The Linux Printing HOWTO

Grant Taylor

gtaylor+pht@picante.com

Version $Revision: 1.2 $, $Date: 2000/09/19 20:36:53 $

This is the Linux Printing HOWTO, a collection of information on how to generate, preview, print and fax anything under Linux (and other Unices in general).

Table of Contents

1. Introduction..........1
1.1. History..........1
1.2. Copyright..........2

2. Quick Start..........3

3. How to print..........4
3.1. With PDQ..........4
3.1.1. Xpdq..........4
3.1.2. Pdq..........4
3.2. With LPD and the lpr command..........4
3.3. GUI Printing Tools..........5
3.3.1. GPR..........5
3.3.2. XPP..........7
3.3.3. XPDQ..........9

4. Kernel printer devices..........10
4.1. The lp device (kernels <=2.1.32)..........10
4.2. The parport device (kernels >= 2.1.33)..........11
4.3. Serial devices..........11
4.4. USB Devices..........12

5. Supported Printers..........13
5.1. Postscript..........13
5.2. Non−Postscript..........14
5.3. What printers work?..........14
5.3.1. Printer compatibility list..........15
5.4. How to buy a printer..........25
5.4.1. What do I have?..........27

6. Spooling software..........28
6.1. LPD..........28
6.1.1. LPD front−ends..........28
6.2. PDQ..........29
6.3. LPRng..........30
6.4. PPR..........30
6.5. CUPS..........31

7. How it all works..........33
7.1. PDQ..........33
7.2. LPD..........34

8. How to set things up..........36
8.1. Configuring PDQ..........36
8.1.1. Drivers and Interfaces..........36
8.1.2. Defining Printers..........37
8.1.3. Creating a PDQ Driver Declaration..........37

Table of Contents

8.2. Configuring LPD.....42
8.2.1. Basic LPD configuration.....42
8.2.2. LPD for PostScript Printers.....44
8.2.3. File Permissions.....46
8.3. Large Installations.....46
8.4. Accounting.....47

9. Vendor Solutions.....49
9.1. Red Hat.....49
9.2. Debian.....49
9.3. SuSE.....49
9.4. Caldera.....50
9.5. Corel.....50
9.6. Mandrake.....50
9.7. Other Distributions.....51

10. Ghostscript.....52
10.1. Invoking Ghostscript.....52
10.2. Ghostscript output tuning.....52
10.2.1. Output location and size.....53
10.2.2. Gamma, dotsizes, etc.....53
10.2.3. Color Printing in Ghostscript.....53

11. Networks.....55
11.1. Printing to a Unix/lpd host.....55
11.1.1. With pdq.....55
11.1.2. With lpd.....55
11.1.3. With rlpr.....55
11.2. Printing to a Windows or Samba printer.....56
11.2.1. From PDQ.....56
11.2.2. From LPD.....56
11.3. Printing to a NetWare Printer.....56
11.3.1. From LPD.....57
11.4. Printing to an EtherTalk (Apple) printer.....57
11.4.1. From PDQ.....57
11.5. Printing to a networked printer.....57
11.5.1. To AppSocket Devices.....58
11.6. Running an if for remote printers with old LPDs.....60
11.7. From Windows.....61
11.8. From an Apple.....62
11.9. From Netware.....62
11.10. Networked Printer Administration.....62
11.10.1. npadmin.....62
11.10.2. Other SNMP tools.....63

12. Windows−only printers.....64
12.1. The Ghostscript Windows redirector.....64

Table of Contents

12.2. HP Winprinters....64
12.3. Lexmark Winprinters....64

13. How to print to a fax machine....65
13.1. Using a faxmodem....65
13.1.1. Faxing from PDQ....65
13.2. Using the Remote Printing Service....66
13.3. Commercial Faxing Services....66

14. How to generate something worth printing....67
14.1. Markup languages....67
14.2. WYSIWYG Word Processors....68

15. Printing Photographs....71
15.1. Ghostscript and Photos....71
15.2. Paper....71
15.3. Printer Settings....72
15.4. Print Durability....72
15.5. Shareware and Commercial Software....72

16. On−screen previewing of printable things....73
16.1. PostScript....73
16.2. TeX dvi....74
16.3. Adobe PDF....74

17. Serial printers under lpd....76
17.1. Setting up in printcap....76
17.2. Older serial printers that drop characters....77

18. What's missing?....78
18.1. Plumbing....78
18.2. Fonts....78
18.3. Metadata....78
18.4. Drivers....79

19. Credits....80

1. Introduction

The Printing HOWTO should contain everything you need to know to help you set up printing services on your Linux box(en). As life would have it, it's a bit more complicated than in the point−and−click world of Microsoft and Apple, but it's also a bit more flexible and certainly easier to administer for large LANs.

This document is structured so that most people will only need to read the first half or so. Most of the more obscure and situation−dependent information in here is in the last half, and can be easily located in the Table of Contents, whereas most of the information through section 8 or 9 is probably needed by most people.

If you find this document or the LinuxPrinting.org website useful, consider buying something through my referral association with buy.com or outpost.com; please use the links on the suggested printers page so that your purchase can be credited to LinuxPrinting.org.

Since version 3.x is a complete rewrite, some information from previous editions has been lost. This is by design, as the previous HOWTOs were so large as to be 60 typeset pages, and had the narrative flow of a dead turtle. If you do not find your answers here, you are encouraged to a) look on the LinuxPrinting.org website and b) drop me a note saying what ought to be here but isn't.

The LinuxPrinting.org website is a good place to find the latest version; it is also, of course, distributed from Metalab (`metalab.unc.edu`) and your friendly local LDP mirror.

1.1. History

This is the fourth generation of the Printing HOWTO. The history of the PHT may be chronicled thusly:

1. I wrote the printing−howto in 1992 in response to too many printing questions in comp.os.linux, and posted it. This predated the HOWTO project by a few months and was the first FAQlet called a \`howto'. This edition was in plain ascii.
2. After joining the HOWTO project, the Printing−HOWTO was merged with an Lpd FAQ by Brian McCauley `<B.A.McCauley@bham.ac.uk>`; we continued to co−author the PHT for two years or so. At some point we incorporated the work of Karl Auer `<Karl.Auer@anu.edu.au>`. This generation of the PHT was in TeXinfo, and available in PS, HTML, Ascii, and Info.
3. After letting the PHT rot and decay for over a year, and an unsuccessful attempt at getting someone else to maintain it, this rewrite happened. This generation of the PHT is written in SGML using the LinuxDoc DTD and the SGML−Tools−1 package. Beginning with version 3.27, it incorporates a summary of a companion printer support database; before 3.27 there was never a printer compatibility list in this HOWTO (!).
4. In mid−January, 2000, I found out about the PDQ print "spooler". PDQ provides a printing mechanism so much better than lpd ever did that I spent several hours playing with it, rewrote parts of this HOWTO, and bumped the version number of the document to 4.
5. In mid−2000, I moved my printing website to `www.linuxprinting.org`, and began offering more powerful configuration tools there. I also converted the HOWTO to DocBook, and initiated coverage of CUPS, LPRng, and GPR/libppd.

1.2. Copyright

2. Quick Start

The quickest way to get started is simply to use the setup tools provided by your vendor. Assuming that this includes support for your driver, and assuming that your vendor shipped the driver for your printer, then it should be easy to get a basic setup going this way. For information on vendor−provided setup tools, see Section 9.

If your vendor's tool doesn't work out, you should figure out if your printer is supposed to work at all. Consult the printer compatibility listings in Section 5.3.1 as well as the online version described there.

If your printer is known to work with a driver, check that you have that driver, and install if it not. Typically you will be able to find a contributed Ghostscript package including newer Ghostscript code and assorted third−party drivers. If not, you can compile it yourself; the process is not trivial, but it is well documented. See Section 10 for more information on Ghostscript.

After installing the proper driver, attempt again to configure your printer with your vendor's tools. If that fails, select a suitable third party tool from those described in Section 8. If that also fails, you'll need to construct your own setup; again see Section 8.

If you're still stuck, you've got a little troubleshooting to do. It's probably best to read most of this document first to get a feel for how things are supposed to work; then you'll be in a better position to debug.

3. How to print

You actually use a different command to print depending on which spooling software you use.

3.1. With PDQ

Most systems today ship with lpd, so this section won't apply. That said, I now recommend that people install and use PDQ in most cases instead of (or in addition to) lpd. PDQ just has much better support for printer options and such.

With PDQ, instead of the lpr command, you use the command pdq or xpdq. Both work much like the traditional lpr in that they will print the files you specify, or stdin if no files are given.

3.1.1. Xpdq

Xpdq is an X Windows application that shows a list of available printers and a summary of the print queue (including current and historical jobs). There are two options under the File menu, one to print specific files, and one to print stdin. You can set whatever options are defined in your printer driver from the Driver Options dialog; typically there will be duplex, resolution, paper type and size settings, and so forth.

3.1.2. Pdq

The PDQ system's command−line printing command is simply called **pdq**. It can be used in place of the lpr command in most situations; it accepts the `−P` printer specification argument. Like lpr, it prints either the listed file(s) or stdin.

Printer options can be controlled with the `−o` and `−a` options.

3.2. With LPD and the lpr command

If you've already got lpd setup to print to your printer, or your system administrator already did so, or your vendor did so for you, then all you need to do is learn how to use the lpr command. The Printing Usage HOWTO covers this, and a few other queue manipulation commands you should probably know. Or just read the lpr(1) man page.

In a nutshell, you specify the queue name with `−P`, and specify a filename to print a file, or nothing to print from stdin. Driver options are traditionally not controllable from lpr, but various systems accept certain options with `−o`, `−Z`, or `−J`.

If, however, you have a new system or new printer, then you'll have to set up printing services one way or another before you can print. Read on!

3.3. GUI Printing Tools

Most spooling systems alone offer only a rather basic command−line interface. Rather than use **lpr** directly, you may wish to obtain and use a front−end interface. These generally let you fiddle with various printing options (the printer, paper types, collation, n−up, etc) in an easy−to−use graphical way. Some may have other features, as well.

3.3.1. GPR

GPR, by Thomas Hubbell, uses code from CUPS to filter Postscript jobs and offer easy user control over job options. Some options (like n−way printing, page selection, etc) are implemented directly by GPR, while most others are implemented by the printer or by the spooler's filter system.

GPR works with LPD or LPRng; or can be compiled specifically for use with VA Linux's modified LPD. When compiled normally, it uses VA's libppd directly to produce printer−specific PostScript which it will then submit to the lpr command. When compiled for VA's LPD, it will submit your unmodified job PostScript to the lpr command, along with the set of job options you specify. This is arguably the better route, since it allows the Postscript to be redirected to a different printer by the spooler when appropriate; unfortunately it required VA's special LPD, which is not in wide circulation yet (although it is of course trivial to install).

To use GPR, first select a printer (by LPD queue name) and check that GPR has loaded the proper PPD file. If it hasn't, you'll need to specify the PPD filename, and specify your printer's options in the Printer Configuration dialog (you get this dialog by pressing the Printer Configuration button; it contains assorted printer setup options defined by the PPD).

Once you've configured your printer in GPR, you can print jobs by specifying the filename and selecting the proper options from the `Common' and `Advanced' tabbed panels. The `Common' options are implemented directly by GPR for all printers, while the `Advanced' options are defined by the PPD file for your printer. You can see these option panels in Figure 2 and Figure 3.

