ASK KLAUS!

Klaus Knopper is the creator of Knoppix and co-founder of the LinuxTag expo. He currently works as a teacher, programmer, and consultant. If you have a configuration problem, or if you just want to learn more about how Linux works, send your questions to:

klaus@linux-magazine.com.

No DVD Boot

I have an "old" 1 GHz PC with a DVD reader and CD writer, 512MB Ram, and 2 * 80MB HD. Unfortunately, apparently the BIOS is unable to boot from the DVD reader.

So, my question: every month I receive a very valuable Linux DVD from this magazine. How can I convert the DVD into multiple CD's from which I can boot?

I have been thinking about this for quite a while now, especially since there are old computers around that cannot even boot from an existing CD-Rom drive directly.

The problem is really that, when booting from CD or DVD, there is no operating system running at this point, and all steps necessary in order to load a kernel and initial ramdisk from the CD or DVD have to be done by the computer's BIOS. The way this works is called "el torito" for CDs or DVDs, and there are a few different modes that are possible. The way most older BIOSes boot from CD is by virtually switching the CD-ROM drive with the floppy drive and loading a floppy image from CD.

Since this floppy image is only allowed in certain sizes, it is not easy to fit a kernel, an initial ramdisk, plus all boot scripts and programs in it. Knoppix 3.x was the last version with a kernel small enough to fit on a single floppy disk. Now we have to use the so-called "no emulation mode," where a bootloader (*isolinux*) is loading kernel and initrd directly from CD. This boot mode is not supported by all old computers, so this may apply to your "old" machine as well. By the way. my oldest testing machine for testing Knoppix is a 486 with 100MHz, 28MB of RAM, and I have to boot this with tricks similar to the tricks I am about to describe.

If you try to boot from a floppy disk first, in order to access the CD, and continue booting from there later, you need to include on the floppy all drivers required for all types of CD or DVD-ROM (possibly including SCSI, USB, Firewire or SATA). It's tricky.

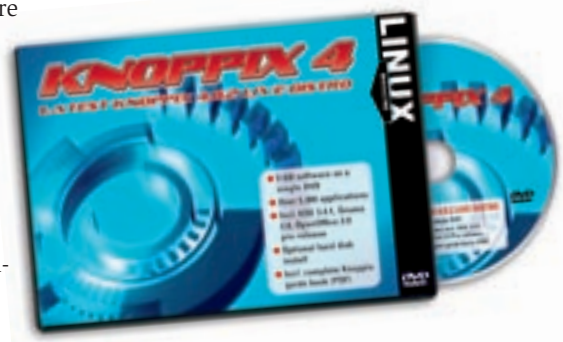
Your idea of just "splitting" the DVD into several CDs is not impossible, but it is hard to do. Using *Knoppix/unionfs*, several CDs could be "joined" into one large file system. But you would need one drive for each CD, and you would need to modify the boot scripts to check each drive for the presence of a unionfs component.

The best/easiest way in your setup would be, for now, to make a bootable CD just with the kernel and the initial ramdisk, and (to be safe) immediately REMOVE this "boot-only" CD from your bootable CD-Drive as soon as the kernel and initrd have been loaded. Then, at least for the Knoppix case, all drives, including the DVD, will be searched for the necessary files to continue starting up the system.

These are the steps to create a bootable CD containing only the necessary boot files from Knoppix (your Knoppix-DVD is mounted at */mnt/dvd* in this example):

```
mkdir /tmp/bootcd
cp -a /mnt/dvd/boot /tmp/bootcd/
chmod -R u+w /tmp/bootcd/
mkisofs -input-charset ➥
ISO-8859-1 -pad -l -r -J ➥
-no-emul-boot -boot-load-size ➥
4 -boot-info-table -b ➥
boot/isolinux/isolinux.bin -c ➥
boot/isolinux/boot.cat ➥
-hide-rr-moved -o ➥
/tmp/bootcd.iso /tmp/bootcd
```

Then you just burn */tmp/bootcd.iso* onto an empty CD-R using your favorite



burning program that has direct iso-burning support (such as cdrecord). Don't be surprised: this boot CD image is really only about 4-16MB in size.

Since most live CDs have to recheck all drives after loading kernel and initrd, this trick will probably work for a lot of different Live DVDs. You could even make a multiboot CD image that allows you to boot several DVDs using the same initial boot CD.

Boot Options

There are a very large number of boot options in Linux. I assume some are used very often and some are very rarely used. What are the most important boot options to know about, and how are they used? What would be on your "top ten list" of essential boot parameters? Do you have any favorites that you find very useful but most people don't know about?

First, we have to distinguish between kernel options, module parameters, and "unofficial" options that are not used anywhere unless a distribution evaluates them inside scripts in the later boot process (like Knoppix does). Everything you type (or list in LILOs or an isolinux "append" parameter) after the name of the kernel image will appear in clear text inside the file */proc/cmdline*.

