Booting

Ok, now that you've gotten Slackware installed on your system, you should learn exactly what controls the boot sequence of your machine, and how to fix it should you manage to break it somehow. If you use Linux long enough, sooner or later you will make a mistake that breaks your bootloader. Fortunately, this doesn't require a reinstall to fix. Unlike many other operating systems that hide the underlying details of how they work, Linux (and in particular, Slackware) gives you full control over the boot process. Simply by editing a configuration file or two and re-running the bootloader installer, you can quickly and easily change (or break) your system. Slackware even makes it easy to dual-boot multiple operating systems, such as other Linux distributions or Microsoft Windows.

mkinitrd

Before we go any further, a quick discussion on the Linux kernel is warranted. Slackware Linux includes at least two, but sometimes more, different kernels. While they are all compiled from the same source code, and hence are the *"same"*, they are not identical. Depending on your architecture and Slackware version, the installer may have loaded your system with several kernels. There are kernels for single-processor systems and kernels for multi-processor systems (on 32bit Slackware). In the old days, there were lots of kernels for installing on many different kinds of hard drive controllers. More importantly for our discussion, there are *"huge"* kernels and *"generic"* kernels.

If you look inside your `/boot` directory, you'll see the various kernels installed on your system.

```
darkstar:~# ls -1 /boot/vmlinuz*
/boot/vmlinuz-huge-2.6.29.4
/boot/vmlinuz-generic-2.6.29.4
```

Here you can see that I have two kernels installed, `vmlinuz-huge-2.6.29.4` and `vmlinuz-generic-2.6.29.4`. Each Slackware release includes different kernel versions and sometimes even slightly different names, so don't be alarmed if what you see doesn't exactly match what I have listed here.

Huge kernels are exactly what you might think; they're huge. However, that does NOT mean that they have all of the possible drivers and such compiled into them. Instead, these kernels are made to boot (and run) on every conceivable computer on which Slackware is supported (there may very well be a few out there that won't boot/work with them though). They most certainly contain support for hardware your machine does not (and never will) have, but that shouldn't concern you. These kernels are included for several reasons, but probably the most important is their use by Slackware's installer - these are the kernels that the Slackware installation disks run. If you chose to let the installer configure your bootloader for you, it chooses to use these kernels due to the incredible variety of hardware they support. In contrast, the generic kernels support very little hardware without the use of external modules. If you want to use one of the generic kernels, you'll need to make use of something called an initrd, which is created using the ***mkinitrd***(8) utility.

So why should you use a generic kernel? Currently the Slackware development team recommends use of a generic kernel for a variety of reasons. Perhaps the most obvious is size. The huge kernels are currently about twice the size of the generic kernels before they are uncompressed and loaded

into memory. If you are running an older machine, or one with some small ammount of RAM, you will appreciate the savings the generic kernels offer you. Other reasons are somewhat more difficult to quantify. Conflicts between drivers included in the huge kernels do appear from time to time, and generally speaking, the huge kernels may not perform as well as the generic ones. Also, by using the generic kernels, special arguments can be passed to hardware drivers seperately, rather than requiring these options be passed on the kernel command line. Some of the tools included with Slackware work better if your kernel uses some drivers as modules rather than statically building them into the kernel. If you're having trouble understanding this, don't be alarmed: just think *"huge kernel = good, generic kernel = better"*.

Unfortunately, using the generic kernels isn't as straightforward as using the huge kernels. In order for the generic kernel to boot your system, you must also include a few basic modules in an initird. Modules are pieces of compiled kernel code that can be inserted or removed from a running kernel (ideally using ***modprobe***(8). This makes the system somewhat more flexible at the cost of a tiny bit of added complexity. You might find it easier to think of modules as device drivers, at least for this section. Typically you will need to add the module for whatever filesystem you chose to use for your root partition during the installer, and if your root partition is located on a SCSI disk or RAID controller, you'll need to add those modules as well. Finally, if you're using software RAID, disk encryption, or LVM, you'll also need to create an initrd regardless of whether you're using the generic kernel or not.