Figure 1. GPR Main Options

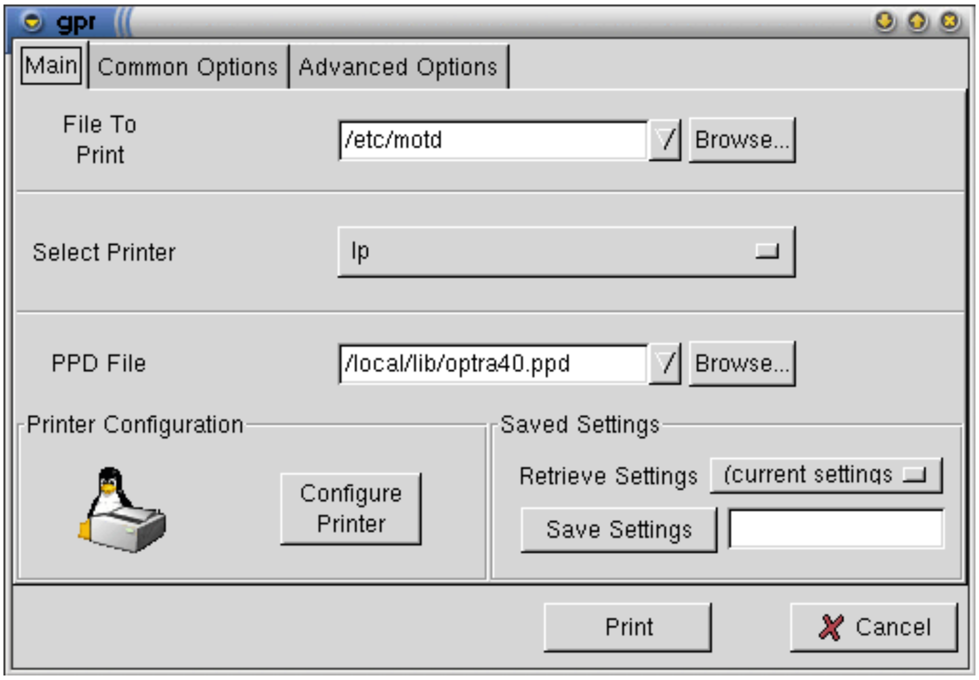


Figure 2. GPR Common Options

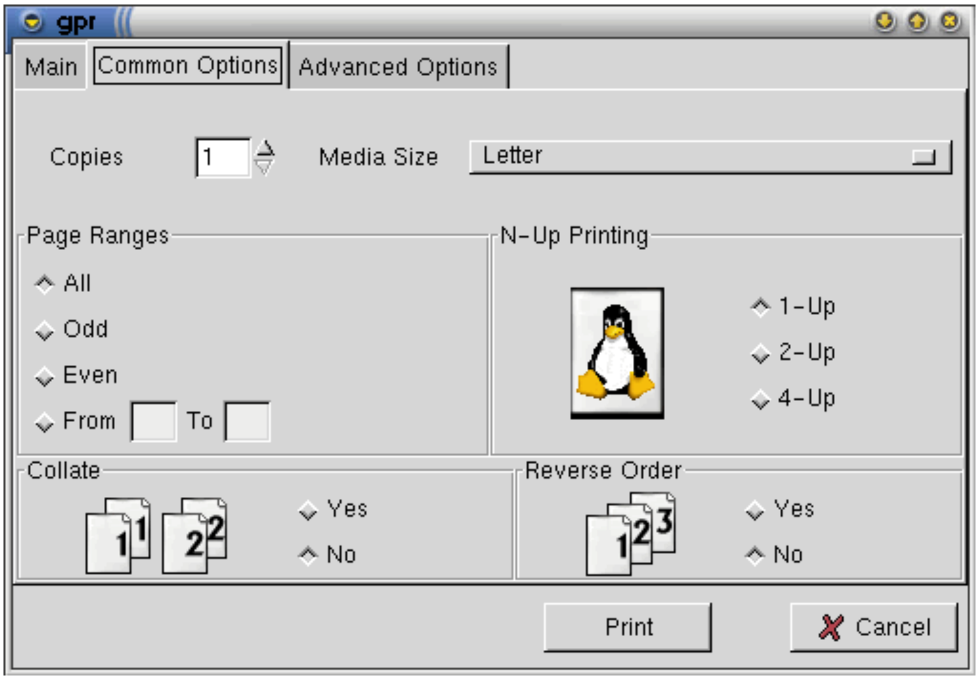
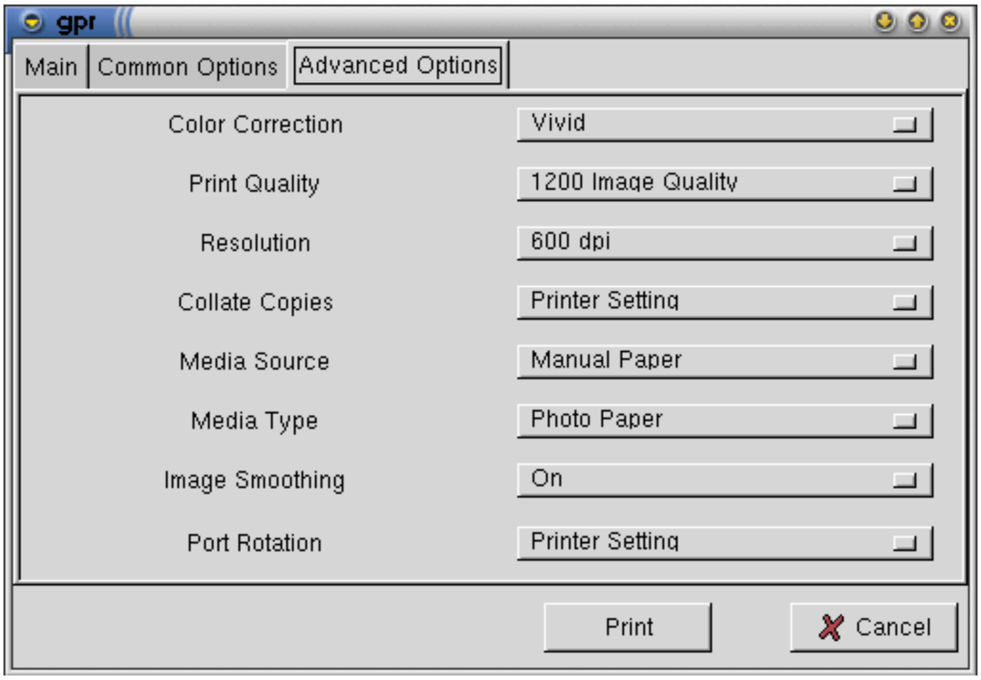


Figure 3. GPR Printer Options



3.3.2. XPP

If you use CUPS as your spooler, you can use the program **XPP** (see Figure 4).

To print with XPP, simply run the xpp program, and specify a file (or nothing, if you're using xpp in place of lpr to print from stdin). Then select a printer from the list of configured printers, and select any options you'd like to apply from the various tabbed panels. See Figure 5 for an example options panel highlighting the standard CUPS options.

You can save your selected printer and all the options with the `Save Settings' button.

Figure 4. XPP Main Window

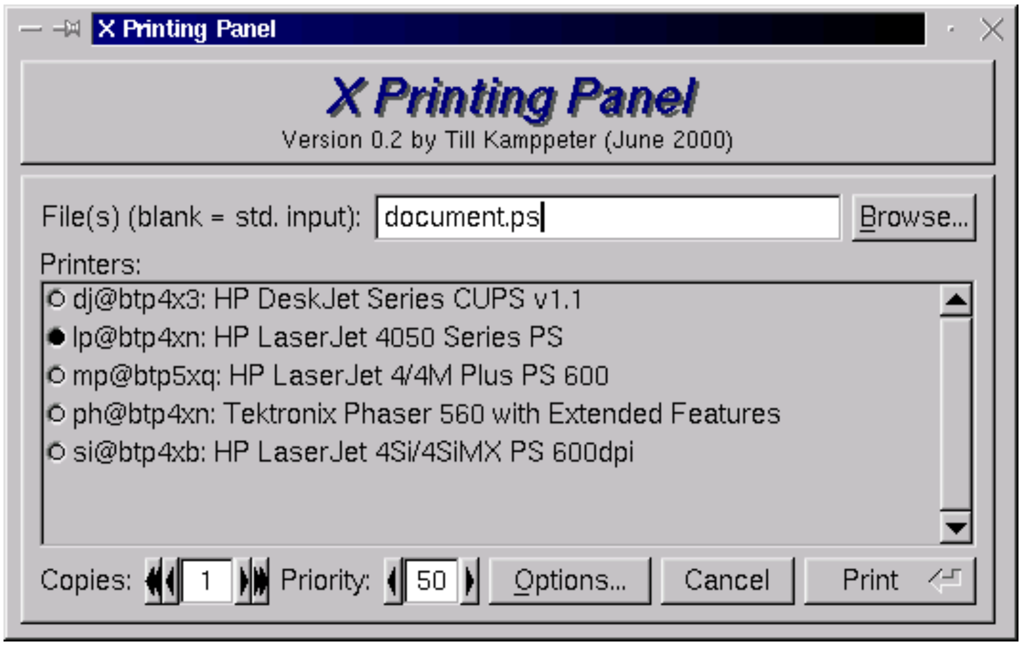
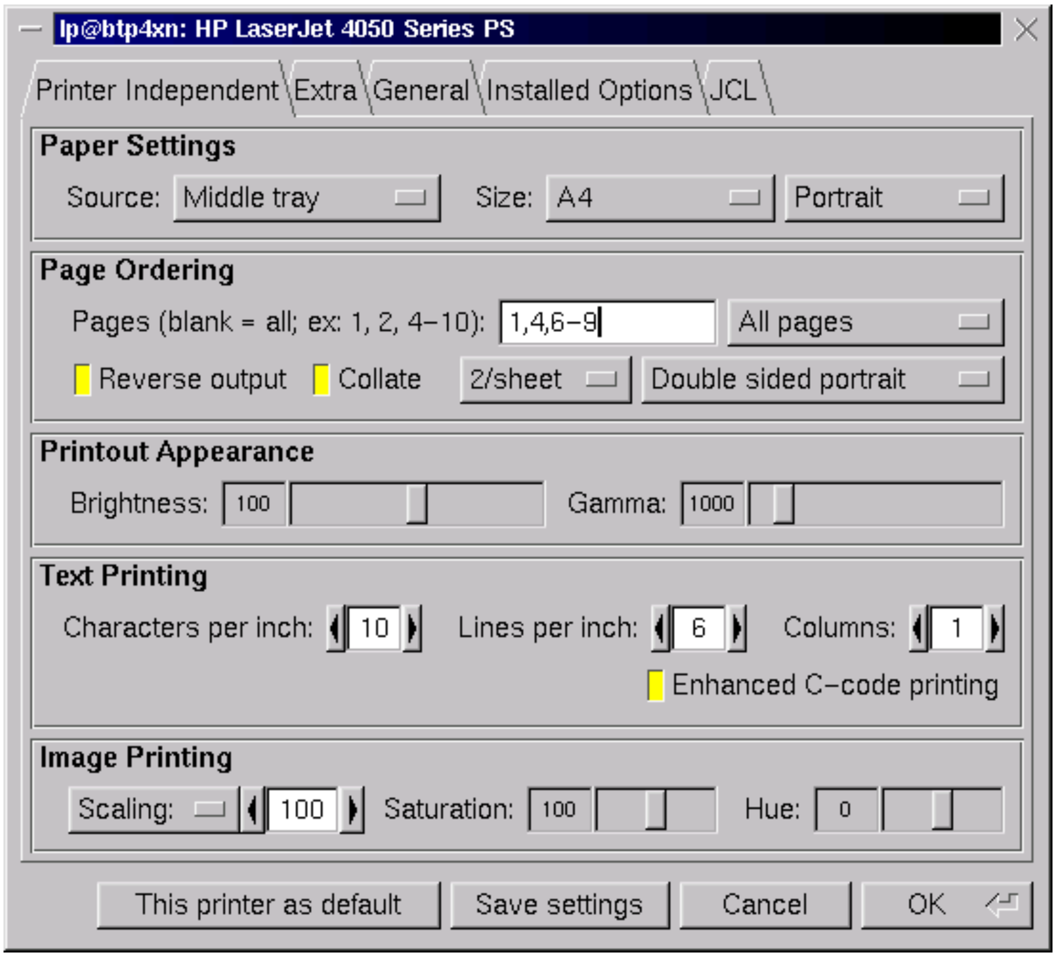


Figure 5. CUPS/XPP Options Window



3.3.3. XPDQ

PDQ can be easily configured to print to queues controlled by most spooling systems, and PDQ's configuration syntax offers a very easy way to define arbitrary filtering and user options for print jobs. So you can thus use **xpdq** as a front−end to LPD printing with great success.

For more information, see Section 6.2.

4. Kernel printer devices

There are two completely different device drivers for the parallel port; which one you are using depends on your kernel version (which you can find out with the command `uname -a`). The driver changed in Linux 2.1.33; essentially all current systems will be running kernel 2.2 or later, so you'll probably want to skip ahead to the parport driver section.

A few details are the same for both styles of driver. Most notably, many people have found that Linux will not detect their parallel port unless they disable "Plug and Play" in their PC BIOS. (This is no surprise; the track record for PnP of non−PCI devices with Windows and elsewhere has been something of a disaster).

4.1. The lp device (kernels <=2.1.32)

The Linux kernel (<=2.1.32), assuming you have compiled in or loaded the lp device (the output of `cat /proc/devices` should include the device lp if it is loaded), provides one or more of */dev/lp0*, */dev/lp1*, and */dev/lp2*. These are NOT assigned dynamically, rather, each corresponds to a specific hardware I/O address. This means that your first printer may be *lp0* or *lp1* depending on your hardware. Just try both.

A few users have reported that their bidirectional lp ports aren't detected if they use an older unidirectional printer cable. Check that you've got a decent cable.

One cannot run the plip and lp drivers at the same time on any given port (under 2.0, anyway). You can, however, have one or the other driver loaded at any given time either manually, or by kerneld with version 2.x (and later 1.3.x) kernels. By carefully setting the interrupts and such, you can supposedly run plip on one port and lp on the other. One person did so by editing the drivers; I eagerly await a success report of someone doing so with only a clever command line.

There is a little utility called `tunelp` floating about with which you, as root, can tune the Linux 2.0 lp device's interrupt usage, polling rate, and other options.

When the lp driver is built into the kernel, the kernel will accept an `lp=` option to set interrupts and io addresses:

```
When the lp driver is built in to the kernel, you may use the
LILO/LOADLIN command line to set the port addresses and interrupts
that the driver will use.

Syntax:      lp=port0[,irq0[,port1[,irq1[,port2[,irq2]]]]]

For example:   lp=0x378,0   or   lp=0x278,5,0x378,7 **

Note that if this feature is used, you must specify *all* the ports
you want considered, there are no defaults.  You can disable a
built-in driver with lp=0.
```

When loaded as a module, it is possible to specify io addresses and interrupt lines on the insmod command line (or in */etc/conf.modules* so as to affect kerneld) using the usual module argument syntax. The parameters are `io=port0,port1,port2` and `irq=irq0,irq1,irq2`. Read ye the man page for *insmod* for more information on this.

**For those of you who (like me) can never find the standard port numbers when you need them, they are as in the second example above. The other port (*lp0*) is at 0x3bc. I've no idea what interrupt it usually uses.

The source code for the Linux 2.0 parallel port driver is in /usr/src/linux/drivers/char/lp.c.

4.2. The parport device (kernels >= 2.1.33)

Beginning with kernel 2.1.33 (and available as a patch for kernel 2.0.30), the lp device is merely a client of the new parport device. The addition of the parport device corrects a number of the problems that plague the old lp device driver − it can share the port with other drivers, it dynamically assigns available parallel ports to device numbers rather than enforcing a fixed correspondence between I/O addresses and port numbers, and so forth.

The advent of the parport device has enabled a whole flock of new parallel−port drivers for things like Zip drives, Backpack CD−ROMs and disks, and so forth. Some of these are also available in versions for 2.0 kernels; look around on the web.

The main difference that you will notice, so far as printing goes, is that parport−based kernels dynamically assign lp devices to parallel ports. So what was lp1 under Linux 2.0 may well be lp0 under Linux 2.2. Be sure to check this if you upgrade from an lp−driver kernel to a parport−driver kernel.

The most popular problems with this device seems to stem from misconfiguration:

The Distribution

Some Linux distributions don't ship with a properly setup /etc/modules.conf (or /etc/conf.modules), so the driver isn't loaded properly when you need it to be. With a recent modutils, the proper magical lines from modules.conf seem to be:

```
alias /dev/printers lp             # only for devfs?
alias /dev/lp*      lp             # only for devfs?
alias parport_lowlevel parport_pc  # missing in Red Hat 6.0−6.1
```

The BIOS

Many PC BIOSes will make the parallel port into a Plug−and−Play device. This just adds needless complexity to a perfectly simple device that is nearly always present; turn off the PnP setting for your parallel prot ("LPT1" in many BIOSes) if your parallel port isn't detected by the Linux driver. The correct setting is often called "legacy", "ISA", or "0x378", but probably not "disabled".

