Configuring devices with Debian GNU/Linux

Romain Lerallut

rom1@users.sourceforge.net

The aim of this document is to help people install basic pieces of hardware, such as soundcards.

SGML source for this document is available on the Newbiedoc Project (newbiedoc.sourceforge.net)'s site. This is GPL territory.

1. Introduction

A basic Debian installation will usually set up several peripherals, including disks (floppies, IDE and SCSI), networking devices and the screen-related stuff (if you're lucky and your video card is supported).

However, it will usually NOT include many essential devices, such as sound cards, TV cards, and a lot of other exotic boards I don't know about but which I'll help you set up.

Pre-requisites:

Well, I'll try to make it as simple as possible, to explain difficult stuff, and to guide you step by step. So I don't think any special knowledge is required, however, I'll assume you're familiar with Unix concepts such as 'kernel' and 'root' and 'compiling'. But that's all. If you don't feel like that at all, just tell me, and I'll update this paragraph.

This documentation is strongly tainted by the fact that I only use an i386 (a PC) architecture. The software configuration should be platform-independent, but not the first few

paragraphs, about hardware configuration.

As usual, comments, both positive (good for the mood) and negative are welcome. Really.

CAUTION: Your Mileage May Vary. Read thoroughly the documentation that comes with new hardware, including FAQs and HOW-TOs to be found on the vendor's website. They may include specific step-by-step instructions, help you avoid some traps, etc.

1.1. Credits

Thanks to the people at the NewbieDoc project, especially to Jon Eisenstein for his constant good will to critique our work.

1.2. Feedback

Not-quite-real press reviews:

"Great stuff ! Keep up the good work" -- New York Time

"One of the best docs around" -- Computer Quarterly

If you wish to comment on this doc, do not hesitate to email me. Some points may be wrong, and others may need clarifying. Note: I'm not a native english speaker, so I make grammatical mistakes, do not hesitate to correct them and send a patch.

2. Preparing the hardware

There are two categories of hardware: Plug-and-play, and , err, non-Plug-and-play. If your piece of hardware is recent, like a PCI board, then it's likely to be PnP. If it is not that recent, (though perfectly fit for duty :) then it may well be non-PnP.

If you also use Windows, you can use it to get info about your hardware. See below .

2.1. Plug and Play

Plug and Play is basically one thing: knowing where, and how to talk to the device. It is not *directly* related to autodetecting new hardware.

If your hardware is plug-and-play, as is the case with most modern stuff (almost all PCI devices, and a few of the last ISA boards), you may plug it and jump directly to the section about *Configuring the software*.

Disable the BIOS feature 'PnP OS': Linux can't assign resources by itself. Disabling it means the BIOS does the assigning.

Actually, it is a good idea to fiddle with the settings to prevent two highly-used devices from sharing the same resources. See your BIOS documentation.

2.2. non-Plug-and-Play

What I basically know is that you must tweak some settings (IRQ, DMA, IO address) on the board by moving jumpers. I've been told some devices could only be configured by using MSDOS/Windows tools.

Either way, you must find free resources (IRQ, DMA), as most non-PnP devices do not support sharing. Then write them on paper and feed them to the driver.

2.3. If you use windows

If you happen to use Windows on the same machine, you can use it to extract lots of info about your hardware, (assuming it's working satisfyingly under Windows).

Right-click on 'My Computer' , choose 'properties' and you'll see a list of all your hardware. Copy down the parameters such as IRQ, DMA, IO ports, etc. Using these will save you a lot of time.

3. Configuring the software

You've plugged your board in your computer, and you want Linux to recognize it. You now have to load and configure the driver that will know how to talk to your board.

If you have a Plug and Play device, it will know how to talk to the driver and tell it its IRQ, DMA, etc... So you don't need to know about them.

However, if some specific options are needed, or if your board isn't PnP, you'll just have to say to the driver the settings you've defined earlier (IRQ, DMA, for non-PnP), and quite possibly other settings. It depends on your hardware. See *Passing Options to a module...*.

3.1. What kind of driver do you have ?

I'll prefix all commands that *need to be run as root* with a '#', since that's the default prompt when you login as root.