An initrd is a compressed ***cpio***(1) archive, so creating one isn't very straightforward. Fortunately for you, Slackware includes a tool that makes this very easy: ***mkinitrd***. A full discussion of ***mkinitrd*** is a bit beyond the scope of this book, but we'll show you all the highlights. For a more complete explanation, check the manpage or run ***mkinitrd*** with the *–help* argument.

```
darkstar:~# mkinitrd --help
mkinitrd creates an initial ramdisk (actually an initramfs cpio+gzip
archive) used to load kernel modules that are needed to mount the
root filesystem, or other modules that might be needed before the
root filesystem is available.  Other binaries may be added to the
initrd, and the script is easy to modify.  Be creative.  :-)
.... many more lines deleted ....
```

When using ***mkinitrd***, you'll need to know a few items of information: your root partition, your root filesystem, any hard disk controllers you're using, and whether or not you're using LVM, software RAID, or disk encryption. Unless you're using some kind of SCSI controller (and have your root partition located on the SCSI controller), you should only need to know your root filesystem and partition type. Assuming you've booted into your Slackware installation using the huge kernel, you can easily find this information with the ***mount*** command or by viewing the contents of `/proc/mounts`.

```
darkstar:~# mount
/dev/sda1 on / type ext4 (rw,barrier=1,data=ordered)
proc on /proc type proc (rw)
sysfs on /sys type sysfs (rw)
usbfs on /proc/bus/usb type usbfs (rw)
/dev/sda2 on /home type jfs (rw)
tmpfs on /dev/shm type tmpfs (rw)
```

In the example provided, you can see that the root partition is located on `/dev/sda1` and is an ext4 type partition. If we want to create an initrd for this system, we simply need to tell this information to

mkinitrd.

```
darkstar:~# mkinitrd -f ext4 -r /dev/sda1
```

Note that in most cases, ***mkinitrd*** is smart enough to determine this information on its own, but it never hurts to specify it manually. Now that we've created our initrd, we simply need to tell LILO where to find it. We'll focus on that in the next section.

Looking up all those different options for ***mkinitrd*** or worse, memorizing them, can be a real pain though, especially if you try out different kernels consistently. This became tedious for the Slackware development team, so they came up with a simple configuration file, `mkinitrd.conf`(5). You can find a sample file that can be easily customized for your system at `/etc/mkinitrd.conf.sample` directory. Here's mine.

```
darkstar:~# >/prompt>cat /etc/mkinitrd.conf.sample
# See "man mkinitrd.conf" for details on the syntax of this file
#
SOURCE_TREE="/boot/initrd-tree"
CLEAR_TREE="0"
OUTPUT_IMAGE="/boot/initrd.gz"
KERNEL_VERSION="$(uname -r)"
#KEYMAP="us"
MODULE_LIST="ext3:ext4:jfs"
#LUKSDEV="/dev/hda1"
ROOTDEV="/dev/sda1
ROOTFS="ext4"
#RESUMEDEV="/dev/hda2"
#RAID="0"
LVM="1"
#WAIT="1"
```

For a complete description of each of these lines and what they do, you'll need to consult the man page for `mkinitrd.conf`. Copy the sample file to to `/etc/mkinitrd.conf` and edit it as desired. Once it is setup properly, you need only run ***mkinitrd*** with the *-F* argument. A proper initrd file will be constructed and installed for you without you having to remember all those obscure arguments.

If you're unsure what options to specify in the configuration file or on the command-line, there is one final option. Slackware includes a nifty little utility that can tell what options are required for your currently running kernel ***/usr/share/mkinitrd/mkinitrd_command_generator.sh***. When you run this script, it will generate a command line for ***mkinitrd*** that should work for your computer, but you may wish to check everything anyway.