You can also read the parport documentation in your kernel sources, or look at the parport web site.

4.3. Serial devices

Serial devices are usually called something like */dev/ttyS1* under Linux. The utility `stty` will allow you to interactively view or set the settings for a serial port; `setserial` will allow you to control a few extended attributes and configure IRQs and I/O addresses for non−standard ports. Further discussion of serial ports under Linux may be found in the Serial−HOWTO.

When using a slow serial printer with flow control, you may find that some of your print jobs get truncated. This may be due to the serial port, whose default behavior is to purge any untransmitted characters from its buffer 30 seconds after the port device is closed. The buffer can hold up to 4096 characters, and if your printer uses flow control and is slow enough that it can't accept all the data from the buffer within 30 seconds after printing software has closed the serial port, the tail end of the buffer's contents will be lost. If the command `cat file > /dev/ttyS2` produces complete printouts for short files but truncated ones for longer files, you may have this condition.

The 30 second interval can be adjusted through the "closing_wait" commandline option of setserial (version 2.12 and later). A machine's serial ports are usually initialized by a call to setserial in the rc.serial boot file. The call for the printing serial port can be modified to set the closing_wait at the same time as it sets that port's other parameters.

4.4. USB Devices

I don't have any USB devices to play with, so all I can offer are pointers. Once set up, you end up with the device file `/dev/usb/lp0`, much as you do with parallel ports, which will work fine in printcap or as a PDQ local−port device.

USB is documented at the Linux USB Website.

5. Supported Printers

The Linux kernel will let you speak with any printer that you can plug into a serial, parallel, or usb port, plus any printer on the network, but this alone is insufficient; you must also be able to generate data that the printer will understand. Primary among the incompatible printers are those referred to as "Windows" or "GDI" printers. They are called this because all or part of the printer control language and the design details of the printing mechanism are not documented. Typically the vendor will provide a Windows driver and happily sell only to Windows users; this is why they are called Winprinters. In some cases the vendor also provides drivers for NT, OS/2, or other operating systems.

Many of these printers *do not work* with Linux. A few of them do, and some of them only work a little bit (usually because someone has reverse engineered the details needed to write a driver). See the printer support list below for details on specific printers.

A few printers are in−between. Some of NEC's models, for example, implement a simple form of the standard printer language PCL that allows PCL−speaking software to print at up to 300dpi, but only NEC knows how to get the full 600dpi out of these printers.

Note that if you already have one of these Winprinters, there are roundabout ways to get Linux to print to one, but they're rather awkward. See Section 12 in this document for more discussion of Windows−only printers.

5.1. Postscript

As for what printers *do* work with Linux, the best choice is to buy a printer with native PostScript support *in firmware*. Nearly all Unix software that produces printable output produces it in PostScript, so obviously it'd be nice to get a printer that supports PostScript directly. Unfortunately, PostScript support is scarce outside the laser printer domain, and is sometimes a costly add−on.

Unix software, and the publishing industry in general, have standardized upon Postscript as the printer control language of choice. This happened for several reasons:

Timing

> Postscript arrived as part of the Apple Laserwriter, a perfect companion to the Macintosh, the system largely responsible for the desktop publishing revolution of the 80s.

It's device−independent

> Postscript programs can be run to generate output on a pixel screen, a vector screen, a fax machine, or almost any sort of printer mechanism, without the original program needing to be changed. Postscript output will look the same on any Postscript device, at least within the limits of the device's capabilities. Before the creation of PDF, people exchanged complex documents online as Postscript files. The only reason this standard didn't "stick" was because Windows machines didn't usually include a Postscript previewer, so Adobe specified hyperlinks and compression for Postscript, called the result PDF, distributed previewers for it, and invented a market for their "distiller" tools (the functionality of which is also provided by ghostscript's ps2pdf and pdf2ps programs).

It's a real programming language

Postscript is a complete programming language; you can write software to do most anything in it. This is mostly useful for defining subroutines at the start of your program to reproduce complex things over and over throughout your document, like a logo or a big "DRAFT" in the background. But there's no reason you couldn't compute À in a Postscript program.

It's open

Postscript is fully specified in a publically available series of books (which you can find at any good bookstore). Although Adobe invented it and provides the dominant commercial implementation, other vendors like Aladdin produce independently coded implementations as well.

5.2. Non−Postscript

Failing the (larger) budget necessary to buy a Postscript printer, you can use any printer supported by Ghostscript, the free Postscript interpreter used in lieu of actual printer Postscript support. Note that most Linux distributions can only ship a somewhat outdated version of Ghostscript due to the license. Fortunately, there is usually a prepackaged up to date Ghostscript made available in each distribution's contrib area.

Adobe now has a new printer language called "PrintGear". I think it's a greatly simplified binary format language with some Postscript heritage but no Postscript compatibility. And I haven't heard of Ghostscript supporting it. But some PrintGear printers seem to support another language like PCL, and these printers will work with Linux (iff the PCL is implemented in the printer and not in a Windows driver).

Similarly, Adobe offers a host−based Postscript implementation called PressReady. This works much like Ghostscript does to provide Postscript support for a non−Postscript printer, but has the disadvantage that it runs only on Windows.

5.3. What printers work?

If you want to buy a printer, you can look in several places to see if it will work. The cooperatively maintained Printing HOWTO printer database aims to be a comprehensive listing of the state of Linux printer support. A summary of it is below; be sure to check online for more details and information on what driver(s) to use.

Ghostscript's printer compatibility page has a list of some working printers, as well as links to other pages.

Dejanews contains hundreds of "it works" and "it doesn't work" testimonials. Try all three, and when you're done, check that your printer is present and correct in the database, so that it will be listed properly in this document in the future.

If you're lazy, I keep a short list of suggested printers on my website. These center around color inkjets and low−cost laser devices; fully compatible mid−range and high−end devices are much easier to find. You can even help support this document and the website by buying from buy.com or outpost.com through me.

5.3.1. Printer compatibility list

This section is a summary of the online database. The online version includes device specifications, notes, driver information, user−maintained documentation, manufacturer web pages, and interface scripts for using drivers with several print spooling systems (including LPR, LPRng, PDQ, and CUPS). The online version of this list is also interactive; people can and do add printers all the time, so be sure to check it as well. Finally, if your printer isn't listed, add it!

Note that this listing is not gospel; people sometimes add incorrect information, which I eventually weed out. Entries I have not sanity−checked are marked with an asterisk (*). Verify from Dejanews that a printer works for someone before buying it based on this list. If you can find no information in Dejanews, mail me and I'll put you in contact with the person who added the printer.

Printers here are categorized into three types:

Perfectly

> Perfect printers work perfectly − you can print to the full ability of the printer, including color, full resolution, etc. In a few cases printers with undocumented "resolution enhancement" modes that don't work are listed as perfect; generally the difference in print quality is small enough that it isn't worth worrying about.

Mostly

> You can print fine, but there may be minor limitations of one sort or another in either printing or other features.

Partially

> You can print, but maybe not in color, or only at a poor resolution. See the online listing for information on the limitation.

Paperweight

> You can't print a darned thing; typically this will be due to lack of a driver and/or documentation on how to write one. Paperweights occasionally get "promoted", either when someone discovers that an existing driver works, or when someone creates a new driver, but you shouldn't count on this happening.

In all cases, since this information is provided by dozens of people, none of it is guaranteed to be correct; entries with an asterisk (*) are particularly suspect. The facts, however, should be easy to corroborate from the driver web pages and manufacturer web sites.

And without further ado, here is the printer compatibility list:

Table 1. Linux Printer Support

Manufacturer	Perfectly	Mostly	Partially	Paperweight

Alps			MD−1000 MD−1300 MD−2000 MD−4000 MD−5000	
Apollo		P−1200		
Apple	12/640ps Dot Matrix ImageWriter ImageWriter LQ LaserWriter 16/600* LaserWriter IINTX* LaserWriter IIg LaserWriter Select 360	Color StyleWriter 1500 Color StyleWriter 2200 Color StyleWriter 2400 Color StyleWriter 2500 LaserWriter NT StyleWriter 1200 StyleWriter I StyleWriter II		
Avery	Personal Label Printer+	Personal Label Printer		
Brother	HL−4Ve HL−8 HL−10V HL−10h HL−630 HL−660 HL−720 HL−730 HL−760 HL−820 HL−1020 HL−1040 HL−1070* HL−1250 HL−1260 HL−1270N HL−1660e	HJ−400 HL−1050 HL−1060 HL−1240	DCP−1200 HL−1030* MC−3000 MFC 7150C MFC−4350 MFC−6550MC MFC−8300 MFC−9100c MFC−9500 MFC−9600	4550* MP−21C

	HL−2060			
C.Itoh	M8510			
CalComp	Artisan 1023 penplotter*			
Canon	BJ−5 BJ−10e BJ−20 BJ−200 BJ−330 BJC−70 BJC−210 BJC−250 BJC−600 BJC−610 BJC−620 BJC−800 BJC−4000 BJC−4100 BJC−4200 BJC−4300* BJC−4400* GP 335 GP 405 LBP−4+ LBP−4U LBP−8A1 LBP−430 LBP−1260 LBP−1760 LIPS−III	BJC−80 BJC−240 BJC−1000* BJC−2000* BJC−2100 BJC−3000 BJC−4310SP BJC−7004* LBP−4sx	BJ−300 BJC−210SP BJC−4550 BJC−6000* BJC−6100 BJC−7000* BJC−7100* BJC−8200 MultiPASS C2500* MultiPASS C3000 MultiPASS C3500* MultiPASS C5000* MultiPASS C5500	BJC−5000 BJC−5100 BJC−6500 BJC−8000 LBP−460* LBP−600 LBP−660* Multipass L6000*
Citizen	ProJet II* ProJet IIc	printiva600C		
Compaq			IJ300 IJ750 IJ900	

DEC	DECWriter 500i* DECwriter 110i* DECwriter 520ic* LA50* LA75* LA75 Plus* LN03* LN07*	LJ250* LN17	1800*		
Dymo−CoStar	ASCII 250* ASCII+* EL40* EL60* LabelWriter II* LabelWriter XL* LabelWriter XL+* SE250* SE250+* Turbo*				
Epson	Action Laser II ActionLaser 1100* ActionPrinter 3250 Dot Matrix L−1000* LP 8000 LQ−24 LQ−500 LQ−570+ LQ−850 LQ−2550 LX−1050 SQ 1170 Stylus Color Stylus Color 400 Stylus Color 440 Stylus Color 460 Stylus Color 480 Stylus Color 500 Stylus Color 600 Stylus Color 640	EPL−5700 Stylus Color 300 Stylus Color 670* Stylus Color 850 Stylus Color II Stylus Color IIs Stylus Pro XL	Stylus Photo 2000P	EPL−5700L	

	Stylus Color 660 Stylus Color 740 Stylus Color 760 Stylus Color 800 Stylus Color 860 Stylus Color 880 Stylus Color 900 Stylus Color 980 Stylus Color 1160 Stylus Color 1500 Stylus Color 1520 Stylus Color 3000 Stylus Color I Stylus Color PRO Stylus Photo Stylus Photo 700 Stylus Photo 720 Stylus Photo 750 Stylus Photo 870 Stylus Photo 1200 Stylus Photo 1270 Stylus Photo EX			
Fujitsu	1200* 2400* 3400* PrintPartner 10V* PrintPartner 16DV* PrintPartner 20W* PrintPartner 8000*			
HP	2000C 2500C Color LaserJet 4500 DesignJet 3500CP DeskJet DeskJet 400 DeskJet 420C DeskJet 500 DeskJet 500C DeskJet 510	Color LaserJet 5 DesignJet 230* DesignJet 350C DesignJet 650C* Designjet 750 C Plus* DeskJet 310 DeskJet 610C DeskJet 610CL DeskJet 612C	Color LaserJet 5000 DeskJet 320 DeskJet 340C DeskJet 890C DeskJet 930C DeskJet 1000C LaserJet 1100A OfficeJet 500* OfficeJet 600*	LaserJet 3100* LaserJet 3150

	DeskJet 520	DeskJet 660C	OfficeJet 625*	
	DeskJet 540	DeskJet 670C	OfficeJet 635*	
	DeskJet 550C	DeskJet 672C	OfficeJet 710*	
	DeskJet 560C	DeskJet 682C	OfficeJet Pro 1170C*	
	DeskJet 600	DeskJet 690C	OfficeJet Pro 1175C*	
	DeskJet 1200C	DeskJet 692C	OfficeJet R45*	
	DeskJet 1600C	DeskJet 694C	OfficeJet R60	
	DeskJet 1600CM	DeskJet 697C	PhotoSmart P1000	
	LaserJet	DeskJet 710C*	PhotoSmart P1100*	
	LaserJet 2 w/PS*	DeskJet 712C		
	LaserJet 2D	DeskJet 720C*		
	LaserJet 2P	DeskJet 722C*		
	LaserJet 2P Plus	DeskJet 810C		
	LaserJet 3	DeskJet 812C		
	LaserJet 3D	DeskJet 815C*		
	LaserJet 3P w/PS	DeskJet 820C		
	LaserJet 4 Plus	DeskJet 832C		
	LaserJet 4L	DeskJet 840C		
	LaserJet 4M	DeskJet 842C*		
	LaserJet 4ML*	DeskJet 850C		
	LaserJet 4P	DeskJet 855C		
	LaserJet 5	DeskJet 870C		
	LaserJet 5L*	DeskJet 870Cse*		
	LaserJet 5M*	DeskJet 870Cxi		
	LaserJet 5MP*	DeskJet 880C		
	LaserJet 5P*	DeskJet 882C		
	LaserJet 6	DeskJet 895C		
	LaserJet 6L*	DeskJet 895Cxi*		
	LaserJet 6MP*	DeskJet 932C		
	LaserJet 1100	DeskJet 950C*		
	LaserJet 2100	DeskJet 970C		
	LaserJet 2100M	DeskJet 970Cse		
	LaserJet 4050N	DeskJet 1100C		
	LaserJet 5000	DeskJet 1120C		
	LaserJet 8000	DeskJet 1220C		
	LaserJet 8100	LaserJet 2		
	LaserJet Plus*	LaserJet 6P		
	LaserJet Series II*	LaserJet 4000		
	Mopier 240*	PSC 500*		
	Mopier 320*			
	PaintJet*			

