called UserMode Linux (UML).
36
37      DOCUMENTATION:
38
39       - There is a lot of documentation available both in electronic form on
40         the Internet and in books, both Linux-specific and pertaining to
41         general UNIX questions.  I'd recommend looking into the documentation
42         subdirectories on any Linux FTP site for the LDP (Linux Documentation
43         Project) books.  This README is not meant to be documentation on the
44         system: there are much better sources available.
45
46       - There are various README files in the Documentation/ subdirectory:
47         these typically contain kernel-specific installation notes for some
48         drivers for example. See Documentation/00-INDEX for a list of what
49         is contained in each file.  Please read the Changes file, as it
50         contains information about the problems, which may result by upgrading
51         your kernel.
```

- The Documentation/DocBook/ subdirectory contains several guides for
  kernel developers and users. These guides can be rendered in a
  number of formats: PostScript (.ps), PDF, HTML, & man-pages, among others.
  After installation, "make psdocs", "make pdfdocs", "make htmldocs",
  or "make mandocs" will render the documentation in the requested format.

INSTALLING the kernel source:

- If you install the full sources, put the kernel tarball in a
  directory where you have permissions (eg. your home directory) and
  unpack it:

```
            gzip -cd linux-2.6.XX.tar.gz | tar xvf -
```

  or

```
            bzip2 -dc linux-2.6.XX.tar.bz2 | tar xvf -
```

  Replace "XX" with the version number of the latest kernel.

  Do NOT use the /usr/src/linux area! This area has a (usually
  incomplete) set of kernel headers that are used by the library header
  files. They should match the library, and not get messed up by
  whatever the kernel-du-jour happens to be.

- You can also upgrade between 2.6.xx releases by patching. Patches are
  distributed in the traditional gzip and the newer bzip2 format. To
  install by patching, get all the newer patch files, enter the
  top level directory of the kernel source (linux-2.6.xx) and execute:

```
            gzip -cd ../patch-2.6.xx.gz | patch -p1
```

  or

```
            bzip2 -dc ../patch-2.6.xx.bz2 | patch -p1
```

  (repeat xx for all versions bigger than the version of your current
  source tree, _in_order_) and you should be ok. You may want to remove
  the backup files (xxx~ or xxx.orig), and make sure that there are no
  failed patches (xxx# or xxx.rej). If there are, either you or me has
  made a mistake.

  Unlike patches for the 2.6.x kernels, patches for the 2.6.x.y kernels
  (also known as the -stable kernels) are not incremental but instead apply
  directly to the base 2.6.x kernel. Please read
  Documentation/applying-patches.txt for more information.

  Alternatively, the script patch-kernel can be used to automate this
  process. It determines the current kernel version and applies any
  patches found.

```
            linux/scripts/patch-kernel linux
```

```
105       The first argument in the command above is the location of the
106       kernel source.  Patches are applied from the current directory, but
107       an alternative directory can be specified as the second argument.
108
109       - If you are upgrading between releases using the stable series patches
110       (for example, patch-2.6.xx.y), note that these "dot-releases" are
111       not incremental and must be applied to the 2.6.xx base tree. For
112       example, if your base kernel is 2.6.12 and you want to apply the
113       2.6.12.3 patch, you do not and indeed must not first apply the
114       2.6.12.1 and 2.6.12.2 patches. Similarly, if you are running kernel
115       version 2.6.12.2 and want to jump to 2.6.12.3, you must first
116       reverse the 2.6.12.2 patch (that is, patch -R) _before_ applying
117       the 2.6.12.3 patch.
118       You can read more on this in Documentation/applying-patches.txt
119
120       - Make sure you have no stale .o files and dependencies lying around:
121
122                   cd linux
123                   make mrproper
124
125       You should now have the sources correctly installed.
126
127   SOFTWARE REQUIREMENTS
128
129       Compiling and running the 2.6.xx kernels requires up-to-date
130       versions of various software packages.  Consult
131       Documentation/Changes for the minimum version numbers required
132       and how to get updates for these packages.  Beware that using
133       excessively old versions of these packages can cause indirect
134       errors that are very difficult to track down, so don't assume that
135       you can just update packages when obvious problems arise during
136       build or operation.
137
138   BUILD directory for the kernel:
139
140       When compiling the kernel all output files will per default be
141       stored together with the kernel source code.
142       Using the option "make O=output/dir" allow you to specify an alternate
143       place for the output files (including .config).
144       Example:
145         kernel source code:     /usr/src/linux-2.6.N
146         build directory:        /home/name/build/kernel
147
148       To configure and build the kernel use:
149       cd /usr/src/linux-2.6.N
150       make O=/home/name/build/kernel menuconfig
151       make O=/home/name/build/kernel
152       sudo make O=/home/name/build/kernel modules_install install
153
154       Please note: If the 'O=output/dir' option is used then it must be
155       used for all invocations of make.
156
157   CONFIGURING the kernel:
```