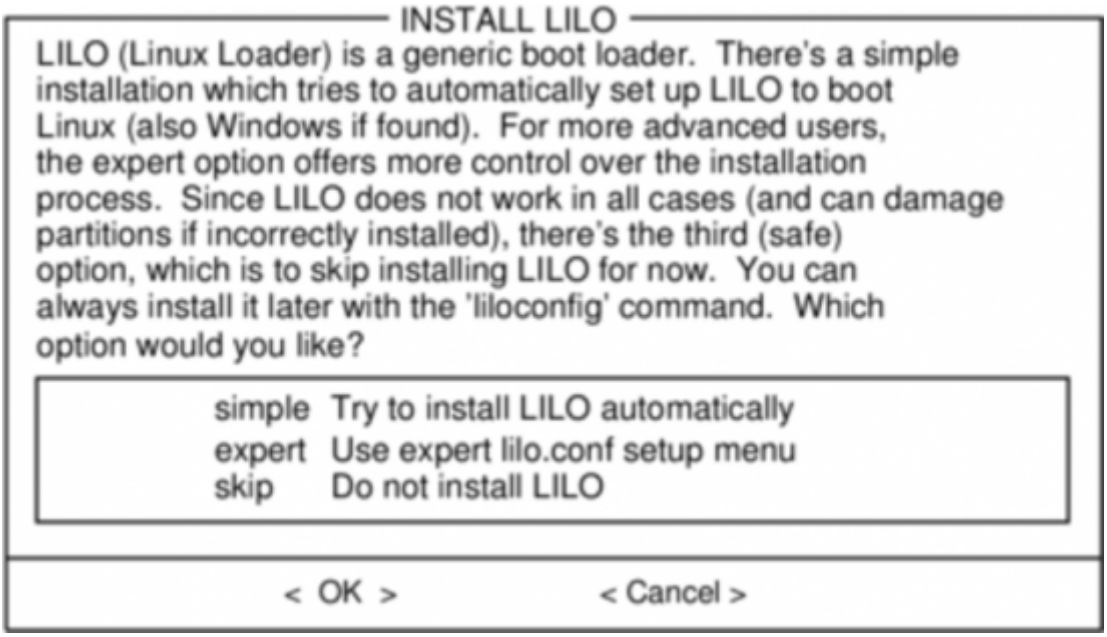
```
darkstar:~# /usr/share/mkinitrd/mkinitrd_command_generator.sh
mkinitrd -c -k 2.6.33.4 -f ext3 -r /dev/sda3 -m \
  usbhid:ehci-hcd:uhci-hcd:ext3 -o /boot/initrd.gz
```

LILO

LILO is the Linux Loader and is currently the default boot loader installed with Slackware Linux. If

you've used other Linux distributions before, you may be more familiar with GRUB. If you prefer to use GRUB instead, you can easily find it in the `extra/` directory on one of your Slackware CDs. However, since LILO is the default Slackware bootloader, we'll focus exclusively on it.

Configuring LILO can be a little daunting for new users, so Slackware comes with a special setup tool called ***liloconfig***. Normally, ***liloconfig*** is first run by the installer, but you can run it at any time from a terminal.



liloconfig has two modes of operation: simple and expert. The *"simple"* mode tries to automatically configure lilo for you. If Slackware is the only operating system installed on your computer, the *"simple"* mode will almost always do the right thing quickly and easily. It is also very good at detecting Windows installations and adding them to `/etc/lilo.conf` so that you can choose which operating system to boot when you turn your computer on.

In order to use *"expert"* mode, you'll need to know Slackware's root partition. You can also setup other linux operating systems if you know their root partitions, but this may not work as well as you expect. ***liloconfig*** will try to boot each linux operating system with Slackware's kernel, and this is probably not what you want. Fortunately, setting up Windows partitions in expert mode is trivial. One hint when using expert mode: you should almost always install LILO to the Master Boot Record (MBR). Once upon a time, it was recommended to install the boot loader onto the root partition and set that partition as bootable. Today, LILO has matured greatly and is safe to install on the MBR. In fact, you will encounter fewer problems if you do so.

liloconfig is a great way to quickly setup your boot loader, but if you really need to know what's going on, you'll need to look at LILO's configuration file: `lilo.conf`(5) under the `/etc` directory. `/etc/lilo.conf` is separated into several sections. At the top, you'll find a *"global"* section where you specify things like where to install LILO (generally the MBR), any special images or screens to show on boot, and the timeout after which LILO will boot the default operating system. Here's what the global section of my lilo.conf file looks like in part.

```
# LILO configuration file

boot = /dev/sda
  bitmap = /boot/slack.bmp
```

```
  bmp-colors = 255,0,255,0,255,0
  bmp-table = 60,6,1,16
  bmp-timer = 65,27,0,255

append=" vt.default_utf8=0"
prompt
timeout = 50

# VESA framebuffer console @ 1024x768x256
vga = 773
.... many more lines ommitted ....
```

For a complete listing of all the possible LILO options, you should consult the man page for `lilo.conf`. We'll briefly discuss the most common options in this document.

