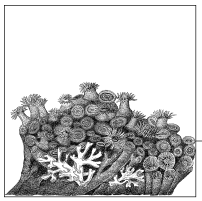

6
Upgrading a Kernel

Inevitably it happens: you have a custom-built kernel, working just wonderfully except for one little thing that you know is fixed in the latest release from the kernel developers. Or a security problem is found, and a new stable kernel release is made public. Either way, you are faced with the issue of upgrading the kernel and you do not want to lose all the time and effort that went into making that perfect kernel configuration.

This chapter is going to show how easy it is to update a kernel from an older versions, while still retaining all of the configuration options from the previous one.

First off, please back up the *.config* file in the kernel source directory. You have spent some time and effort into creating it, and it should be saved in case something goes wrong when trying to upgrade.

```
$ cd ~/linux/linux-2.6.17.11
$ cp .config ../good_config
```

Only five simple steps are needed to upgrade a kernel from a previously built one:

1. Get the new source code.
2. Apply the changes to the old source tree to bring it up to the newer level.
3. Reconfigure the kernel based on the previous kernel configuration.
4. Build the new kernel.
5. Install the new kernel.

The last two steps work the same as described before, so we will only discuss the first three steps in this chapter.

In this chapter, we are going to assume that you have built a successful 2.6.17.9 kernel release, and want to upgrade to the 2.6.17.11 release.

Download the New Source

The Linux kernel developers realize that users do not wish to download the entire source code to the kernel for every update. That would be a waste of bandwidth and time. Because of this, they offer a patch that can upgrade an older kernel release to a newer one.*

On the main *kernel.org* web site, you will remember that it contained a list of the current kernel versions that are available for download, as shown in Figure 6-1.

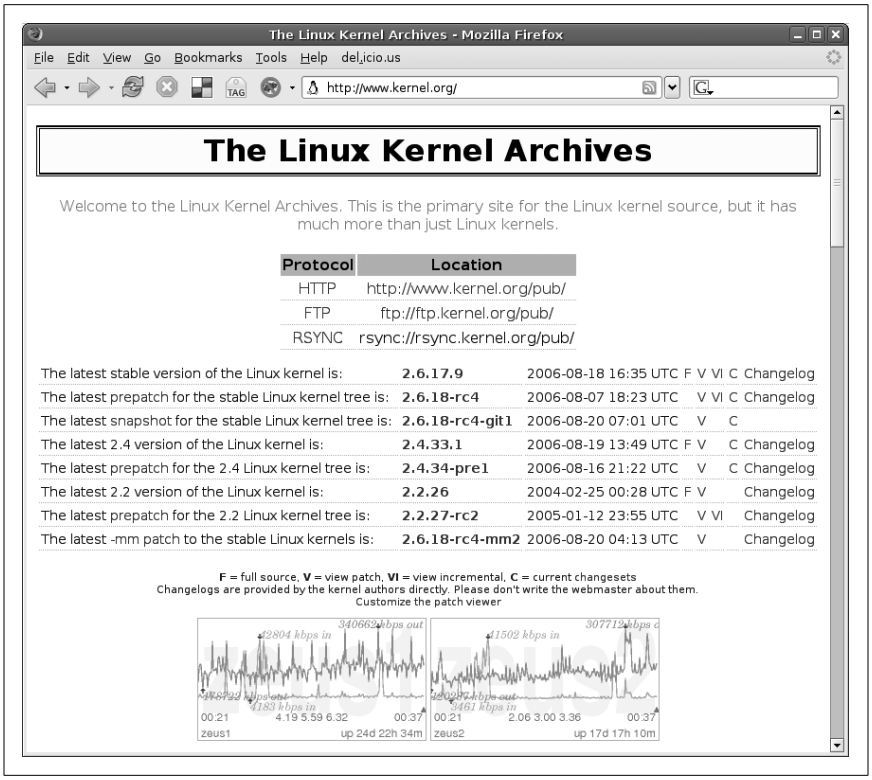

Figure 6-1. The main kernel.org web site

Previously, you used the link pointed to you by the `F` to download the entire source code for the kernel. However, if you click on the name of the kernel release, it will download a patch file instead, as shown in Figure 6-2.

* It is called *patch* because the program *patch* takes the file and applies it to the original tree, creating the new tree. The patch file contains a representation of the changes that are necessary to reconstruct the new files, based on the old ones. Patch files are readable, and contain a list of the lines that are to be removed and the lines that are to be added, with some context within the file showing where the changes should be made.