Easy case: the driver is included in the kernel distribution. The magic word is:

```
# modconf
```

Browse through the menus until you find the right module, then press enter and say 'Yes' when asked if you want the modules loaded in the kernel.

if you need to pass arguments to the module, see the syntax below. If not, just press ENTER.

Medium case: you need to recompile your kernel. Usually that's because you're not using a standard kernel, or because your hardware is a bit too 'exotic'. You need to recompile your kernel (no big deal, usually). Since we at Newbiedoc think things ahead :-), there is a Kernel Compiling Doc (http://newbiedoc.sourceforge.net/tutorials/kernel-pkg.en/index-kernel-pkg.html).

Patching a kernel: sometimes patches are available that modify your kernel source. To apply a patch:

```
cd /usr/src/kernel-source-xx.yy.zz
cat my_patch | patch -p0
```

(CHECK!! I can't seem to remember if it's -p0 or -p1 , but try the other if one doesn't work :-)

Hard case: You got the driver from elsewhere. Don't worry, it's not that bad if you have the source. You need first to compile it, then to try and load it, and then have it done automatically at boot-time. Section "*Compiling a driver*" is for you. The problem almost goes away (almost) if the driver's source is packaged as a Debian package, because most of the hard work will be done automatically for you. See *Compiling a Debian-packaged module*, below.

Evil case: Your driver is precompiled. Usually it is compiled against a given kernel (say 2.2.15 , 2.2.16, and 2.2.17). In this case, either you run the right kernel, and in that case it should work well. Or you don't run the right kernel, and you have to consider switching kernel, getting a different driver, or getting similar hardware from a more linux-friendly vendor. In all cases, read the docs that comes with the driver since some stuff will be very specific, I can't help you there.

3.2. Compiling a driver

For those whose driver is not included in the standard kernel

3.2.1. Compiling from a generic tarball

The standard way is to uncompress the source somewhere:

```
tar zxvf driver-src.tgz
```

Then run the 'configure' script:

```
cd unpacked_source_directory
./configure
```

If there is no 'configure' script, don't worry. It happens.

```
make
```

will compile the driver, leaving you with a file named, say, `my_new_driver.o`.

```
# make install
```

should be done as root and will install the driver where it should be installed. Check
`/lib/modules/kernel_version/wherever/my_new_driver.o`

3.2.2. Compiling a Debian-packaged module

For your convenience, some modules are packaged as Debian packages, specifically tailored to run well on a Debian system.

Those packages are usually of the form `foobar-src-xx.yy.deb`. To download one:

```
# apt-get install foobar-src
```

Apt will put the tarballs in /usr/src.

Uncompress the tarball: it will find its way to `/usr/src/modules/foobar`.

To compile the module, you either need the kernel source, or more simply the kernel-headers if you don't want to compile your kernel: it only contains the basic files.

The headers *must* be the exact ones for your kernel version.

`dpkg -l kernel-image`

will tell you which kernel-image you use

Once you're done, you should have a new directory: `/usr/src/kernel-source-xx.yy.zz` Go into that directory and do (as root):

```
# make-kpkg modules
```

It will compile all modules in `/usr/src/modules` and produce .debs for each: `foobar-xx.yy.zz_i386.deb`, for example, for module foobar compiled against kernel xx.yy.zz on an i386 platform.

Then you simply dpkg each .deb:

```
# dpkg -i foobar-xx.yy.zz_i386.deb
```

And each module will be installed in the correct place.

3.2.3. Testing the driver

The driver is compiled and ready for testing.

First of all, the dependencies between all modules need to be computed:

```
depmod
```

Then load the module:

```
# modprobe my_new_driver
```

It might need arguments, consult the README and see below the section about passing options to a module. To test if the module was correctly loaded:

```
# lsmod
```

will list all loaded modules. And to remove a module:

```
#rmmod my_new_driver
```

Some modules can't be removed because others depend on them. The names in brackets in an 'lsmod' are the modules dependencies.

3.3. Passing Options to a module...

3.3.1. ...at loadtime

Modules concerned by this are mostly those that talk to non Plug-and-Play devices, and some modules (PnP or not) that need a lot of info.

Usually (for non-PnP stuff) you need at least to tell the module the IRQ of the device it's talking to, and possibly some other stuff. This should load a good ol' SoundBlaster:

```
#modprobe sb irq=7 io=0x220 dma=1 dma16=6
```

Check if the modprobe succeeded by doing

```
#lsmod
```

there are dependencies for modules, so a module may request that others are loaded, and will load them by itself if they are available. For example, the 'sb' module might need the 'sound' module, etc.