The first thing that should draw your attention is the *"boot"* line. This determines where the bootloader is installed. In order to install to the Master Boot Record (MBR) of your hard drive, you simply list the hard drive's device entry on this line. In my case, I'm using a SATA hard drive that shows up as a SCSI device `/dev/sda`. In order to install to the boot block of a partition, you'll have to list the partition's device entry. For example, if you are installing to the first partition on the only SATA hard drive in your computer, you would probably use `/dev/sda1`.

The *"prompt"* option simply tells LILO to ask (prompt) you for which operating system to boot. Operating systems are each listed in their own section deeper in the file. We'll get to them in a minute. The timeout option tells LILO how long to wait (in tenths of seconds) before booting the default OS. In my case, this is 5 seconds. Some systems seem to take a very long time to display the boot screen, so you may need to use a larger timeout value than I have set. This is in part why the simple LILO installation method utilizes a very long timeout (somewhere around 2 whole minutes). The append line in my case was set up by ***liloconfig***. You may (and probably should) see something similar when looking at your own `/etc/lilo.conf`. I won't go into the details of why this line is needed, so you're just going to have to trust me that things work better if it is present. :^)

Now that we've looked into the global section, let's take a look at the operating systems section. Each linux operating system section begins with an *"image"* line. Microsoft Windows operating systems are specified with an *"other"* line. Let's take a look at a sample `/etc/lilo.conf` that boots both Slackware and Microsoft Windows.

```
# LILO configuration file
... global section ommitted ....
# Linux bootable partition config begins
image = /boot/vmlinuz-generic-2.6.29.4
  root = /dev/sda1
  initrd = /boot/initrd.gz
  label = Slackware64
  read-only
# Linux bootable partition config ends
# Windows bootable partition config begins
other = /dev/sda3
  label = Windows
  table = /dev/sda
# Windows bootable partition config ends
```

For Linux operating systems like Slackware, the image line specifies which kernel to boot. In this case, we're booting `/boot/vmlinuz-generic-2.6.29.4`. The remaining sections are pretty self-explanatory. They tell LILO where to find the root filesystem, what initrd (if any) to use, and to initially mount the root filesystem read-only. That initrd line is very important for anyone running a generic kernel or using LVM or software RAID. It tells LILO (and the kernel) where to find the initrd you created using ***mkinitrd***.

Once you've gotten `/etc/lilo.conf` set up for your machine, simply run ***lilo***(8) to install it. Unlike GRUB and other bootloaders, LILO requires you re-run ***lilo*** anytime you make changes to its configuration file, or else the new (changed) bootloader image will not be installed, and those changes will not be reflected.

```
darkstar:~# lilo
Warning: LBA32 addressing assumed
Added Slackware *
Added Backup
6 warnings were issued.
```

Don't be too scared by many of the warnings you may see when running ***lilo***. Unless you see a fatal error, things should be just fine. In particular, the LBA32 addressing warning is commonplace.

Dual Booting

A bootloader (like LILO) is a very flexible thing, since it exists only to determine which hard drive, partition, or even a specific kernel on a partition to boot. This inherently suggests a choice when booting, so the idea of having more than one operating system on a computer comes very naturally to a LILO or GRUB user.

People *"dual boot"* for a number of reasons; some people want to have a stable Slackware install on one partition or drive and a development sandbox on another, other people might want to have Slackware on one and another Linux or BSD distribution on another, and still other people may have Slackware on one partition and a proprietary operating system (for work or for that one application that Linux simply cannot offer) on the other.

Dual booting should not be taken lightly, however, since it usually means that you'll now have two different operating systems attempting to manage the bootloader. If you dual boot, the likelihood of one OS over-writing or updating the bootloader entries without your direct intervention is great; if this happens, you'll have to modify GRUB or LILO manually so you can get into each OS.

There are two ways to dual (or multi) boot; you can put each operating system on its own hard drive (common on a desktop, with their luxury of having more than one drive bay) or each operating system on its own partition (common on a laptop where only one physical drive is present).