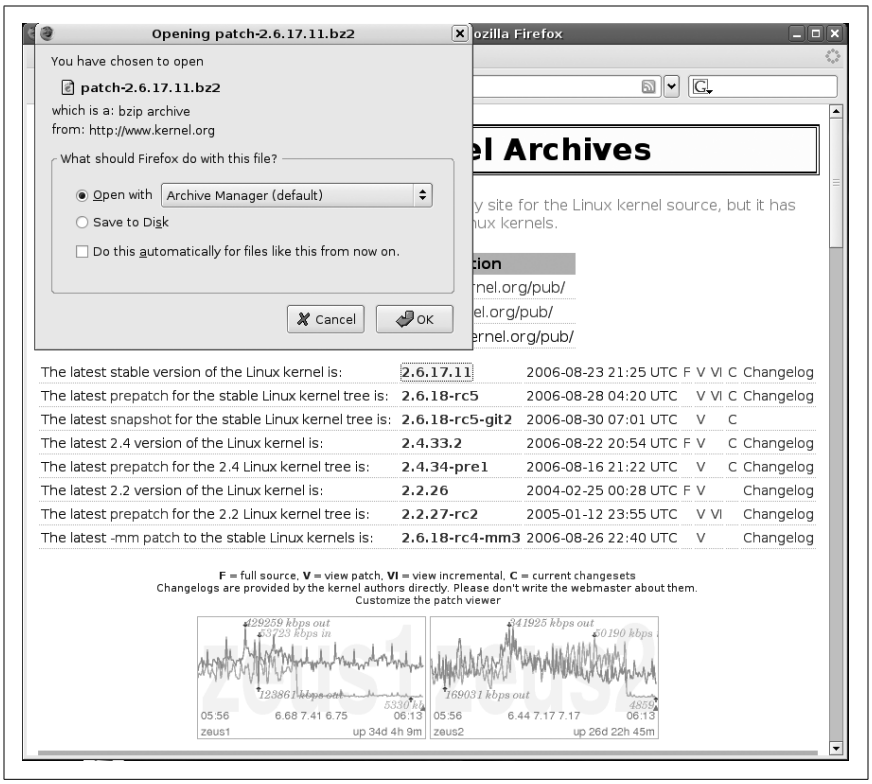

Figure 6-2. Downloading a patch from kernel.org

This is what we want to do when upgrading. But we need to figure out what patch to download.

Which Patch Applies to Which Release?

A kernel patch file will upgrade the source code from only one specific release to another specific release. Here is how the different patch files can be applied:

- Stable kernel patches apply to the base kernel version. This means that the 2.6.17.10 patch will only apply to the 2.6.17 kernel release. The 2.6.17.10 kernel patch will not apply to the 2.6.17.9 kernel or any other release.
- Base kernel release patches only apply to the previous base kernel version. This means that the 2.6.18 patch will only apply to the 2.6.17 kernel release. It will not apply to the last 2.6.17.y kernel release, or any other release.
- Incremental patches upgrade from a specific release to the next release. This allows developers to not have to downgrade their kernel and then upgrade it, just to switch from the latest stable release to the next stable release (remember that the stable release patches are only against the base kernel, not the previous stable release). Whenever possible, it is recommended that you use the incremental patches to make your life easier.

Finding the Patch

As we want to go from the 2.6.17.9 kernel release, to the 2.6.17.11 release, we will need to download two different patches. We will need a patch from the 2.6.17.9 release to the 2.6.17.10 release, and then from the 2.6.17.10 release to the 2.6.17.11 release.*

The stable and base kernel patches are located in the same directory structure as the main source trees. All incremental patches can be found one level lower, in the *incr* subdirectory. So, to find the patch that goes from 2.6.17.9 to 2.6.17.10, we look in the */pub/linux/kernel/v2.6/incr* directory to find the files we need:†

```
$ cd ~/linux
$ lftp ftp.kernel.org/pub/linux/kernel/v2.6/incr
cd ok, cwd=/pub/linux/kernel/v2.6/incr
lftp ftp.kernel.org:/pub/linux/kernel/v2.6/incr> ls *2.6.17.9*.bz2
-rw-rw-r--    1 536      536          2872 Aug 22 19:23 patch-2.6.17.9-10.
bz2
lftp ftp.kernel.org:/pub/linux/kernel/v2.6/incr> get patch-2.6.17.9-10.bz2
2872 bytes transferred
lftp ftp.kernel.org:/pub/linux/kernel/v2.6/incr> get patch-2.6.17.10-11.bz2
7901 bytes transferred
lftp ftp.kernel.org:/pub/linux/kernel/v2.6/incr> exit
$ ls -F
good_config linux-2.6.17.9/ patch-2.6.17.10-11.bz2  patch-2.6.17.9-10.bz2
```