Most settings are documented in the README file that came with the driver. For standard kernel drivers, see `/usr/src/kernel-source-XX.YY.ZZ/Documentation`. You may also want to check out the vendor's website for some FAQs and HOW-TOs.

3.3.2. ...Automatically

Modules can be loaded automatically, see below *Automatic Module Loading*

So where do I specify the options ? Add a file to the `/etc/modutils` directory (the name of the file doesn't matter). For the SoundBlaster module we loaded previously, the syntax would then be:

```
options sb irq 7 io 0x220 dma 1 dma16 5
```

The kernel also sometimes needs to be able to link a module to an entry in /dev:

```
alias char-major-195 NVdriver
```

This says that devices 195.* are to be managed by the NVdriver module. This is usually well documented in the doc that comes with the driver.

When all is done, do as root:

```
# update-modules
```

which will concatenate all the files in the `/etc/modutils` directory (including subdirs) and store that in `/etc/modules.conf` which can be read by modprobe. *Do NOT edit modules.conf directly, since it will be overwritten by the next update-modules.*

3.4. Automatic Module Loading

There are two ways for modules to be loaded:

1. If they are needed by another module, or by a program that needs access to a device (graphics board, sound card,...). In that case there's nothing to worry about. The modules know which other fellow modules they need thanks to 'depmod', and the programs actually use `/dev/*` files, which are linked to corresponding modules by the 'alias' keyword in `/etc/modules.conf`
2. If they appear in `/etc/modules`

To automatically load a module, just write its name in the file `/etc/modules` , without any options or path, just the name (such as 'sb','ne2k-pci','vfat', etc...)

If arguments are needed, the module will know by itself to look for them in `/etc/modules.conf`. See paragraph about passing options ...*Automatically*

3.5. Summary

To summarize:

- **Standard Modules on a stock kernel**

  1. as root do

     ```
     #modconf
     ```

     provide the correct options (if needed) and have fun...

- **Non Standard Modules**

  1. Compile the driver
  2. Find the correct settings using

     ```
     #modprobe my_module irq XX dma YY other_option ZZ
     ```

  3. Add a file in `/etc/modutils` containing these settings (if no setting is needed, then you may skip this step). And run

     ```
     #update-modules
     ```

  4. Try to load the module with

     ```
     #modprobe my_module
     ```

     This shouldn't fail if the previous step was correctly done.

  5. If you want (or need), reference your module in `/etc/modules`.
  6. Have fun...

4. Troubleshooting

Unfortunately, there are some cases when things do not go smoothly.

4.1. Trouble at the compilation

In case you have problem compiling your modules:

- Do you have all the tools required ? (make, gcc, make-kpkg, etc...)
- Do you have the correct headers installed ? (kernel headers, but also standard libc headers, etc...)
- have you checked that module X didn't need the source of module Y (header files, actually) to be successfully compiled ? If so download module Y's source, and try again.

4.2. Unresolved symbols

A grand classic.

The module you're trying to load can't find some functions it needs. That means: you haven't run 'depmod' or you lack a module.

If running 'depmod' doesn't solve your problem, read the doc again, and check that you have *all* the other modules that are needed.

4.3. Can't locate module char-major-xx

Another classic.

You need to add the corresponding 'alias' in `/etc/modutils`, or compile the module if it's not available.

Some aliases are defined by default in `/etc/modutils/aliases`. If you know what you're doing you can disable them there.

5. Finding more info

I can't stress it enough, the documentation that comes with your hardware is the Ultimate Reference.

Aside from that:

- The Linux Documentation Project (http://www.linuxdoc.org) and its collection of howtos.
  - The Hardware Howto (http://www.linuxdoc.org/HOWTO/Hardware-HOWTO/index.html) lists Linux-compatible hardware.
  - The Plug-and-Play Howto (http://www.linuxdoc.org/HOWTO/Plug-and-Play-HOWTO.html) sometimes a bit thick, but good info.
  - The specific Howtos: networking, graphics cards, etc...
- The Debian-User mailing list *and its archives*. You can search the archives there (http://lists.debian.org/search.html)