```
158
159       Do not skip this step even if you are only upgrading one minor
160       version.  New configuration options are added in each release, and
161       odd problems will turn up if the configuration files are not set up
162       as expected.  If you want to carry your existing configuration to a
163       new version with minimal work, use "make oldconfig", which will
164       only ask you for the answers to new questions.
165
166     - Alternate configuration commands are:
167           "make config"      Plain text interface.
168           "make menuconfig"  Text based color menus, radiolists & dialogs.
169           "make xconfig"     X windows (Qt) based configuration tool.
170           "make gconfig"     X windows (Gtk) based configuration tool.
171           "make oldconfig"   Default all questions based on the contents of
172                              your existing ./.config file and asking about
173                              new config symbols.
174           "make silentoldconfig"
175                              Like above, but avoids cluttering the screen
176                              with questions already answered.
177                              Additionally updates the dependencies.
178           "make defconfig"   Create a ./.config file by using the default
179                              symbol values from either arch/$ARCH/defconfig
180                              or arch/$ARCH/configs/${PLATFORM}_defconfig,
181                              depending on the architecture.
182           "make ${PLATFORM}_defconfig"
183                             Create a ./.config file by using the default
184                             symbol values from
185                             arch/$ARCH/configs/${PLATFORM}_defconfig.
186                             Use "make help" to get a list of all available
187                             platforms of your architecture.
188           "make allyesconfig"
189                              Create a ./.config file by setting symbol
190                              values to 'y' as much as possible.
191           "make allmodconfig"
192                              Create a ./.config file by setting symbol
193                              values to 'm' as much as possible.
194           "make allnoconfig" Create a ./.config file by setting symbol
195                              values to 'n' as much as possible.
196           "make randconfig"  Create a ./.config file by setting symbol
197                              values to random values.
198
199       You can find more information on using the Linux kernel config tools
200       in Documentation/kbuild/kconfig.txt.
201
202           NOTES on "make config":
203           - having unnecessary drivers will make the kernel bigger, and can
204             under some circumstances lead to problems: probing for a
205             nonexistent controller card may confuse your other controllers
206           - compiling the kernel with "Processor type" set higher than 386
207             will result in a kernel that does NOT work on a 386.  The
208             kernel will detect this on bootup, and give up.
209           - A kernel with math-emulation compiled in will still use the
210             coprocessor if one is present: the math emulation will just
```

never get used in that case. The kernel will be slightly larger,
but will work on different machines regardless of whether they
have a math coprocessor or not.
- the "kernel hacking" configuration details usually result in a
bigger or slower kernel (or both), and can even make the kernel
less stable by configuring some routines to actively try to
break bad code to find kernel problems (kmalloc()). Thus you
should probably answer 'n' to the questions for
"development", "experimental", or "debugging" features.

COMPILING the kernel:

- Make sure you have at least gcc 3.2 available.
For more information, refer to Documentation/Changes.

Please note that you can still run a.out user programs with this kernel.

- Do a "make" to create a compressed kernel image. It is also
possible to do "make install" if you have lilo installed to suit the
kernel makefiles, but you may want to check your particular lilo setup first.

To do the actual install you have to be root, but none of the normal
build should require that. Don't take the name of root in vain.

- If you configured any of the parts of the kernel as `modules', you
will also have to do "make modules_install".

- Verbose kernel compile/build output:

Normally the kernel build system runs in a fairly quiet mode (but not
totally silent). However, sometimes you or other kernel developers need
to see compile, link, or other commands exactly as they are executed.
For this, use "verbose" build mode. This is done by inserting
"V=1" in the "make" command. E.g.:

```
    make V=1 all
```

To have the build system also tell the reason for the rebuild of each
target, use "V=2". The default is "V=0".

- Keep a backup kernel handy in case something goes wrong. This is
especially true for the development releases, since each new release
contains new code which has not been debugged. Make sure you keep a
backup of the modules corresponding to that kernel, as well. If you
are installing a new kernel with the same version number as your
working kernel, make a backup of your modules directory before you
do a "make modules_install".
Alternatively, before compiling, use the kernel config option
"LOCALVERSION" to append a unique suffix to the regular kernel version.
LOCALVERSION can be set in the "General Setup" menu.

- In order to boot your new kernel, you'll need to copy the kernel
image (e.g. .../linux/arch/i386/boot/bzImage after compilation)

to the place where your regular bootable kernel is found.

- Booting a kernel directly from a floppy without the assistance of a bootloader such as LILO, is no longer supported.

  If you boot Linux from the hard drive, chances are you use LILO which uses the kernel image as specified in the file /etc/lilo.conf. The kernel image file is usually /vmlinuz, /boot/vmlinuz, /bzImage or /boot/bzImage. To use the new kernel, save a copy of the old image and copy the new image over the old one. Then, you MUST RERUN LILO to update the loading map!! If you don't, you won't be able to boot the new kernel image.

  Reinstalling LILO is usually a matter of running /sbin/lilo.
  You may wish to edit /etc/lilo.conf to specify an entry for your old kernel image (say, /vmlinux.old) in case the new one does not work. See the LILO docs for more information.

  After reinstalling LILO, you should be all set. Shutdown the system, reboot, and enjoy!

  If you ever need to change the default root device, video mode, ramdisk size, etc. in the kernel image, use the 'rdev' program (or alternatively the LILO boot options when appropriate). No need to recompile the kernel to change these parameters.

- Reboot with the new kernel and enjoy.

IF SOMETHING GOES WRONG:

- If you have problems that seem to be due to kernel bugs, please check the file MAINTAINERS to see if there is a particular person associated with the part of the kernel that you are having trouble with. If there isn't anyone listed there, then the second best thing is to mail them to me (torvalds@linux-foundation.org), and possibly to any other relevant mailing-list or to the newsgroup.

- In all bug-reports, *please* tell what kernel you are talking about, how to duplicate the problem, and what your setup is (use your common sense). If the problem is new, tell me so, and if the problem is old, please try to tell me when you first noticed it.

- If the bug results in a message like

```
unable to handle kernel paging request at address C0000010
Oops: 0002
EIP:   0010:XXXXXXXX
eax: xxxxxxxx   ebx: xxxxxxxx   ecx: xxxxxxxx   edx: xxxxxxxx
esi: xxxxxxxx   edi: xxxxxxxx   ebp: xxxxxxxx
ds: xxxx  es: xxxx  fs: xxxx  gs: xxxx
Pid: xx, process nr: xx
xx xx xx xx xx xx xx xx xx xx
```

or similar kernel debugging information on your screen or in your system log, please duplicate it *exactly*. The dump may look incomprehensible to you, but it does contain information that may help debugging the problem. The text above the dump is also important: it tells something about why the kernel dumped code (in the above example it's due to a bad kernel pointer). More information on making sense of the dump is in Documentation/oops-tracing.txt

- If you compiled the kernel with CONFIG_KALLSYMS you can send the dump as is, otherwise you will have to use the "ksymoops" program to make sense of the dump (but compiling with CONFIG_KALLSYMS is usually preferred). This utility can be downloaded from ftp://ftp.<country>.kernel.org/pub/linux/utils/kernel/ksymoops/ . Alternately you can do the dump lookup by hand:

- In debugging dumps like the above, it helps enormously if you can look up what the EIP value means. The hex value as such doesn't help me or anybody else very much: it will depend on your particular kernel setup. What you should do is take the hex value from the EIP line (ignore the "0010:"), and look it up in the kernel namelist to see which kernel function contains the offending address.

  To find out the kernel function name, you'll need to find the system binary associated with the kernel that exhibited the symptom. This is the file 'linux/vmlinux'. To extract the namelist and match it against the EIP from the kernel crash, do:

```
nm vmlinux | sort | less
```

  This will give you a list of kernel addresses sorted in ascending order, from which it is simple to find the function that contains the offending address. Note that the address given by the kernel debugging messages will not necessarily match exactly with the function addresses (in fact, that is very unlikely), so you can't just 'grep' the list: the list will, however, give you the starting point of each kernel function, so by looking for the function that has a starting address lower than the one you are searching for but is followed by a function with a higher address you will find the one you want. In fact, it may be a good idea to include a bit of "context" in your problem report, giving a few lines around the interesting one.

  If you for some reason cannot do the above (you have a pre-compiled kernel image or similar), telling me as much about your setup as possible will help. Please read the REPORTING-BUGS document for details.

- Alternately, you can use gdb on a running kernel. (read-only; i.e. you cannot change values or set break points.) To do this, first compile the kernel with -g; edit arch/i386/Makefile appropriately, then do a "make clean". You'll also need to enable CONFIG_PROC_FS (via "make config").

  After you've rebooted with the new kernel, do "gdb vmlinux /proc/kcore". You can now use all the usual gdb commands. The command to look up the

point where your system crashed is "l *0xXXXXXXXX". (Replace the XXXes with the EIP value.)

gdb'ing a non-running kernel currently fails because gdb (wrongly) disregards the starting offset for which the kernel is compiled.