Dual Booting with Partitions

In order to set up a dual-boot system with each operating system on its own partition, you must first create partitions. This is easiest if done prior to installing the first operating system, in which case it's a simple case of pre-planning and carving up your hard drive as you feel necessary. See the section

called “Partitioning” for information on using the ***fdisk*** or ***cfdisk*** partitioning applications.

If you're dual booting two Linux distributions, it is inadvisable to attempt to share a /home directory between the systems. While it is technically possible, doing so will increase the chance of your personal configurations from becoming mauled by competing desktop environments or versions.

It is, however, safe to use a common swap partition.

You should partition your drive into at least three parts:

- One partition for Slackware
- One partition for the secondary OS
- One partition for swap

First, install Slackware Linux onto the first partition of the hard drive as described in Chapter 2, Installation.

After Slackware has been installed, booted, and you've confirmed that everything works as expected, then reboot to the installer for the second OS. This OS will invariably offer to utilize the entire drive; you obviously do **not** want to do that, so constrain it to only the second partition. Furthermore, the OS will attempt to install a boot loader to the beginning of the hard drive, overwriting LILO.

You have a few possible courses of action with regards to the boot loader:

Possible Boot Loader Scenarios

If the secondary OS is Linux, disallow it from installing a boot manager.
If you're dual booting to another Linux distribution, the installer of that distribution usually asks if you want a boot loader installed. You're certainly free to not install a boot manager for it at all, and manually manage both Slackware and the other distribution with LILO.

Depending on the distribution, you might be editing LILO more frequently than you would if you were only running Slackware; some distributions are notorious for frequent kernel updates, meaning that you'll need to edit LILO to reflect the new configuration after such an update. But if you didn't want to edit configuration files every now and again, you probably wouldn't have chosen Slackware.

If the secondary OS is Linux, let it overwrite LILO with GRUB.

If you're dual booting to another Linux distribution, you are perfectly capable of just using GRUB rather than LILO, or install Slackware last and use LILO for both. Both LILO and GRUB have very good auto-detection features, so whichever one gets installed last should pick up the other distribution's presence and make an entry for it.

Since other distributions often attempt to auto-update their GRUB menus, there is always the chance that during an update something will become maligned and you suddenly find you can't boot into Slackware. If this happens, don't panic; just boot into the other Linux partition and manually edit GRUB so that it points to the correct partition, kernel, and initrd (if applicable) for Slackware in its menu.

Allow the secondary OS to overwrite LILO and go back later to manually re-install and re-configure LILO.

This is not a bad choice, especially when Windows is the secondary OS, but potential pitfalls are that when Windows updates itself, it may attempt to overwrite the MBR (master boot record) again, and you'll have to re-install LILO manually again.

To re-establish LILO after another OS has erased it, you can boot from your Slackware install media and enter the setup stage. Do <emphasis>not</emphasis> re-partition your drive or re-install Slackware; skip immediately to Configure.

Even when using the *"simple"* option to install, LILO should detect both operating systems and automatically configure a sensible menu for you. If it fails, then add the entries yourself.

Dual Booting from Hard Drives

Dual booting between different physical hard drives is often easier than with partitions since the computer's BIOS or EFI almost invariably has a boot device chooser that allows you to interrupt the boot process immediately after POST and choose what drive should get priority.

The snag key to enter the boot picker is different for each brand of motherboard; consult the motherboard's manual or read the splash screen to find out what your computer requires. Typical keys are F1, F12, DEL. For Apple computers, it is always the Option (Alt) key.

If you manage the boot priority via BIOS or EFI, then each boot loader on each hard drive is only aware of its own drive and will never interfere with one another. This is rather contrary to what a boot loader is designed to do but can be a useful workaround when dealing with proprietary operating systems which insist upon being the only OS on the system, to the detriment of the user's preference.

If you don't have the luxury of having multiple internal hard drives and don't feel comfortable juggling another partition and OS on your computer, you might also consider using a bootable USB thumbdrive or even a virtual machine to give you access to another OS. Both of these options is outside the scope of this book, but they've commonplace and might be the right choice for you, depending on your needs.

Chapter Navigation

Previous Chapter: Installation

Next Chapter: Basic Shell Commands

Sources

- Original source: http://www.slackbook.org/beta
- Originally written by Alan Hicks, Chris Lumens, David Cantrell, Logan Johnson

slackware, booting, mkinitrd, lilo, dual-boot