Genuine kernel options are set in the static part of the kernel, for example: *noapic* or *pci=bios*. The first example switches off the "advanced interrupt controller," which is a workaround to some problems frequently observed with interrupt routing (or buggy hardware). The second example tells the kernel to stick to the interrupt distribution as set in the BIOS, rather than trying a direct (sometimes "best guess") approach. This can solve problems at a very early stage of the boot process. Unfortunately, there is no way to "undo" these parameters with a followup option, so putting them into the "append" option line in *lilo.conf*, *syslinux.cfg* or *isolinux.cfg* is only recommended if you are sure you won't need *apic* or *pci=direct* when booting on a different computer with the same bootdisk. And there are boards that insist on having *apic* or *acpi* support, otherwise they won't let Linux come up at all.

A quite large but not complete list of kernel options can be found in the documentation that comes with each source release of the linux kernel, in the file *Documentation/kernel-parameters.txt*.

Module parameters, the second type of options, are parameters passed on to specific kernel modules that are compiled into the kernel. These module parameters sometimes contain the module name in the first part, and the options (or a list thereof) following as a (comma-separated, if there are more than one) list, like this:

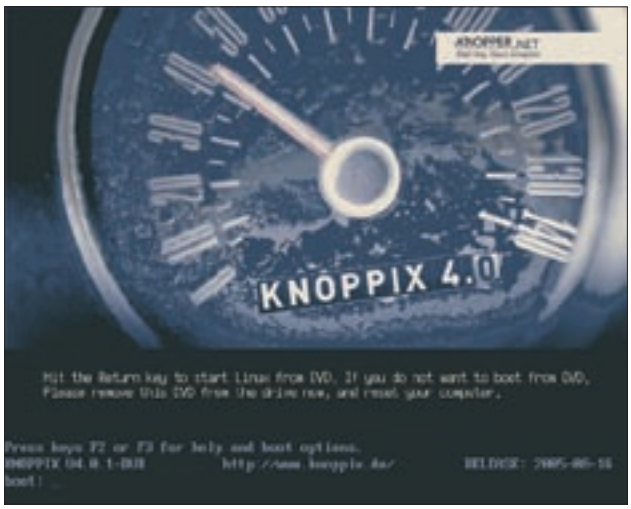


Figure 1: The Knoppix 4.01 boot screen.

```
psmouse.psmouse_proto=imps
```

This example would tell the psmouse module (ps/2 mouse port, sometimes referred to as */dev/psaux*), which, in this example, is compiled directly into the kernel, to use the scrollwheel-enabled imps-protocol rather than the autodetection or plain ps/2 protocol. This is sometimes necessary for notebooks when you cannot get parts of a touchpad or buttons to work correctly (frequently used for Synaptics touchpads that don't respond well to the original Synaptics driver), or for cases when the mouse seems to jump over the screen for no apparent reason.

The third type of option relates to arbitrary, space-separated texts that are just ignored by the kernel, and by the kernel modules. If these options have the form *name=value*, you can evaluate them directly as shell variables inside */linuxrc* or an *initrd* (initial ramdisk), which is very practical for configuration scripts that don't implement their own enhanced */proc/cmdline* parser. Knoppix uses this for activating DMA when the user types *knoppix dma* at the boot command line. *dma* is not a (current) kernel option, but it is used for a shell conditional inside the */linuxrc* hardware detection part.

Here is my short list of "favorite" kernel parameters:

- *acpi=off* – Disable a (sometimes defective) "advanced configuration and power[management] interface." Some computers won't boot without ACPI turned off, whereas some others really do need ACPI in order to distribute interrupts to devices correctly. If ACPI works well for your computer, you should probably use it in favor of the older APM, since ACPI allows you to control things like fan cooling and suspend states better than the more simple APM. If you know that your computer needs ACPI, but your kernel switches it off for some reason even though you are sure you have compiled it in, try *acpi=force* instead.
- *noapic* – Frequently confused with ACPI (see above), "Advanced Programmable Interrupt Controller," which is partly inside some CPUs and partly (especially for handling I/O) a chip on the computer mainboard. The *noapic* option allows you to have more than the very limited and preoccupied 15 available interrupts in order to handle a large number of different devices, but it sometimes fails to work (for example, some components like network cards seem to be "dead," in spite of their being detected successfully). noapic stands for "no APIC." (Strange that they sometimes say "… off," and sometimes "no…," isn't it?) noapic turns this "advanced" part of the chipset off and reverts to the old but safe method. This means you are more likely to get interrupt conflicts when using noapic, but some devices that are not accessible with apic will suddenly work when they seemed to be broken before.