	PaintJet XL* PaintJet XL300* ThinkJet*			
Heidelberg	Digimaster 9110*			
Hitachi	DDP 70 (with MicroPress)*			
IBM	3853 JetPrinter* 4019* 4029 10P* 4303 Network Color Printer* Execjet 4072* Page Printer 3112* ProPrinterII*			
Imagen	ImPress*			
Infotec	infotec 4651 MF*			
Kodak	DigiSource 9110* IS 70 CPII*			
Kyocera	F−3300 FS−600* FS−600 (KPDL−2)* FS−680* FS−800* FS−1200* FS−1700+* FS−1750* FS−3750* FS−5900C* P−2000*	F−800T* FS−3500*		
Lexmark	4039 10plus Optra Color 40 Optra Color 45	1020 Business 3000	1000 1100 2030	1020 2050 Winwriter 100*

	Optra Color 1200 Optra Color 1275 Optra E* Optra E+* Optra E310 Optra E312 Optra Ep* Optra K 1220* Optra R+* Optra S 1250* Optra S 1855* Optra Se 3455* Optra W810 Valuewriter 300* Z32		2070 3200 5000 5700 7000 7200 Winwriter 400* Z11* Z51	Winwriter 150c* Winwriter 200* Z22 Z52*
Minolta	PagePro 6* PagePro 6e* PagePro 6ex* PagePro 8*		PagePro 8L*	PagePro 6L
Mitsubishi	CP50 Color Printer*			
NEC	P2X* PinWriter P6* PinWriter P6 plus* PinWriter P7* PinWriter P7 plus* PinWriter P60* PinWriter P70* SilentWriter LC 890* Silentwriter2 S60P* Silentwriter2 model 290* SuperScript 660i* SuperScript 1800	Silentwriter 95f*	SuperScript 100C* SuperScript 150C* SuperScript 650C* SuperScript 750C* SuperScript 860* SuperScript 870* SuperScript 1260*	SuperScript 610plus* SuperScript 660* SuperScript 660plus*
Oce	3165*			
Okidata	ML 380* OL 410e	Microline 182 OL 400w*	Microline 192+ Okipage 6w*	Okijet 2010

	OL 600e* OL 610e/PS OL 800 OL 810e/PS OL400ex OL810ex OL820* OL830Plus Okipage 6e Okipage 6ex* Okipage 8c Okipage 8p Okipage 10e Okipage 12i Okipage 20DXn	OL 610e/S OkiPage 4w+* OkiPage 8w Lite* OkiPage 8z* Okijet 2500* Okipage 4w* Okipage 8w* Super 6e		
Olivetti	JP350S* JP450* JP470* PG 306*			
PCPI	1030*			
Panasonic	KX−P1123* KX−P1124* KX−P1150* KX−P1180i* KX−P2023* KX−P2135* KX−P2150* KX−P4410* KX−P4450* KX−P5400* KX−P8420* KX−P8475* KX−PS600* kx−p1624*	KX−P2123* KX−P6150*	KX−P6500*	KX−P6100* KX−P6300 GDI* KX−P8410*
Printrex			820 DL*	

QMS	2425 Turbo EX* LPK−100*	magicolor 2+* ps−810*		magicolor 2
Raven		LP−410		
Ricoh	4081* 4801* 6000* Aficio 220* Aficio AP2000	Aficio 401*		Aficio Color 2206* Afico FX10*
Samsung	ML−85* ML−4600* ML−5000a* ML−6000/6100* ML−7000/7000P/7000N* ML−7050* QL−5100A* QL−6050* SI−630A*	ML−85G QL−85G		ML−5050G* SF/MSYS/MJ−4700/4800/4500C*
Seiko	SpeedJET 200*	SLP* SLP 120* SLP 220* SLP EZ30* SLP Plus* SLP Pro*		
Sharp	AR−161*			
Star	LC24−100* LS−04 NL−10*	LC 90* LC24−200* StarJet 48*		WinType 4000*
Tally	MT908*			
Tektronix	3693d color printer, 8−bit mode* 4693d color printer, 2−bit mode*			

	4693d color printer, 4−bit mode* 4695* 4696* 4697* Phaser 780 Phaser 850* Phaser IISX* Phaser PX*			
Xerox	2700 XES 3700 XES 4045 XES* DocuPrint 4508 DocuPrint C20 DocuPrint C55* DocuPrint N17 DocuPrint N32* Document Centre 400*	DocuPrint C6* DocuPrint P8e DocuPrint P12* Docuprint C6* Docuprint C8* XJ6C*	Document Homecentre WorkCentre 450cp* WorkCentre 470cx* XJ8C*	DocuPrint P8* Work Centre XK35c WorkCenter XE90fx* WorkCentre XD120f* WorkCentre XE80 workcentre 385*

* This entry has not been sanity−checked by me.

5.4. How to buy a printer

It's a bit difficult to select a printer these days; there are many models to choose from. Here are some shopping tips:

Cost

You get what you pay for. Most printers under $200−300 will print reasonably well, but printing costs a lot per page. For some printers, it only takes one or two cartridges to add up to the cost of a new printer! Similarly, the cheapest printers won't last very long. The least expensive printers, for example, have a MTBF of about three *months*; obviously these are poorly suited for heavy use.

Inkjets

Inkjet printheads will clog irreparably over time, so the ability to replace the head somehow is a feature. Inkjet printheads are expensive, with integrated head/ink cartridges costing ten times (!) what ink−only cartridges go for, so the ability to replace the head only when needed is a feature. Epson Styluses tend to have fixed heads, and HP DeskJets tend to have heads integrated into the cartridges. Canons have three−part cartridges with independently replaceable ink tanks; I like this design. OTOH, the HP cartridges aren't enormously more expensive, and HP makes a better overall line; Canon is often the third choice from the print quality standpoint; and Epson Styluses are the best supported under Linux at the moment. You just can't win.

Lasers

Laser printers consume a drum and toner, plus a little toner wiping bar. The cheapest designs include toner and drum together in a big cartridge; these designs cost the most to run. The best designs for large volume take plain toner powder or at least separate toner cartridges and drums.

Photography

The best color photograph output is from continuous tone printers which use a silver halide plus lasers approach to produce surprise! actual photographs. Since these printers cost tens of thousands to buy, Ofoto.com offers inexpensive print−by−print jobs. The results are stunning; even the best inkjets don't compare.

The best affordable photo prints come from the dye−sublimation devices like some members of the Alps series (thermal transfer of dry ink or dye sublimation). Unfortunately they have poor Linux support (the one report I have speaks of banding and grainy pictures), and even then it's unclear if the dye−sub option is supported.

The more common photo−specialized inkjets usually feature 6 color CMYKcm printing or even a 7 color CMYKcmy process. All photo−specialized printers are expensive to run; either you always run out of blue and have to replace the whole cartridge, or the individual color refills for your high−end photo printer cost an arm and a leg. Special papers cost a bundle, too; you can expect top−quality photo inkjet output to run over a US dollar per page. See also the section on printing photographs later in this document, and the sections on color tuning (such as it is) in Ghostscript.

Speed

Speed is proportional to processing power, bandwidth, and generally printer cost. The fastest printers will be networked Postscript printers with powerful internal processors. Consumer−grade printers will depend partly on Ghostscript's rendering speed, which you can affect by having a reasonably well−powered machine; full pages of color, in particular, can consume large amounts of host memory. As long as you actually *have* that memory, things should work out fine.

Forms

If you want to print on multicopy forms, then you need an impact printer; many companies still make dot matrix printers, most of which emulate traditional Epson models and thus work fine.

Labels

There are two supported lines of label printer; look for the Dymo−Costar and the Seiko SLP models. Other models may or may not work. Avery also makes various sizes of stick−on labels in 8.5x11 format that you can run through a regular printer.

Plotting

Big drafting formats are usually supported these days by monster inkjets; HP is a popular choice. Mid−sized (11x17) inkjets are also commonly used for smaller prints. Much plotting of this sort is done with the languages RTL, HP−GL, and HP−GL/2, all of which are simple HP proprietary vector languages usually generated directly by application software.

5.4.1. What do I have?

I own an HP Deskjet 500, a Lexmark Optra 40, and a Canon BJC−4100. All work perfectly: the HP and Canon are older models, well supported by Ghostscript; and the Optra is a more modern color inkjet with full Postscript and PCL 5 support (!).

I also own a Hawking Technology 10/100 Ethernet print server (model 7117, actually made by Zero One Technologies in Taiwan); this makes it possible to put the printer anywhere with power and a network jack, instead of just near a computer. It's a little dongle that attaches to the printer's parallel port and has an Ethernet jack on the other side. The only flaw with this is that it doesn't allow bidirectional communication, so I can't arrange to be sent email when the ink is low.

6. Spooling software

Until recently, the choice for Linux users was simple − everyone ran the same old lpd lifted mostly verbatim out of BSD's Net−2 code. Even today, most vendors ship this software. But this is beginning to change. SVR4−like systems including Sun's Solaris come with a completely different print spooling package, centered around lpsched.

Today, there are a number of good systems to chose from. I describe them all below; read the descriptions and make your own choice. PDQ is the simplest modern system with a GUI; it is suitable for both basic home users and (in a hybrid pdq/lprng setup) people in many larger environments. For business environments with mainly networked Postscript printers, a front−end program like GPR with LPRng is a good alternative; it handles PPD options directly and has a slightly nicer interface. In other cases CUPS is a good option; it too has excellent Postscript printer support, and offers IPP support, a web interface, and a number of other features.

6.1. LPD

LPD, the original BSD Unix Line Printer Daemon, has been the standard on Unix for years. It is available for every style of Unix, and offers a rather minimal feature set derived from the needs of timesharing−era computing. Despite this somewhat peculiar history, it is still useful today as a basic print spooler. To be really useful with modern printer, a good deal of extra work is needed in the form of companion filter scripts and front−end programs. But these exist, and it does all work.

LPD is also the name given to the network printing protocol by RFC 1179. This network protocol is spoken not only by the LPD daemon itself, but by essentially every networked print server, networked printer, and every other print spooler out there; LPD is the least common denominator of standards−based network printing.

LPRng (see Section 6.3) is a far better implementation of the basic LPD design than the regular one; if you must use LPD, consider using LPRng instead. There is far less voodoo involved in making it do what you want, and what voodoo there is is well documented.

There are a large number of LPD sources floating around in the world. Arguably, some strain of BSD Unix is probably the official owner, but everyone implements changes willy−nilly, and they all cross−pollinate in unknown ways, such that it is difficult to say with certainty exactly which LPD you might have. Of the readily available LPDs, VA Linux offers one with a few minor modifications that make the user interface much more flexible. The SourceForge LPD supports command−line option specification with a `−o` flag; options are then passed through to filters. This is similar to the features offered by a number of traditional Unix vendors, and similar to (although incompatible with) LPRng's `−z` option mechanism.

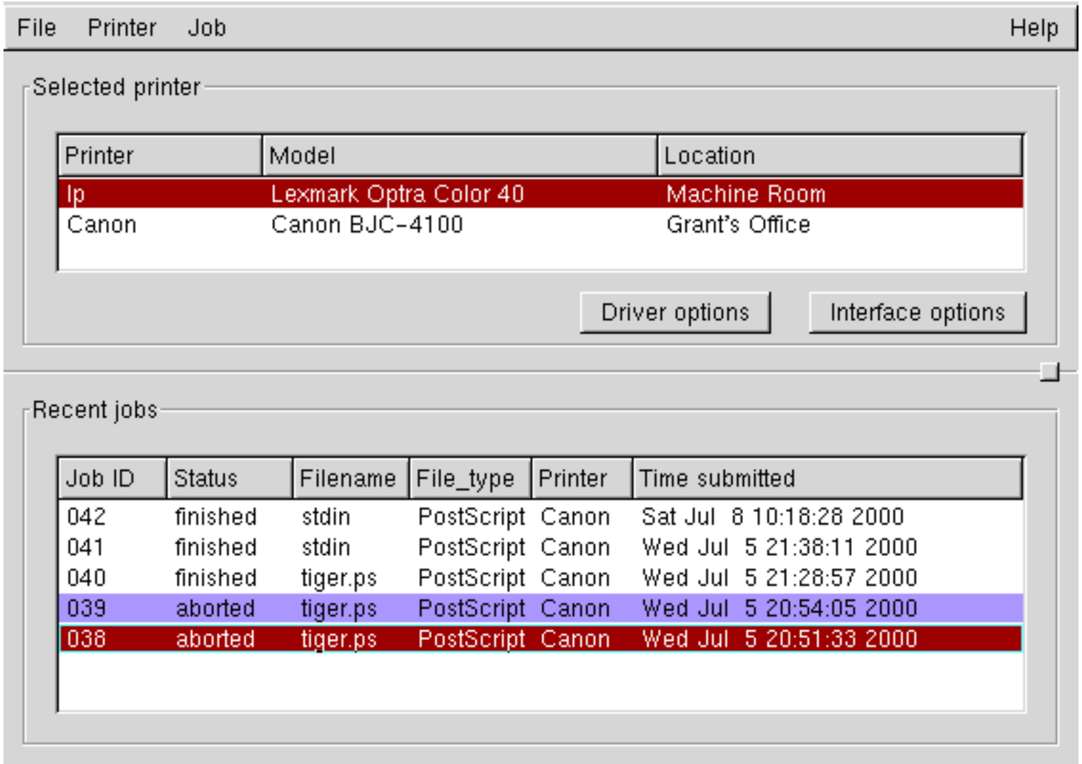
6.1.1. LPD front−ends

If you go with LPD, the best way to use it is via a front−end. There are several to chose from; GPR (see Section 3.3) and XPDQ (see Section 6.2) are perhaps the two best. Others exist; tell me about them.

6.2. PDQ

PDQ is a non–daemon–centric print system which has a built–in, and sensible, driver configuration syntax. This includes the ability to declare printing options, and a GUI or command line tool for users to specify these options with; users get a nice dialog box in which to specify resolution, duplexing, paper type, etc (see Figure 7).

