What to do with the Linux kernel

KERNEL HACKS

If you get right down to it, the Linux kernel is the real Linux. This month we focus on tools for tuning and tailoring the kernel.

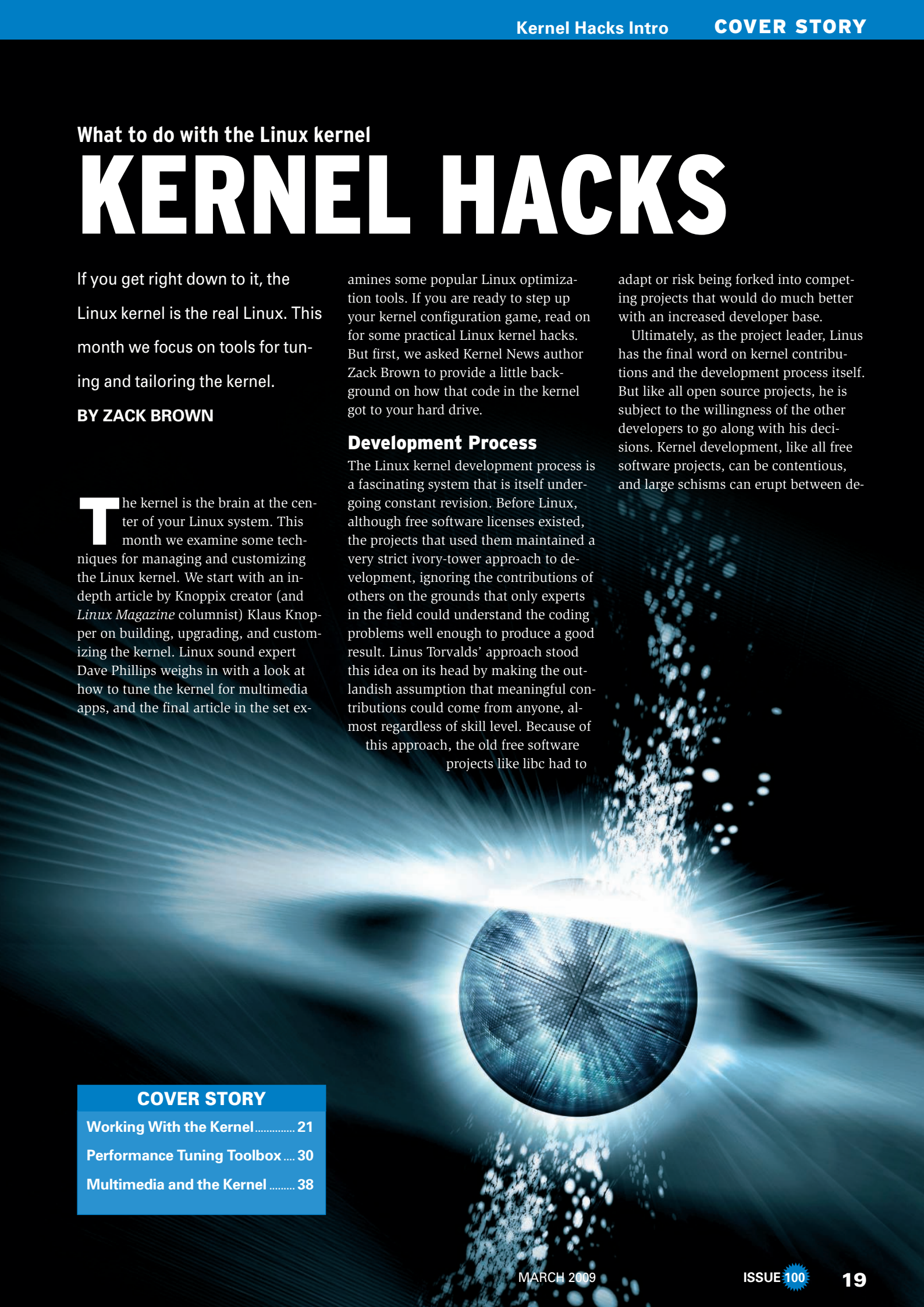
BY ZACK BROWN

The kernel is the brain at the center of your Linux system. This month we examine some techniques for managing and customizing the Linux kernel. We start with an in-depth article by Knoppix creator (and *Linux Magazine* columnist) Klaus Knopper on building, upgrading, and customizing the kernel. Linux sound expert Dave Phillips weighs in with a look at how to tune the kernel for multimedia apps, and the final article in the set examines some popular Linux optimization tools. If you are ready to step up your kernel configuration game, read on for some practical Linux kernel hacks. But first, we asked Kernel News author Zack Brown to provide a little background on how that code in the kernel got to your hard drive.