Applying the Patch

As the patches we have downloaded are compressed, the first thing to do is uncompress them with the *bzip2* command:

```
$ bzip2 -dv patch-2.6.17.9-10.bz2
  patch-2.6.17.9-10.bz2: done
$ bzip2 -dv patch-2.6.17.10-11.bz2
  patch-2.6.17.10-11.bz2: done
$ ls -F
good_config linux-2.6.17.9/  patch-2.6.17.10-11  patch-2.6.17.9-10
```

Now we need to apply the patch files to the kernel directory. Go into the directory:

```
$ cd linux-2.6.17.9
```

Now run the *patch* program to apply the first patch moving the source tree from the 2.6.17.9 to the 2.6.17.10 release:

```
$ patch -p1 < ../patch-2.6.17.9-10
```

* If you need to upgrade more than two versions, it is recommended as a way to save steps, to go backward and then upgrade forward. In this case, we could go backward from 2.6.17.9 to 2.6.17 and then forward from 2.6.17 to 2.6.17.11.

† In this example, we use the very good *lftp* FTP program to download the patch files. Any FTP program or a web browser can be used to download the same files. The important thing here is to show where the files are located.

```
patching file Makefile
patching file block/elevator.c
patching file fs/udf/super.c
patching file fs/udf/truncate.c
patching file include/net/sctp/sctp.h
patching file include/net/sctp/sm.h
patching file net/sctp/sm_make_chunk.c
patching file net/sctp/sm_statefuns.c
patching file net/sctp/socket.c
```

Verify that the patch really did work properly and that there are no errors or warnings in the output of the patch program. It is also a good idea to look at the *Makefile* of the kernel to see the kernel version:

```
$ head -n 5 Makefile
VERSION = 2
PATCHLEVEL = 6
SUBLEVEL = 17
EXTRAVERSION = .10
NAME=Crazed Snow-Weasel
```

Now that the kernel is at the 2.6.17.10 release level, do the same thing as before, and apply the patch to bring it up to the 2.6.17.11 level:

```
$ patch -p1 < ../patch-2.6.17.10-11
patching file Makefile
patching file arch/ia64/kernel/sys_ia64.c
patching file arch/sparc/kernel/sys_sparc.c
patching file arch/sparc64/kernel/sys_sparc.c
patching file drivers/char/tpm/tpm_tis.c
patching file drivers/ieee1394/ohci1394.c
patching file drivers/md/dm-mpath.c
patching file drivers/md/raid1.c
patching file drivers/net/sky2.c
patching file drivers/pci/quirks.c
patching file drivers/serial/Kconfig
patching file fs/befs/linuxvfs.c
patching file fs/ext3/super.c
patching file include/asm-generic/mman.h
patching file include/asm-ia64/mman.h
patching file include/asm-sparc/mman.h
patching file include/asm-sparc64/mman.h
patching file kernel/timer.c
patching file lib/spinlock_debug.c
patching file mm/mmap.c
patching file mm/swapfile.c
patching file net/bridge/netfilter/ebt_ulog.c
patching file net/core/dst.c
patching file net/core/rtnetlink.c
patching file net/ipv4/fib_semantics.c
patching file net/ipv4/netfilter/arp_tables.c
patching file net/ipv4/netfilter/ip_tables.c
patching file net/ipv4/netfilter/ipt_ULOG.c
patching file net/ipv4/route.c
patching file net/ipx/af_ipx.c
patching file net/netfilter/nfnetlink_log.c
```

Again verify that the output of the patch program did not show any errors and look at the *Makefile*:

```
$ head -n 5 Makefile
VERSION = 2
PATCHLEVEL = 6
SUBLEVEL = 17
EXTRAVERSION = .11
NAME=Crazed Snow-Weasel
```

Now that the source code is successfully updated to the version you wish to use, it is a good idea to go back and change the directory name to refer to the kernel version number to avoid confusion at a later time:

```
$ cd ..
$ mv linux-2.6.17.9 linux-2.6.17.11
$ ls -F
good_config linux-2.6.17.11/  patch-2.6.17.10-11  patch-2.6.17.9-10
```