Figure 6. XPDQ Main Window



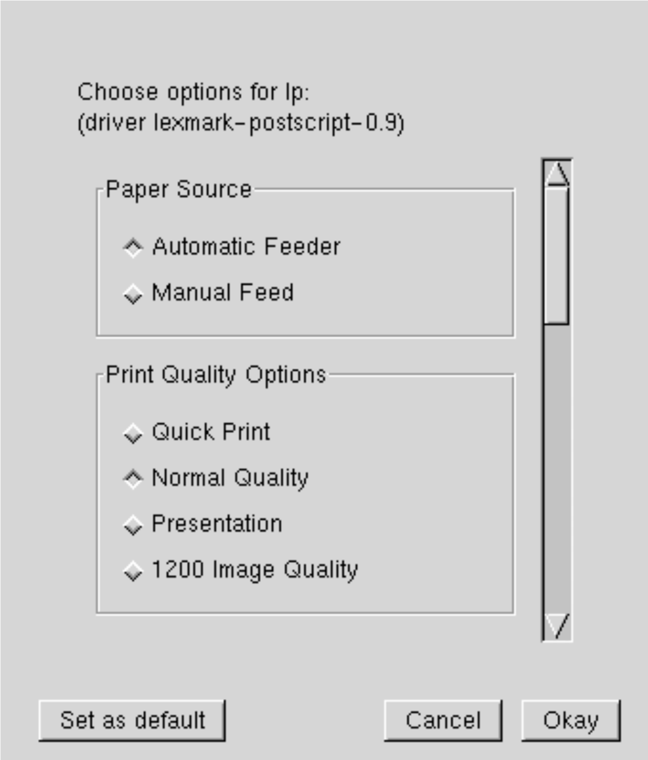
Running all of the filters as the user has a number of advantages: the security problems possible from Postscript are mostly gone, multi–file LaTeX jobs can be printed effectively as dvi files, and so forth.

This is what I now use; I've written driver spec files for my printers, and there are several included with the distribution, so there are plenty of examples to base yours on. I've also written a few tools to automate driver spec generation to help the rest of you.

PDQ is not without flaws: most notably it processes the entire job before sending it to the printer. This means that, for large jobs, PDQ may simply be impractical you can end up with hundreds of megs being copied back and forth on your disk. Even worse, for slow drivers like the better quality inkjet drivers, the job will not start printing until Ghostscript and the driver have finished processing. This may be many minutes after submission.

If you have many users, many printers, or anything else complex going on, I recommend using PDQ as a front–end to LPD–protocol based network printing (you can print via the lpd protocol to the local machine). In most such situations, rather than using the traditional BSD lpd as the back–end, I recommend LPRng:

Figure 7. XPDQ Driver Options Window



6.3. LPRng

Some Linux vendors (including Caldera) provide LPRng, a far less ancient LPD print spooling implementation. LPRng is far easier to administer for large installations (read: more than one printer, any serial printers, or any peculiar non−lpd network printers) and has a less frightfully haphazard codebase than does stock lpd. It can even honestly claim to be secure − there are no SUID binaries, and it supports authentication via PGP or Kerberos.

LPRng also includes some example setups for common network printers − HP LaserJets, mainly, that include some accounting abilities. If you'd like more information on LPRng, check out the LPRng Web Page. LPRng uses more or less the same basic filter model as does BSD lpd, so the LPD support offered by my website applies to LPRng as well. This can help you effectively use free software drivers for many printers.

LPRng is distributed under either the GPL or an Artistic license.

6.4. PPR

PPR is a Postscript−centric spooler which includes a rudimentary Postscript parsing ability from which it derives several nice features. It includes good accounting capabilities, good support for Appletalk, SMB, and LPD clients, and much better error handling than lpd. PPR, like every other spooler here, can call Ghostscript to handle non−Postscript printers.

I only recently found out about PPR; I don't know of anyone who has tried it. It was written by, and is in use at, Trinity College. The license is BSD−style; free for all use but credit is due.

According to the documentation, it's somewhat experimental. Malformed Postscript jobs won't print; instead they bounce, and it's up to the user to fix the Postscript. This may make it unsuitable for some environments, although most users generate Postscript with a small handful of well−characterized Postscript generators, so it probably wouldn't be that big an issue.

6.5. CUPS

One interesting newcomer on the scene is CUPS, an implementation of the Internet Printing Protocol (IPP), an HTTP−like RFC standard replacement protocol for the venerable (and klunky) LPD protocol. The implementation of CUPS has been driven by Michael Sweet of Easy Software Products; CUPS is distributed under the GPL.

I've finally done some work with CUPS, and it does work as advertised. There are a number of very good features in it, including sensible option handling; web, gui, and command−line interfaces; and a mime−based filtering system with strong support for Postscript. Since it is so new, however, it does have a number of quirks, and it is hard to recommend for large or secure installations at this time (as of version 1.1). It is a fine solution, however, for smaller installations or especially larger installatons with trusted users.

Like other systems, CUPS can be used with most existing drivers. Unfortunately, it's a bit tricky to configure an arbitrary driver for use with CUPS at least if you want all the options to work so it's best to find a preexisting PPD file and filter script to make your driver go. There are at least four sets of drivers which you can use with CUPS:

CUPS−O−Matic

> My web−based CUPS−O−Matic system can generate a suitable PPD for use with any printer driver that has full details entered in the Linux Printing Database. The PPD gets used together with a backend script named **cupsomatic**. CUPS−O−Matic uses free software drivers. At the moment I am concentrating on correctness rather than completeness, so rather few drivers are in fact supported. This will change over time.

CUPS Drivers and KUPS

> The CUPS Drivers project is accumulating PPD files useable with either Postscript printers or the backend filter **ps2gs2raw**. These PPD files use free software drivers. KUPS is a companion setup program.

Postscript PPDs

> CUPS can use vendor−supplied PPD files for Postscript printers directly. Often these come with the Windows drivers for a printer, or can be found on the printer vendor's website. Adobe also distributes PPD files for many Postscript printers.

ESP Print Pro

> Easy Software Products, Inc. sells CUPS bundled with a collection of proprietary drivers. Although they are not free software, they do drive many common printers. The bundle is somewhat expensive

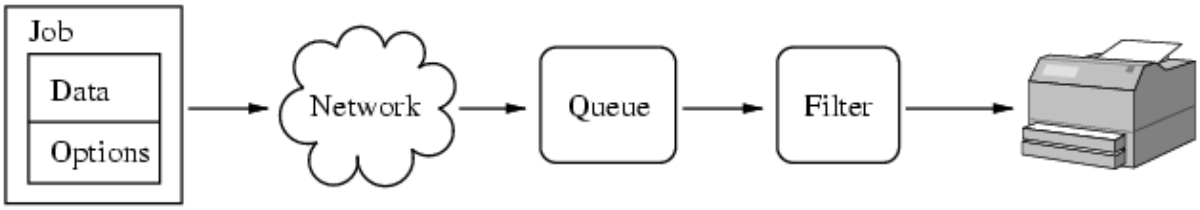
measured against the price of a single supported printer, but it certainly has a place. These drivers are reputedly not terribly good, but they are somewhat comprehensive, and even mediocre quality is preferable to a paperweight.

The third−party program **XPP** (see Figure 4) offers a very nice graphical interface to the user functionality of CUPS, including an marvelous interface to print−time options (shown in Figure 5). For information on using XPP, see Section 3.3.2.

7. How it all works

In order to get printing working well, you need to understand how your spooling software works. All systems work in essentially the same way, although the exact order might vary a bit, and some systems skip a step or two:

Figure 8. Spooling Illustration



1. The user submits a job along with his selection of options. The job data is usually, but not always, Postscript.
2. The spooling system copies the job and the options over the network in the general direction of the printer.
3. The spooling system waits for the printer to be available.
4. The spooling system applies the user's selected options to the job, and translates the job data into the printer's native language, which is usually not Postscript. This step is called *filtering*; most of the work in setting things up lies in getting the proper filtering to happen.
5. The job is done. The spooling system will usually do assorted cleanup things at this point. If there was an error along the way, the spooler will usually notify the user somehow (for example, by email).

7.1. PDQ

Pdq stands for "Print, Don't Queue", and the way it works reflects this design. The following sequence of events happens when you use PDQ to print:

- You run pdq or xpdq, specifying a file.
- You specify a printer.
- You specify the settings for the various options and arguments defined in the printer's PDQ driver file (duplex, copies, print quality, and so forth).
- PDQ analyzes the contents of what you printed, and follows the instructions in the PDQ driver file which tell it how to process your data for this printer with your options.
- PDQ sends the processed data to the printer according to the interface defined for that printer (straight to `/dev/lp0`, or to an LPD daemon on the network, over the network to an Apple or Microsoft system, or even to a fax machine).
- If PDQ can't send the data to the printer right away, it spawns a background process to wait and try again until it succeeds or hits a time limit.

At all times during this process, and afterwards, the state of each print job can be seen and inspected using xpdq. Jobs that failed are shown in red and can be resent.

7.2. LPD

Lpd stands for Line Printer Daemon, and refers in different contexts to both the daemon and the whole collection of programs which run print spooling. These are:

lpd

The spooling daemon. One of these runs to control everything on a machine, AND one is run per printer while the printer is printing.

lpr

The user spooling command. Lpr contacts lpd and injects a new print job into the spool.

lpq

Lists the jobs in a print queue.

lpc

The Lpd system control command. With lpc you can stop, start, reorder, etc, the print queues.

lprm

lprm removes a job from the print spool.

So how does it fit together? The following things happen:

1. At boot time, **lpd** is run. It waits for connections and manages printer queues.
2. A user submits a job with the **lpr** command or, alternatively, with an lpr front−end like GPR, PDQ, etc. **Lpr** contacts **lpd** over the network and submits both the user's data file (containing the print data) and a control file (containing user options).
3. When the printer becomes available, the main **lpd** spawns a child **lpd** to handle the print job.
4. The child **lpd** executes the appropriate filter(s) (as specified in the `if` attribute in `/etc/printcap`) for this job and sends the resulting data on to the printer.

The lp system was originally designed when most printers were line printers − that is, people mostly printed plain ascii. By placing all sorts of magic in the `if` filter, modern printing needs can be met with **lpd** (well, more or less; many other systems do a better job).

There are many programs useful for writing LPD filters. Among them are:

gs

Ghostscript is a host−based Postscript interpreter (aka a Raster Image Processor or RIP). It accepts Postscript and produces output in various printer languages or a number of graphics formats. Ghostscript is covered in Section 10.

ppdfilt

ppdfilt is a standalone version of a CUPS component. It filters Postscript, executing a few basic transformations on it (n−up printing, multiple copies, etc) and adding in user option statements according to a Postscript Printer Definition (PPD) file usually included with Postscript printers.

ppdfilt is best used together with an option−accepting LPD system (like the VA Linux LPD, or LPRng) and a filter script which parses user−provided options into the equivalent **ppdfilt** command. VA Linux and HP provide a modified rhs−printfilters package which does exactly this; it produces nice results if you have a Postscript printer. See Section 8.2.2 for information on this system.

ps2ps

ps2ps is a utility script included with Ghostscript. It filters Postscript into more streamlined Postscript, possibly at a lower Language Level. This is useful if you have an older Postscript printer; most modern software produces modern Postscript.

mpage

mpage is a utility which accepts text or Postscript, and generates n−up output that is, output with several page images on each piece of paper. There are actually several programs which do this, including **enscript**, **nenscript**, and **a2ps**.

a2ps

a2ps, aka any−to−ps, is a program which accepts a variety of file types and converts them to Postscript for printing.

---

8. How to set things up

For common configurations, you can probably ignore this section entirely − instead, you should jump straight to Section 9 below, or better yet, your vendor's documentation. Most Linux distributions supply one or more "idiot−proof" tools to do everything described here for common printers.

If your vendor's tool doesn't work out for you, or you'd like the ability to interactively control printing options when you print, then you should use some other system. PDQ is a good choice; it provides very good functionality and is easy to setup. APS Filter is another good system; it configures LPD queues and filters very easily on most any sort of Unix system.

You can also use the printing system interfaces from the Linux Printing Website to connect many free drivers into several spooling systems. Once this project is complete, these interfaces will offer the best functionality: all styles of free software drivers are supported, user−settable options are available, and most common spooling systems are supported.

8.1. Configuring PDQ

PDQ can be configured by either the superuser or by a joeuser. Root's changes are made to `/etc/printrc`, and affect everyone, while joeuser can only modify his personal `.printrc`. Everything applies to both types of configuration.

If PDQ is not available prepackaged for your distribution, you should obtain the source distribution from the PDQ web page and compile it yourself. It is an easy compile, but you must first be sure to have installed the various GTK development library packages, the C library development package, the gcc compiler, make, and possibly a few other development things.

8.1.1. Drivers and Interfaces

PDQ lets users select a printer to print to. A printer is defined in PDQ as the combination of a "driver" and an "interface". Both drivers and interfaces are, in fact, merely snippets of text in the PDQ configuration file.

A PDQ interface says everything about how to ship data out to a printer. The most common interfaces, which are predefined in the PDQ distribution's example printrc file, are:

local−port

> A local port interface speaks to a parallel or serial port on the machine PDQ is running on. Using this interface, PDQ can print directly to your parallel port. Note that if you have a multiuser system this can cause confusion, and if you have a network the local−port interface will only apply to one system. In those cases, you can define a raw unfiltered lpd queue for the port and print to the system's lpd daemon exactly the same way from all systems and accounts without any troubles. This interface has a device name argument; the typical value would be `/dev/lp0`.

bsd−lpd

> A bsd lpd interface speaks over the network to an LPD daemon or LPD−speaking networked printer.

PDQ supports job submission, cancellation, and queries to LPD interfaces. This interface has hostname and queuename arguments.

appletalk

The appletalk interface allows you to print to printers over the Appletalk network; if you have a printer plugged into your Mac this is the way to go. This interface needs to have the Netatalk package installed to work.

A PDQ driver says everything about how to massage print data into a format that a particular printer can handle. For Postscript printers, this will include conversion from ascii into Postscript; for non−Postscript printers this will include conversion from Postscript into the printer's language with Ghostscript.

If one of PDQ's included driver specifications doesn't fit your printer, then read the section below on how to write your own.