"...a combination of acpi=off noapic pci=bios"

- *pci = bios* – By default, Linux tries to associate system interrupts with hardware components in a "direct" approach based on a kind of "best guess," without trying to check settings in the BIOS configuration if not absolutely necessary. "pci = bios" tells the kernel to use the user-defined BIOS-settings for interrupt distribution instead of its own way. Sometimes you will need this when a card is given an interrupt that is already taken by another device that Linux did not detect before.
- *pnpbios = off* You remember "Plug and Play" cards? This was probably the first attempt to unburden the user from configuring drivers manually in order to assign interrupts to different devices (mostly ISA-type cards), and instead reading a range of allowed interrupt/I/O address combinations from a card and doing a semi-automatic configuration.It frequently didn't work, and instead, "plug and play" first tried the user's patience until he/she gave up, unplugged the card, and bought a new one. Anyway, the Plug and Play BIOS extension that is still present in some older boards can cause problems with wrongly allocated interrupts. Switching "plug and play" off disables automatic configuration of legacy cards, and, unless you still have a card that really requires p&p (soundcards of this type were popular at one time), shutting off the Plug and Play BIOS can solve another possible category of problems.

Most of these problems with interrupts, I/O addresses, buses, etc. could have been avoided in the first place if PC hardware hadn't started life with a quick and dirty, if not broken, design, that is just dynamically extended nowadays, but not replaced.

The combination *acpi = off noapic pci = bios pnpbios = off* solves the majority of cases (for me), when Linux (especially full-featured kernels designed for live CDs) hangs during boot for no apparent reason.

Actually, there is another type of boot option that isn't really a boot option. It is used by the specific bootloader to find and relocate an initial ramdisk, change the verbosity level, or choose the location of the kernel image. Sometimes these options (like *initrd = ...*, which tells the bootloader where to read the initial ramdisk BEFORE the kernel is started) are also passed through as if they were real kernel options.

Sata Problems

I have just purchased a HP Pavilion a1250n computer with a SATA hard drive. I have been using Linux full-time since 1997, and I am not considered a beginner by any means. However, I have come across a problem that I cannot seem to resolve.

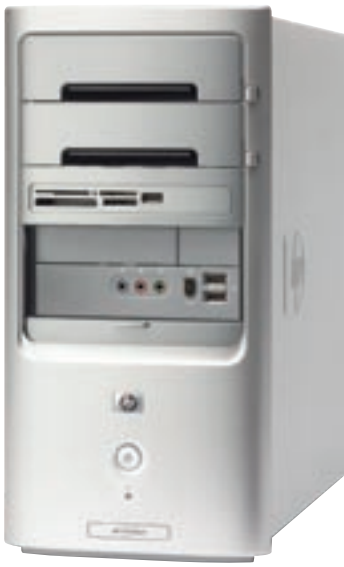
Linux does not seem to recognize the sata hard drive that is in this HP Pavilion. It hangs just after the boot sequence shown in Listing 1.

At this point, the system will not load the hard drive controller. I have tried using other SATA drivers but none will work. However, I can boot Linux in safe mode. Any help would be greatly appreciated since HP can't help me. I want my Linux back.

For a definitive answer to that specific controller problem, I would have to do some more research, but I can give you some general hints about controller/SATA problems.

First, I would see if anything changes when using my "favorite" options from the previous question, or using certain combinations of those options, such as (*acpi = off noapic pnpbios = off pci = bios*), since they usually do not cause any performance loss.

There have been reports that some SIL controllers also start working with the *irqpoll* kernel option, though this solution is not really recommended since it can cause side effects for other devices and slow down the system.



Some combinations of *acpi/apic/bios* settings also seem to influence the way the SATA controller works, and the full specifications may not have been reverse engineered yet in order to fix the driver itself. Thinking about it, you should really try booting with *irqpoll* first.

The best recommendation for non-working SATA controllers (and you may not like this recommendation...) is to wait for the next kernel release, which may provide support for new controllers or fixes for broken ones.

In your case, if the SATA controller works in "safe mode" (by the way, what is "safe mode", a special boot option in your distribution?), you may want to check which kernel is running in that mode, and which options have been enabled. Try the command *uname -a ; cat /proc/cmdline*, and maybe you will find a quick solution. ■

Send your Linux questions to klaus@linux-magazine.com.

Listing 1: Boot Sequence Before the Problem

```
01 Lading sata_sil
02 Ata1: SATA max udma/100 cmd 0xF880E080 ct1 0xF88oeo8a bmdma
   oxF880E000 irq 11
03 Ata2: SATA max udma/100 cmd )xF880Eoco ct1 0xF88oEoca bmda
   0xF880E008 irq 11
04 Input: 1mEX ps/2 Logitecch explorer mouse on isa0060/serial01
05 Ata1: Dev o ata, max udma/100 488397168 sectors: lba48
```