Reconfigure the Kernel

Previously, we used the *make menuconfig* or *gconfig* or *xconfig* method to change different configuration options. But once you have a working configuration, the only thing that is necessary is to update it with any new options that have been added to the kernel since the last release. To do this, the *make oldconfig* and *make silentoldconfig* options should be used.

make oldconfig takes the current kernel configuration in the *.config* file, and updates it based on the new kernel release. To do this, it prints out all configuration questions, and provides an answer for them if the option is already handled in the configuration file. If there is a new option, the program stops and asks the user what the new configuration value should be set to. After answering the prompt, the program continues on until the whole kernel configuration is finished.

make silentoldconfig works exactly the same way as *oldconfig*, but it does not print anything to the screen, unless it needs to ask a question about a new configuration option.

Usually, when upgrading between different versions of the stable releases, no new configuration options are added, as this is supposed to be a stable kernel series. If this happens, there are no new questions that need to be answered for the kernel configuration, so the program continues successfully without any need for user intervention. An example of this is moving from the 2.6.17.9 to 2.6.17.11 release:

```
$ cd linux-2.6.17.11
$ make silentoldconfig
scripts/kconfig/conf -s arch/i386/Kconfig
#
# using defaults found in .config
#
```

The following example shows what happens when a new kernel option shows up in a new release. The kernel option to enable *Mutex debugging* is a new one for certain kernel releases. Here is the output when this happened:

```
$ make silentoldconfig
scripts/kconfig/conf -s arch/i386/Kconfig
#
# using defaults found in .config
#
*
* Restart config...
*
*
* Kernel hacking
*
Show timing information on printks (PRINTK_TIME) [Y/n/?] y
Magic SysRq key (MAGIC_SYSRQ) [Y/n/?] y
Kernel debugging (DEBUG_KERNEL) [Y/n/?] y
  Kernel log buffer size (16 => 64KB, 17 => 128KB) (LOG_BUF_SHIFT) [16] 16
  Detect Soft Lockups (DETECT_SOFTLOCKUP) [Y/n/?] y
  Collect scheduler statistics (SCHEDSTATS) [N/y/?] n
  Debug slab memory allocations (DEBUG_SLAB) [Y/n/?] y
    Memory leak debugging (DEBUG_SLAB_LEAK) [Y/n] y
  Mutex debugging, deadlock detection (DEBUG_MUTEXES) [N/y/?] (NEW) y
```

The configuration program stops at this option and asks for the user to choose an option. Press y, and the program continues on:

```
  Spinlock debugging (DEBUG_SPINLOCK) [Y/n/?] y
  Sleep-inside-spinlock checking (DEBUG_SPINLOCK_SLEEP) [Y/n/?] y
  kobject debugging (DEBUG_KOBJECT) [N/y/?] n
  Highmem debugging (DEBUG_HIGHMEM) [N/y/?] n
  Compile the kernel with debug info (DEBUG_INFO) [N/y/?] n
Debug Filesystem (DEBUG_FS) [Y/?] y
Debug VM (DEBUG_VM) [N/y/?] n
Compile the kernel with frame pointers (FRAME_POINTER) [N/y/?] n
Compile the kernel with frame unwind information (UNWIND_INFO) [N/y/?] n
Force gcc to inline functions marked 'inline' (FORCED_INLINING) [N/y/?] n
torture tests for RCU (RCU_TORTURE_TEST) [N/m/y/?] n
Check for stack overflows (DEBUG_STACKOVERFLOW) [N/y/?] n
Stack utilization instrumentation (DEBUG_STACK_USAGE) [N/y/?] n
Stack backtraces per line (STACK_BACKTRACE_COLS) [2] 2
*
* Page alloc debug is incompatible with Software Suspend on i386
*
Write protect kernel read-only data structures (DEBUG_RODATA) [N/y/?] n
Use 4Kb for kernel stacks instead of 8Kb (4KSTACKS) [N/y/?] n
```

So upgrading the kernel configuration for a new release is as simple as using a different configuration option to *make*. With this method, you do not need to use the graphical or text-oriented configuration programs for any new kernel update.

Can't This Be Automated?

The whole process of downloading the proper patch file, uncompressing it, and then applying it seems to be ripe for automating. Kernel developers being the type that like to automate repetitive tasks, the program *ketchup* has been created to handle all of this automatically. See Appendix A for more details on how this program works and how to use it.