8.1.2. Defining Printers

To define a printer in PDQ:

- First check that you've got suitable driver and interface declarations in the system or your personal printrc.
- If you want to define the printer in `/etc/printrc` (for all users), then su to root.
- Run **xpdq**, and select Printer−>Add printer. This "wizard" will walk you through the selection of a driver and interface.

That's really all there is to it; most of the work lies in finding or creating a suitable driver specification if you can't find one premade.

8.1.3. Creating a PDQ Driver Declaration

Here I'll walk through an example of how to make a PDQ driver declaration. Before you try that, though, there are several places to look for existing driver specs:

- PDQ itself comes with a collection of prewritten driver files.
- The Linux Printing Website's database includes a program called "PDQ−O−Matic" which will generate a PDQ specification from the information in the database. Assuming that the database contains the proper information for your printer and driver, this is the best path if you have a non−Postscript printer.
- I've written a tool called **ppdtopdq** which takes a Postscript Printer Definition file and converts it into a PDQ driver specification, with about 75% success. This is an option if you have a Postscript printer.

There are several places to look for the information needed to write your own PDQ driver:

- The PDQ driver specification syntax is quite rich, and is fully documented in the printrc(5) man page.
- The PDQ distribution includes a few example files. Look in particular at the Epson Stylus file, which demonstrates the structure of the definition for a Ghostscript−driven printer.
- The Printing HOWTO Database includes raw Linux driver information for over 600 printers. This

will tell you what options to give Ghostscript, or what extra program to run on the Ghostscript output.

If you have to create your own driver specification, or if you enhance one from the PDQ distribution or one of the PDQ driver generator programs mentioned above, please share your creation with the world! Send it to me (`gtaylor+pht@picante.com`), and I'll make sure that it gets found by future PDQ users with your type of printer.

Now, let's walk through the writing of a driver specification for a printer listed in the Printing HOWTO's database as working, but for which you can't find a PDQ driver spec. I'll use the Canon BJC−210 as the example printer.

First, we look at the database entry for this printer. Note that it is supported "perfectly", so we can expect to get comparable results (or better) to Windows users. The important information is in two places in the entry:

Notes

    The human−readable notes will often contain useful information. For some printers, there is a More Info link, which usually refers to a web page run by a user with this printer, or to the driver's home page.

Driver List

    Most printers have a list of drivers that are known to work. This is the most important part. You can follow the driver links to a driver−specific page, which will often have more information about how to execute the driver, as well as a link to the driver's web page, if it has one.

A PDQ driver spec has two logical functions: user interaction, and print job processing. These are represented in the file in three places:

Option Declarations

    These define what options the user can set, and declare PDQ variables for later parts of the driver to use.

Language Filters

    These process the print job from whatever format it arrived in (typically Postscript or ASCII) into a language the printer can understand (for example, PCL). Option values are available here, as well as in the output filter.

Output Filter

    This final filter bundles up the printer data regardless of input type; often printer options are set here.

Let's work on each of these for a Canon BJC−210:

8.1.3.1. Options

The driver list for this printer includes the bj200 and bjc600 drivers, both of which are Ghostscript style drivers. The notes suggest that we use the bj200 for black−and−white printing.

So, as far as the user is concerned, the BJC−210 supports one useful option: the user should pick color or black−and−white. Let's declare that as choice option called "MODE":

```
option {
  var = "MODE"
  desc = "Print Mode"
  # default_choice "Color"    # uncomment to default to color
  choice "BW" {
    # The value part assigns to the variable MODE whatever you
    # want. Here we'll assign the text that varies between the
    # two Ghostscript option sets for the two modes.
    value = "bj200"
    help = "Fast black printing with the black cartridge."
    desc = "Black-only"
  }
  choice "Color" {
    value = "bjc600"
    help = "Full-color printing."
    desc = "Color"
  }
}
```

With the above choice declarations, the user will see a Color or BW choice in the driver options dialog when he prints from xpdq. In the command−line pdq tool, he may specify `−oBW` or `−oColor`. The default can be set from xpdq, or declared above with the `default_choice` keyword.

8.1.3.2. Language Filtering

PDQ normally identifies its input with the **file(1)** command. For each type returned by **file** that you want to handle, you provide a `language_driver` clause. The clause consists mostly of a script to process the printjob language, in any (!) scripting language you wish (the default is the usual Bourne shell).

In our case, we want to print Postscript and ASCII on our BJC−210. This needs two language drivers: one to run Ghostscript for Postscript jobs, and one to add carriage returns to ASCII jobs:

```
# The first language_driver in the file that matches what file(1)
# says is what gets used.
language_driver ps {
  # file(1) returns "PostScript document text conforming at..."
  filetype_regx = "postscript"
  convert_exec = {
    gs -sDEVICE=$MODE -r360x360 \     # gs options from the database
       -q -dNOPAUSE -dBATCH -dSAFER \ # the "usual" Ghostscript options
       -sOutputFile=$OUTPUT $INPUT    # process INPUT into file OUTPUT

    # Those last two lines will often be the same for gs-supported
    # printers.  The gs... line, however, will be different for each
    # printer.
  }
}

# We declare text after postscript, because the command "file" will
# often describe a postscript file as text (which it is).
language_driver text {
  # No filetype_regx; we match the driver's name: "text"
  convert_exec = {#!/usr/bin/perl
     # a Perl program, just because we can!
```

```
     my ($in, $out) = ($ENV{'INPUT'}, $ENV{'OUTPUT'});
     open INPUT, "$in";
     open OUTPUT, ">$out";
     while(<INPUT>) {
        chomp;
        print OUTPUT, "$_\r\n";
     }
  }
}
```

That's it! While other printers may need output filtering (as described in the next section), the above clauses are it for the BJC−210. We just wrap them all up in a named `driver` clause:

```
driver canon-bjc210-0.1 {
  option {
    var = "MODE"
    desc = "Print Mode"
    # default_choice "Color"    # uncomment to default to color
    choice "BW" {
      # The value part assigns to the variable MODE whatever you
      # want. Here we'll assign the text that varies between the
      # two Ghostscript option sets for the two modes.
      value = "bj200"
      help = "Fast black printing with the black cartridge."
      desc = "Black-only"
    }
    choice "Color" {
      value = "bjc600"
      help = "Full-color printing."
      desc = "Color"
    }
  }

  # The first language_driver in the file that matches what file(1)
  # says is what gets used.
  language_driver ps {
    # file(1) returns "PostScript document text conforming at..."
    filetype_regx = "postscript"
    convert_exec = {
      gs -sDEVICE=$MODE -r360x360 \      # gs options from the database
         -q -dNOPAUSE -dBATCH -dSAFER \ # the "usual" Ghostscript options
         -sOutputFile=$OUTPUT $INPUT    # process INPUT into file OUTPUT

      # Those last two lines will often be the same for gs-supported
      # printers.  The gs... line, however, will be different for each
      # printer.
    }
  }

  # We declare text after postscript, because the command "file" will
  # often describe a postscript file as text (which it is).
  language_driver text {
    # No filetype_regx; we match the driver's name: "text"
    convert_exec = {#!/usr/bin/perl
       # a Perl program, just because we can!
       my ($in, $out) = ($ENV{'INPUT'}, $ENV{'OUTPUT'});
       open INPUT, "$in";
       open OUTPUT, ">$out";
       while(<INPUT>) {
          chomp;
          print OUTPUT, "$_\r\n";
```

```
      }
    }
  }
}
```

8.1.3.3. Output Filtering

If you want to prepend or append something to all printjobs, or do some sort of transformation on all the data of all types, then it belongs in the `filter_exec` clause. Our little Canon doesn't require such a clause, but just to have an example, here's a simple illustration showing how to support duplexing and resolution choice on a Laserjet or clone that speaks PJL:

```
driver generic-ljet4-with-duplex-0.1 {
  # First, two option clauses for the user-selectable things:
  option {
    var = "DUPLEX_MODE"
    desc = "Duplex Mode"
    default_choice = "SIMPLEX"
    choice "SIMPLEX" {
      value = "OFF"
      desc = "One-sided prints"
    }
    choice "DUPLEX" {
      value = "ON"
      desc = "Two-sided prints"
    }
  }

  option {
    var = "GS_RES"
    desc = "Resolution"
    default_choice = "DPI600"
    choice "DPI300" {
      value = "-r300x300"
      desc = "300 dpi"
    }
    choice "DPI600" {
      value = "-r600x600"
      desc = "600 dpi"
    }
  }

  # Now, we handle Postscript input with Ghostscript's ljet4 driver:
  language_driver ps {
    filetype_regx = "postscript"
    convert_exec = {
       gs -sDEVICE=ljet4 $GS_RES \
          -q -dNOPAUSE -dBATCH -dSAFER \
          -sOutputFile=$OUTPUT $INPUT
    }
  }

  # Finally, we wrap the job in PJL commands:
  filter_exec {
    # requires echo with escape code ability...
    echo -ne '\33%-12345X' > $OUTPUT

    echo "@PJL SET DUPLEX=$DUPLEX_MODE"    >> $OUTPUT
    # You can add additional @PJL commands like the above line here.
```

```
    # Be sure to always append (>>) to the output file!

    cat $INPUT >> $OUTPUT
    echo -ne '\33%-12345X' >> $OUTPUT
  }
}
```

8.2. Configuring LPD

Most Linux systems ship with LPD. This section describes a very basic setup for LPD; further sections detail the creation of complex filters and network configuration.

8.2.1. Basic LPD configuration

The minimal setup for lpd results in a system that can queue files and print them. It will not pay any attention to wether or not your printer will understand them, and will probably not let you produce attractive output. But we have to start somewhere.

To add a print queue to lpd, you must add an entry in `/etc/printcap`, and make the new spool directory under `/var/spool/lpd`.

An entry in `/etc/printcap` looks like:

```
# LOCAL djet500
lp|dj|deskjet:\
        :sd=/var/spool/lpd/dj:\
        :mx#0:\
        :lp=/dev/lp0:\
        :sh:
```

This defines a spool called *lp*, *dj*, or *deskjet*, spooled in the directory `/var/spool/lpd/dj`, with no per–job maximum size limit, which prints to the device `/dev/lp0`, and which does not have a banner page (with the name of the person who printed, etc) added to the front of the print job.

Go now and read the man page for `printcap`.

The above looks very simple, but there a catch – unless I send in files a DeskJet 500 can understand, this DeskJet will print strange things. For example, sending an ordinary Unix text file to a deskjet results in literally interpreted newlines, and gets me:

```
This is line one.
                 This is line two.
                                  This is line three.
```

ad nauseam. Printing a PostScript file to this spool would get a beautiful listing of the PostScript commands, printed out with this "staircase effect", but no useful output.

Clearly more is needed, and this is the purpose of filtering. The more observant of you who read the printcap man page might have noticed the spool attributes `if` and `of`. Well, `if`, or the input filter, is just what we need here.

If we write a small shell script called **filter** that adds carriage returns before newlines, the staircasing can be eliminated. So we have to add in an `if` line to our printcap entry above:

```
lp|dj|deskjet:\
        :sd=/var/spool/lpd/dj:\
        :mx#0:\
        :lp=/dev/lp0:\
        :if=/var/spool/lpd/dj/filter:\
        :sh:
```

A simple filter script might be:

```
#!perl
# The above line should really have the whole path to perl
# This script must be executable: chmod 755 filter
while(<STDIN>){chomp $_; print "$_\r\n";};
# You might also want to end with a form feed: print "\f";
```

If we were to do the above, we'd have a spool to which we could print regular Unix text files and get meaningful results. (Yes, there are four million better ways to write this filter, but few so illustrative. You are encouraged to do this more efficiently.)

The only remaining problem is that printing plain text is really not too hot − surely it would be better to be able to print PostScript and other formatted or graphic types of output. Well, yes, it would, and it's easy to do. The method is simply an extention of the above linefeed−fixing filter.

Such a filter is called a *magic* filter. It plays the same role as the language filters of PDQ. Don't bother writing one yourself unless you print strange things − there are a good many written for you already, and most have easy−to−use interactive configuration tools. You should simply select a suitable pre−written filter:

LPD−O−Matic

> Lpdomatic is a filter designed to use data from the Linux Printing printer database. It will soon support essentially all free software printer drivers, including regular Ghostscript drivers, Uniprint drivers, and the assorted filter programs floating around out there. It works with various strains of LPD, including stock BSD, LPRng, and the new VA Linux LPD, to allow option selection.

APS Filter

> apsfilter is a filter designed for use on a wide variety of Unices. It supports essentially all Ghostscript drivers. It, too, works with various strains of LPD, including stock BSD and LPRng. At the moment, this is probably the best third−party system around for non−PostScript printers.

RHS−Printfilters

> RHS−Printfilters is a filter system constructed by Red Hat. It shipped beginning, I think, in version 4 of Red Hat Linux, as the backend to the easy−to−use **printtool** GUI printer configuration tool. Other distributions, including Debian, now ship the rhs−printfilters/printool combo as a printing option. Thus this filter system is arguably the most widely deployed one.
>
> The rhs filter system is built on an ascii database listing distributed with it. This listing supports many Ghostscript and Uniprint drivers, but not filter−style drivers. The filters constructed also do not support much in the way of user−controllable options at print time.

The **printtool** places a configuration file named `postscript.cfg` in the spool directory. Inside this Bourne shell−style file, each setting is a variable. In unusual cases, you can make useful changes directly to the config file which the printtool won't allow; typically this would be the specification of an unusual Ghostscript driver, or a PPD filename for the VA rhs−printfilters version.

VA Linux has made some enhancements to the rhs−printfilters system under contract from HP. With the proper versions, it is now possible to select options for Postscript printers under control of Adobe PPD files. I cover this system in Section 8.2.2.

There's one catch to such filters: older version of lpd don't run the *if* filter for remote printers, while most newer ones do (although often with no arguments). The version of LPD shipped with modern Linux and FreeBSD distributions does; most commercial unices that still ship LPD have a version that does not. See the section on network printing later in this document for more information on this. If you only have locally−connected printers, then this won't affect you.

8.2.2. LPD for PostScript Printers

While most versions of LPD don't gracefully handle PostScript (nevermind user options), VA Linux recently modified LPD and Red Hat's filtering software to support PostScript printers fairly well. For the moment, this system works only with Red Hat 6.2, although the packages could be easily adapted for other distributions.