Development Process

The Linux kernel development process is a fascinating system that is itself undergoing constant revision. Before Linux, although free software licenses existed, the projects that used them maintained a very strict ivory-tower approach to development, ignoring the contributions of others on the grounds that only experts in the field could understand the coding problems well enough to produce a good result. Linus Torvalds' approach stood this idea on its head by making the outlandish assumption that meaningful contributions could come from anyone, almost regardless of skill level. Because of this approach, the old free software projects like libc had to adapt or risk being forked into competing projects that would do much better with an increased developer base.

Ultimately, as the project leader, Linus has the final word on kernel contributions and the development process itself. But like all open source projects, he is subject to the willingness of the other developers to go along with his decisions. Kernel development, like all free software projects, can be contentious, and large schisms can erupt between de-



COVER STORY

velopers with different ideas about how to do things. Some open source projects can get so divisive that one developer forks the entire code base and continues development with whichever other contributors care to follow.

How does kernel development take place today? To begin with, the development community supports several mailing lists, the primary one being linux-kernel, which you can read about at *http://www.tux.org/lkml*. Every release of the kernel also includes a *MAINTAINERS* file in the documentation directory, which lists everyone officially responsible for a portion of the kernel, along with the mailing lists relevant to that portion. Each mailing list is (or should be) a place you can post to without subscribing. The people who reply will CC you in their responses. This arrangement is all part of the original idea of encouraging contributions from everyone. You don't have to be deeply involved in kernel development or any particular area of kernel development, in order to make a contribution. All you need is the source code and a desire to help.

The mailing lists are the primary means of communication between kernel developers. Programmers make proposals, submit and discuss patches, debate controversies, and announce new releases of various projects, including the kernel itself. But the *MAINTAINERS* file is only the start of the process. The developers maintain several kernel forks, each designed with the ultimate goal of feeding their differences back into the primary kernel. With the recent arrival of the of the Git code management system (developed by Linus and a number of other kernel developers), it is now possible for groups of developers to develop their own common fork, before feeding the changes upstream to Linus.

Signing Off

Depending on the part of the kernel you're working on, and the maintainer you contact, that maintainer will sign off on your patch and submit it to one of the public trees. Once there, the patch will be tested by a wider audience. When you first post your patch, the only people who actually test it are the people who actually take the trouble to apply the patch by hand to one of their kernel trees. Once in one of the public trees, the patch will be at least marginally tested by anyone who downloads and tries out that tree. Some trees are more popular than others. Andrew Morton's *-mm* tree is actually the kernel of choice for various developers, who run it regularly on their home systems.

Several years ago, the community developed the concept of kernel *lieutenants* – an inner circle of experienced developers who Linus trusts to send only good, solid patches. The idea of a lieutenant is more of a useful idea than any kind of formal arrangement. Some folks who do consistently good work tend to be able to get their submissions in faster, and they tend to be the people Linus would prefer the various maintainers route certain patches through. These essentially social relationships between Linus and the lieutenants further reduce the bottleneck of Linus having to review every single patch. Although these lieutenants are a legitimate part of kernel development process, the few times anyone has posted to the mailing list asking for a list of Linus' lieutenants, the request was not taken very seriously, because no such formal designation exists. It's just a small group of people who seem to work well with the patches.

Maintainers

If your patch is more than just a fix – for instance, if it is a new driver or some other discrete part of the kernel, you might also consider submitting a patch to the *MAINTAINERS* file and adding yourself as the official maintainer. No one has the official task of identifying maintainers and adding them to the list; it is up to the contributors themselves to take responsibility for their portion of the code. In some cases, the question of maintainership is obvious. If you wrote the driver and plan to maintainer it, go ahead and list yourself as the maintainer. If someone has abandoned maintainership of something, or if you suspect they have, and you want to maintain the project in their place, the best thing to do is to ask them, or ask on the mailing list. Someone with authority over that part of the kernel will probably get right back to you. In some cases, maintainership is handed over directly via public announcement after the current maintainer has completed a private search for a replacement.

The maintainer's obligation is fairly fluid. A maintainer might do most of the coding personally or simply act as a custodian of the code, primarily accepting patches by other people. About the only requirement is that a maintainer should always keep their entry in the *MAINTAINERS* file up to date and respond to emails, even if it's to say that they don't do much work on the project anymore and would like to find a replacement.

Conclusion

The kernel development process has its own culture and its own sense of etiquette. It is a culture that favors encouraging others, as well as rewarding people who contribute and accept feedback on their contributions. As the process continues to evolve, one overarching theme is: How can the project make the best possible use of everyone's desire to help? ■