8.2.2.1. How it works

VA's new system uses Postscript Printer Definition, or PPD, files. PPD files are provided by printer manufacturers and declare the available options on a printer, along with the Postscript code needed to activate them. With the VA system, the normal LPD scheme works a little differently:

1. The user can specify options with the `−o` flag. For example, you might specify `−o MediaType:Transparency` if you were about to print on overhead film. Alternatively, the front−end GPR can be used to specify options in a dialog box; you can see screenshots of GPR in Section 3.3.1.
2. LPR passes the options to LPD as an extended attribute in the LPD control file.
3. A modified version of the rhs−printfilters package is given the extended options data in an environment variable, and uses ppdfilt to add these options to the print data.

8.2.2.2. Obtaining and Installing

You can obtain RPM packages, or source tarballs, from the project's website on SourceForge. For installation details, consult the project's installation micro−HOWTO. In essence, you need to uninstall the Red Hat version of printtool, lpd, and rhs−printfilters entirely, and then install the VA versions, plus ppdfilt, gpr, and a few other utilities.

You will also need PPD files for your Postscript printers. PPD files are usually fairly easy to find. VA Linux and HP distribute PPD files for many Laserjet models. Other vendors provide PPDs for their own printers, and Adobe distributes PPD files for many printers.

At the moment, much of this is a bit difficult to install. But future installation tools will build upon the printer configuration library **libprinterconf**, which enables both the autodetection and rhs−printfilter configuration of both networked and local printers.

> **Note:** It is possible to use GPR alone, without the modified LPD or even rhs−printfilters. GPR can be compiled with all the logic needed to massage Postscript jobs directly. This may be an easier−to−install option suitable for people who never really need to print using lpr directly.

8.2.2.3. Controlling Postscript Options

Once you've setup VA's Postscript−capable LPD system, you can control your printer's options in two ways:

With the GUI

To use GPR, you first make sure that you've specified the proper PPD file. Then the printer's options will be available on the `Advanced' panel. Basic **ppdfilt** options will be available on the `Common' panel.

With the command line

This **lpr** supports the `−o` option. You may specify any option/value pair from your printer's PPD file with `−o`. For example, consider this PPD file option clause:

```
*OpenUI *PrintQuality/Print Quality: PickOne
*DefaultPrintQuality: None
*OrderDependency: 150 AnySetup *PrintQuality
*PrintQuality None/Printer Setting: ""
*PrintQuality Quick/QuickPrint:  "<< /DeviceRenderingInfo ...
*PrintQuality Normal/Normal: "<< /DeviceRenderingInfo << /...
*PrintQuality Pres/Presentation: "<< /DeviceRenderingInfo ...
*PrintQuality Image/1200 Image Quality: "<< /DeviceRenderi...
*CloseUI: *PrintQuality
```

For the option `PrintQuality`, the possible values are `Quick`, `Normal`, `Pres`, or `Image`. You might give a command like:

```
% lpr −o PrintQuality:Image file.ps
```

There are a number of options common to all printers which will work in addition to the ones from your PPD. These include:

`page−ranges`

You can specify a range of pages to print. For example, `page−ranges:2−3`.

`page−set`

You can print only odd or even pages. For example, `page−set:odd`.

`number−up`

You can print multiple pages on each piece of paper. For example, `number−up:2`.

Other options are detailed in the **ppdfilt** man page.

8.2.3. File Permissions

By popular demand, I include below a listing of the permissions on interesting files on my system. There are a number of better ways to do this, ideally using only SGID binaries and not making everything SUID root, but this is how my system came out of the box, and it works for me. (Quite frankly, if your vendor can't even ship a working lpd you're in for a rough ride).

```
-r-sr-sr-x   1 root     lp    /usr/bin/lpr*
-r-sr-sr-x   1 root     lp    /usr/bin/lprm*
-rwxr--r--   1 root     root  /usr/sbin/lpd*
-r-xr-sr-x   1 root     lp    /usr/sbin/lpc*
drwxrwxr-x   4 root     lp    /var/spool/lpd/
drwxr-xr-x   2 root     lp    /var/spool/lpd/lp/
```

Lpd must currently be run as root so that it can bind to the low−numbered lp service port. It should probably become UID lp.lp or something after binding, but I don't think it does. This is simply one more reason to avoid the stock BSD LPD.

PDQ uses a different, non−daemon−centric scheme, so it has different programs. The only SUID root programs are the lpd interface programs **lpd_cancel**, **lpd_print**, and **lpd_status**; these are SUID because actual Unix print servers require print requests to originate from a priviledged port. If the only printers for which you use PDQ's bsd−lpd interface are networked print servers (like the HP JetDirect or Lexmark's MarkNet adapters) then you do not need the SUID bit on these programs.

8.3. Large Installations

Large installations, by which I mean networks including more than two printers or hosts, have special needs. Below are some tips. For really large environments, merely distributing printcap/filter information becomes a difficult problem; the Cisco Enterprise Print System addresses this and is probably either a good starting point or a nearly complete solution, depending on your needs. Medium to large environments can be well supported by native LPRng features.

- Each printer should have a single point of control, where an administrator can pause, reorder, or redirect the queue. To implement this, have everyone printing to a local server, which will then queue jobs and direct them to the proper printer. For large campuses or distributed networks, have one server per building or other suitable network subset.
- Use LPRng, at least on servers; the BSD LPD is too buggy for "real" use. So is CUPS, at least right now in mid−2000. But don't take my word for it you should test a number of spoolers and see which suits you best.
- Client systems should not have unique printing configurations. To implement this, use LPRng's extended printcap syntax so that you have one printcap to use everywhere. CEPS provides for this by building atop a lightweight distributed database instead of traditional printcap files.
- Print queues should not be named for make or model; name print queues for something sensible like location (floor2_nw) or capability (color_transparency). Three years from now, when a printer breaks, you will be able to replace it with a different make or model without causing confusion.

* Operate a web page which shows detailed information on each printer, including location, capabilities, etc. Consider having it show the queue and include a button to remove jobs from the queue. Complex networked environments are unmanagable for users without proper documentation.
* On Unix systems, use PDQ or the like to allow selection of print job attributes such as duplex or paper size, and to force users to run all Ghostscript processing under the proper user ID. If you have all Postscript printers (as is best), you can also select from the GPR or XPP front−ends; both are prettier.
* On Windows and Apple systems, use either the platform−specific drivers *everywhere* (Samba supports the Windows automagical driver−download mechanism) or, better, use generic Postscript drivers *everywhere*. Do not mix and match; primitive word processors often produce different output when the installed printer driver changes; users cannot deal with output that vaires depending on the particular client/printer pair.
* If at all possible, buy a large−volume printer for large−volume printing. If on a budget, use LPRng's multiple printers/one queue facility and assign a babysitter; printers are complex mechanical devices that will often jam and run out of paper in such configurations.
* Do not feel that printers must be plugged into workstations; Ethernet "print servers" now cost under $100. The ability to locate printers anywhere you can network is a big improvement over forced location near a host; locate printers in sensible, central locations.
* Use any SNMP trap or other monitoring/alert facility available to you − someone should be tasked with running around and fixing printers with no ink or paper. Npadmin (see Section 11.10.1) can be used to do some management operations with SNMP printers.

8.4. Accounting

Regular LPD provides very little to help you with accounting. You can specify the name of an accounting file in the `af` printcap attribute, but this is merely passed as an argument to your `if` filter. It's up to you to make your `if` filter write entries to the accounting file, and up to you to process the accounting file later (the traditional format is mainly useful for line printers, and is nontrivial to parse in Perl, so there's no reason to preserve it). Also, if you're using my **lpdomatic** program as your filter, you'll need to make changes, since it depends on being given a configuration file as the ``accounting'' file name.

Ghostscript provides a PageCount operator that you can use to count the number of pages in each job; basically you just tack a few lines of postscript onto the end of the job to write an accounting file entry; for the best example of this see the file `unix−lpr.sh` in the Ghostscript source distribution.

Note that the **unix−lpr** implementation of accounting writes to a file from the Ghostscript interpreter, and is thus incompatible with the recommended −dSAFER option. A better solution might be to query the printer with a PJL command after each job, or to write a postscript snippet that prints the pagecount on stdout, where it can be captured without having to write to a file.

The LPRng print spooler includes an HP−specific sample implementation of accounting; I assume that it queries the printer with PJL. This technique should work for most PJL, Postscript, or SNMP printers with which you have two−way communications.

If you have a networked printer that supports SNMP, you can use the npadmin program to query a pagecount after each job. This should work properly for all print jobs. See Section 11.10.1 for more information on npadmin.

9. Vendor Solutions

This section is, by definition, incomplete. Feel free to send in details of your favourite distribution. At the moment, I am aware of no distribution that supports, or even provides, the software I recommend: PDQ.

There are a number of third−party packages out there designed to make printer configuration under Unix easy. These are covered in Section 8; see the subsection there for your particular spooling software for pointers.

9.1. Red Hat

Red Hat has a GUI printer administration tool called printtool which can add remote printers and printers on local devices. It lets you choose a ghostscript−supported printer type and Unix device file to print to, then installs a print queue in `/etc/printcap` and uses a filter program from the rhs−printfilters package to support postscript and other common input types. This solution works fairly well, and is trivial to setup for common cases.

Where Red Hat fails is when you have a printer which isn't supported by their standard Ghostscript (which is GNU rather than Aladdin Ghostscript, and which supports fewer printers). Check in the printer compatibility list above (or online) if you find that you can't print properly with the stock Red Hat software. If your printer isn't supported by Red Hat's tools, you may need to install a contributed verison of Aladdin Ghostscript, and will probably also be better off if you use the lpdomatic or apsfilter packages, which know all about the printers supported by late−model Ghostscripts, and others besides.

In future versions of Red Hat the printtool will be reimplemented to support a larger list of printers and with the intent to support an eventual rhs−printfilters replacement (the current filter has difficulty with many common printers like some non−PCL DeskJets and most Lexmarks). Some VA Linux−developed PPD features may be incorporated, as well.

9.2. Debian

Debian offers a choice between plain LPD, LPRng, or CUPS; LPRng or CUPS are probably the better choices. I believe Debian also offers a choice of printer configuration tools; apsfilter version 5 or later is probably your best bet, since that verison adds support for LPRng and Ghostscript's uniprint driver scheme. Red Hat's printtool is also supported, for those who like GUI administration tools.

9.3. SuSE

The printing system on SuSE Linux is based on apsfilter, with some enhancements; SuSE's apsfilter will recognize all common file formats (including HTML, if html2ps is installed). There are two ways to setup printers on SuSE systems:

- YaST will let you configure "PostScript", "DeskJet" and "Other printers", supported by Ghostscript drivers; it's also possible to setup HP's GDI printers (DeskJet 710/720, 820, 1000, via the "ppa" package). YaST will provide `/etc/printcap` entries for every printer ("raw", "ascii", "auto" and

"color", if the printer to configure is a color printer). YaST will create spool directories and it will arrange apsfilterrc files, where you're able to fine tune some settings (Ghostscript preloads, paper size, paper orientation, resolution, printer escape sequences, etc.). With YaST it's also possible to setup network printers (TCP/IP, Samba, or Novell Netware Printer).
- In addition SuSE includes the regular SETUP program from the original apsfilter package (with some enhancements); run **lprsetup** to invoke this configuration script. Once you get accustomed to its GUI, you'll be able to configure local and network printers.

The SuSE installation manual explains both of these setup procedures.

Wolf Rogner reported some difficulties with SuSE. Apparently the following bugs may bite:

- Apsfilter's regular SETUP script is a bit broken, as are the KDE setup tools. Use YaST. [Ed: does this still apply? It's been some time sice Wolf's report.]
- For networked printers that need to be fed from Ghostscript, you'll need to first uncomment the line REMOTE_PRINTER="remote" in `/etc/apsfilterrc`. Then run YaST to configure the printer and, under Network configurations, set up a remote printer queue.
- YaST's setup doesn't allow color laser printers, so configure a mono printer and then change mono to color everwhere in the printcap entry. You may have to rename the spool directory, too.

9.4. Caldera

Caldera ships LPRng. I have no idea what sort of setup tools they offer.

9.5. Corel

Corel is Debian−based, so all the Debian facts above should still apply. In addition, they've written their own setup tool, based on the sysAPS library which in turn uses my database. They've certainly done so as part of WordPerfect.

Corel operates a printing support newsgroup named corelsupport.linux.printing. The bulk of the traffic appears to be WordPerfect and Corel Linux related.

9.6. Mandrake

As of version 7.2b1, Mandrake ships with CUPS standard. The program QtCUPS is used to provide a clean GUI administration interface. Till went to some trouble to include as many drivers as possible, and they ship CUPS PPD files build with my own foomatic interface code.

I think Earlier Mandrake versions shipped with the Red Hat printtool.

9.7. Other Distributions

Please send me info on what other distributions do!

10. Ghostscript.

Ghostscript is an incredibly significant program for Linux printing. Most printing software under Unix generates PostScript, which is typically a $100 option on a printer. Ghostscript, however, is free, and will generate the language of your printer from PostScript. When tied in with your PDQ printer driver declaration or **lpd** input filter, it gives you a virtual PostScript printer and simplifies life immensely.

Ghostscript is available in two forms. The commercial version of Ghostscript, called Aladdin Ghostscript, may be used freely for personal use but may not be distributed by commercial entities. It is generally a year or so ahead of the free Ghostscript; at the moment, for example, it supports many color inkjets that the older Ghostscripts do not and has rather better PDF support.

The free version of Ghostscript is GNU Ghostscript, and is simply an aged version of Aladdin ghostscript. This somewhat awkward arrangement has allowed Aladdin to be a totally self–funded free software project; the leading edge versions are done by L Peter and a few employees, and are licensed to hardware and software vendors for use in commercial products. Unfortunately, while this scheme has provided for L Peter's continued work on Ghostscript for years, it has also inhibited the participation of the wider free software community. Driver authors, in particular, find the arrangement poor. L Peter's retirement plans mandate a larger community involvement in the project, so he is considering license changes, and has established a SourceForge project.

Whatever you do with **gs**, be very sure to run it with the option for disabling file access (`–dSAFER`). PostScript is a fully functional language, and a bad PostScript program could give you quite a headache.

Speaking of PDF, Adobe's Portable Document Format (at least through 1.3) is actually little more than organized PostScript in a compressed file. Ghostscript can handle PDF input just as it does PostScript. So you can be the first on your block with a PDF–capable printer.

10.1. Invoking Ghostscript

Typically, Ghostscript will be run by whatever filter you settle upon (I recommend **apsfilter** or my own **lpdomatic** if your vendor didn't supply anything that suits you), but for debugging purposes it is often handy to run it directly.

gs –help will give a brief listing of options and available drivers (note that this list is the list of drivers compiled in, not the master list of all available drivers).

You might run gs for testing purposes like: `**gs <options> –q –dSAFER –sOutputFile=/dev/lp1 test.ps**'.

10.2. Ghostscript output tuning

There are a number of things one can do if Ghostscript's output is not satisfactory (actually, you can do anything you darn well please, since you have the source).

Some of these options, and others are described in the Ghostscript User Guide (the file `Use.htm` in the Ghostscript distribution; possibly installed under `/usr/doc` or `/usr/share/doc` on your system) are all excellent candidates for driver options in your filter system or PDQ driver declaration.

10.2.1. Output location and size

The location, size, and aspect ratio of the image on a page is controlled by the printer−specific driver in ghostscript. If you find that your pages are coming out scrunched too short, or too long, or too big by a factor of two, you might want to look in your driver's source module and adjust whatever parameters jump out at you. Unfortunately, each driver is different, so I can't really tell you what to adjust, but most of them are reasonably well commented.

10.2.2. Gamma, dotsizes, etc.

Most non−laser printers suffer from the fact that their dots are rather large. This results in pictures coming out too dark. If you experience this problem with an otherwise untunable driver, you could use your own transfer function. Simply create the following file in the ghostscript lib−dir and add its name to the gs call just before the actual file. You may need to tweak the actual values to fit your printer. Lower values result in a brighter print. Especially if your driver uses a Floyd−Steinberg algorithm to rasterize colors, lower values (0.2 − 0.15) are probably a good choice.

```
%!
%transfer functions for cyan magenta yellow black
{0.3 exp} {0.3 exp} {0.3 exp} {0.3 exp} setcolortransfer
```

It is also possible to mend printers that have some kind of color fault by tweaking these values. If you do that kind of thing, I recommend using the file `colorcir.ps`, that comes with ghostscript (in the `examples/` subdirectory), as a test page.

For many of the newer color inkjet drivers, there are command−line options, or different upp driver files, which implement gamma and other changes to adapt the printer to different paper types. You sould look into this before playing with Postscript to fix things.

10.2.3. Color Printing in Ghostscript

Ghostscript's default color dithering is optimized for low−resolution devices. It will dither rather coarsely in an attempt to produce 60ppi output (not dpi, ppi − the "apparent" color pixels per inch you get after dithering). This produces rather poor output on modern color printers; inkjets with photo paper, in particular, are capable of mich finer ppi settings.

To adjust this, use the Ghostscript option `−dDITHERPPI=x`, where `x` is the value to use. This may or may not have an effect with all drivers; many newer drivers (the Epson Stylus **stp** driver, for example) implement their own dithering and pay no attention to this setting. Some drivers can use either the regular Ghostscript or driver−specific dithering (the Canon Bubblejet **bjc600** driver, for example).

Ghostscript's dithering is in fact rather rudimentary. Many things needed for good output on modern printers are simply not available in the Ghostscript core. Various projects to fix this situation and the free software world does have the software to do so ready and waiting are hampered by Ghostscript's licensing situation and the resulting "cathedral" development style. Beginning at the Open Source Printing Summit 2000, however, all the necessary people are talking, so you can expect this situation to improve shortly.

11. Networks

One of the features of most spoolers is that they support printing over the network to printers physically connected to a different machine, or to the network directly. With the careful combination of filter scripts and assorted utilities, you can print transparently to printers on all sorts of networks.

11.1. Printing to a Unix/lpd host

To allow remote machines to print to your printer using the LPD protocol, you must list the machines in */etc/hosts.equiv* or */etc/hosts.lpd*. (Note that *hosts.equiv* has a host of other effects; be sure you know what you are doing if you list any machine there). You can allow only certain users on the other machines to print to your printer by usign the *rs* attribute; read the `lpd` man page for information on this.

11.1.1. With `pdq`

With PDQ, you define a printer with the interface type "bsd−lpd". This interface takes arguments for the remote hostname and queue name; the printer definition wizard will prompt you for these.

11.1.2. With `lpd`

To print to another machine, you make an */etc/printcap* entry like this:

```
# REMOTE djet500
lp|dj|deskjet:\
        :sd=/var/spool/lpd/dj:\
        :rm=machine.out.there.com:\
        :rp=printername:\
        :sh:
```

Note that there is still a spool directory on the local machine managed by `lpd`. If the remote machine is busy or offline, print jobs from the local machine wait in the spool area until they can be sent.

11.1.3. With `rlpr`

You can also use *rlpr* to send a print job directly to a queue on a remote machine without going through the hassle of configuring lpd to handle it. This is mostly useful in situations where you print to a variety of printers only occasionally. From the announcement for *rlpr*:

Rlpr uses TCP/IP to send print jobs to lpd servers anywhere on a network.

Unlike lpr, it *does not* require that the remote printers be explicitly known to the machine you wish to print from, (e.g. through */etc/printcap*) and thus is considerably more flexible and requires less administration.

rlpr can be used anywhere a traditional lpr might be used, and is backwards compatible with traditional BSD lpr.

The main power gained by rlpr is the power to print remotely *from anywhere to anywhere* without regard for how the system you wish to print from was configured. Rlpr can work as a filter just like traditional lpr so that clients executing on a remote machine like netscape, xemacs, etc, etc can print to your local machine with little effort.

Rlpr is available from `Metalab`.

11.2. Printing to a Windows or Samba printer

There is a Printing to Windows mini−HOWTO out there which has more info than there is here.

11.2.1. From PDQ

There is not a prebuilt smb interface that I am aware of, but it would be fairly easy to create using the model set by the Netatalk−based appletalk interface. Someone please create one and submit it for inclusion!

Read the Windows/LPD section below for more tips on how to do it.

11.2.2. From LPD

It is possible to direct a print queue through the `smbclient` program (part of the samba suite) to a TCP/IP based SMB print service. Samba includes a script to do this called `smbprint`. In short, you put a configuration file for the specific printer in question in the spool directory, and install the `smbprint` script as the *if*.

The */etc/printcap* entry goes like this:

```
lp|remote-smbprinter:\
    :sh:\
    :lp=/dev/null:\
    :sd=/var/spool/lpd/lp:\
    :if=/usr/local/sbin/smbprint:
```

You should read the documentation inside the `smbprint` script for more information on how to set this up.

You can also use `smbclient` to submit a file directly to an SMB printing service without involving `lpd`. See the man page.

11.3. Printing to a NetWare Printer

The ncpfs suite includes a utility called `nprint` which provides the same functionality as `smbprint` but for NetWare. You can get ncpfs from Metalab. From the LSM entry for version 0.16:

" With ncpfs you can mount volumes of your netware server under Linux. You can also print to netware print queues and spool netware print queues to the Linux printing system. You need kernel 1.2.x or 1.3.54 and above. ncpfs does NOT work with any 1.3.x kernel below 1.3.54. "

11.3.1. From LPD

To make `nprint` work via lpd, you write a little shell script to print stdin on the NetWare printer, and install that as the *if* for an lpd print queue. You'll get something like:

```
sub2|remote-NWprinter:\
        :sh:\
        :lp=/dev/null:\
        :sd=/var/spool/lpd/sub2:\
        :if=/var/spool/lpd/nprint-script:
```

The `nprint-script` might look approximately like:

```
#! /bin/sh
# You should try the guest account with no password first!
/usr/local/bin/nprint -S net -U name -P passwd -q printq-name -
```

11.4. Printing to an EtherTalk (Apple) printer

The netatalk package includes something like `nprint` and `smbclient`. Others have documented the procedure for printing to and from an Apple network far better than I ever will; see the Linux Netatalk−HOWTO.

11.4.1. From PDQ

PDQ includes an interface declaration called "appletalk". This uses the Netatalk package to print to a networked Apple printer. Just select this interface in xpdq's "Add printer" wizard.

11.5. Printing to a networked printer

Many printers come with an ethernet interface which you can print to directly, typically using the LPD protocol. You should follow the instructions that came with your printer or its network adaptor, but in general, such printers are "running" lpd, and provide one or more queues which you can print to. An HP, for example, might work with a printcap like:

```
lj-5|remote-hplj:\
        :sh:\
        :sd=/var/spool/lpd/lj-5:\
        :rm=printer.name.com:\
        :rp=raw:
```

or, using the PDQ bsd−lpd interface arguments of REMOTE_HOST=printer.name.com and QUEUE=raw.

HP Laserjet printers with JetDirect interfaces generally support two built in lpd queues − "raw" which accepts PCL (and possibly Postscript) and "text" which accepts straight ascii (and copes automatically with the staircase effect). If you've got a JetDirect Plus3 three−port box, the queues are named "raw1", "text2", and so forth.

Note that the ISS company has identified an assortment of denial of service attacks which hang HP Jetdirect interfaces. Most of these have been addressed beginning in Fall 98. These sorts of problems are common in embedded code; few appliance−style devices should be exposed to general Internet traffic.

In a large scale environment, especially a large environment where some printers do not support PostScript, it may be useful to establish a dedicated print server to which all machines print and on which all ghostscript jobs are run. This will allow the queue to be paused or reordered using the topq and lprm commands.

This also allows your Linux box to act as a spool server for the printer so that your network users can complete their print jobs quickly and get on with things without waiting for the printer to print any other job that someone else has sent. This is suggested too if you have unfixable older HP Jetdirects; it reduces the likelihood of the printers wedging.

To do this, set up a queue on your linux box that points at the ethernet equipped HP LJ (as above). Now set up all the clients on your LAN to point at the Linux queue (eg lj−5 in the example above).

Some HP network printers apparently don't heed the banner page setting sent by clients; you can turn off their internally generated banner page by telnetting to the printer, hitting return twice, typing "banner: 0" followed by "quit". There are other settings you can change this way, as well; type "?" to see a list.

The full range of settings can be controlled with HP's webJetAdmin software. This package runs as a daemon, and accepts http requests on a designated port. It serves up forms and Java applets which can control HP printers on the network. In theory, it can also control Unix print queues, but it does so using the rexec service, which is completely unsecure. I don't advise using that feature.

11.5.1. To AppSocket Devices

Some printers (and printer networking "black boxes") support only a cheesy little non−protocol involving plain TCP connections; this is sometimes called the "AppSocket" protocol. Notable in this category are early−model JetDirect (including some JetDirectEx) cards. Basically, to print to the printer, you must open a TCP connection to the printer on a specified port (typically 9100, or 9100, 9101 and 9102 for three−port boxes) and stuff your print job into it. LPRng has built−in support for stuffing print jobs into random TCP ports, but with BSD lpd it's not so easy. The best thing is probably to obtain and use the little utility called netcat.

A netcat−using PDQ interface would look something like this:

```
interface tcp-port-0.1 {

   help "This is one of the first interfaces supported by standalone
         network printers and print servers.  The device simply
         listens for a TCP connection on a certain port, and sends
         data from any connection to the printer.\nThis interface
         requires the netcat program (\"nc\")."

   required_args "REMOTE_HOST"

   argument {
```

```
      var = "REMOTE_HOST"
      desc = "Remote host"
      help = "This is IP name or number of the print server."
   }

   argument {
      var = "REMOTE_PORT"
      def_value = "9100"
      desc = "Remote port"
      help = "This is the TCP port number on the print server that the
              print job should be sent to.  Most JetDirect cards, and
              clones, accept jobs on port 9100 (or 9101 for port 2,
              etc)."
   }

   requires "nc"

   send_exec { cat $OUTPUT | nc $REMOTE_HOST $REMOTE_PORT }

}
```

Failing that, it can be implemented, among other ways, in Perl using the program below. For better performance, use the program netcat ("nc"), which does much the same thing in a general purpose way. Most distributions should have netcat available in prepackaged form.

```
#!/usr/bin/perl
# Thanks to Dan McLaughlin for writing the original version of this
# script (And to Jim W. Jones for sitting next to Dan when writing me
# for help ;)

$fileName = @ARGV[0];

open(IN,"$fileName") || die "Can't open file $fileName";

$dpi300     = "\x1B*t300R";
$dosCr      = "\x1B38;k3G";
$ends = "\x0A";

$port =  9100 unless $port;
$them = "bach.sr.hp.com" unless $them;

$AF_INET = 2;
$SOCK_STREAM = 1;
$SIG{'INT'} = 'dokill';
$sockaddr = 'S n a4 x8';

chop($hostname = `hostname`);
($name,$aliases,$proto) = getprotobyname('tcp');
($name,$aliases,$port) = getservbyname($port,'tcp')
    unless $port =~ /^\d+$/;;
($name,$aliases,$type,$len,$thisaddr) =
        gethostbyname($hostname);
($name,$aliases,$type,$len,$thataddr) = gethostbyname($them);
$this = pack($sockaddr, $AF_INET, 0, $thisaddr);
$that = pack($sockaddr, $AF_INET, $port, $thataddr);

if (socket(S, $AF_INET, $SOCK_STREAM, $proto)) {
#    print "socket ok\n";
}
else {
    die $!;
```

```
}
# Give the socket an address.
if (bind(S, $this)) {
#    print "bind ok\n";
}
else {
    die $!;
}

# Call up the server.

if (connect(S,$that)) {
#    print "connect ok\n";
}
else {
    die $!;
}

# Set socket to be command buffered.

select(S); $| = 1; select(STDOUT);

#    print S "@PJL ECHO Hi $hostname! $ends";
#    print S "@PJL OPMSG DISPLAY=\"Job $whoami\" $ends";
#    print S $dpi300;

# Avoid deadlock by forking.

if($child = fork) {
    print S $dosCr;
    print S $TimesNewR;

    while (<IN>) {
        print S;
    }
    sleep 3;
    do dokill();
} else {
    while(<S>) {
        print;
    }
}

sub dokill {
    kill 9,$child if $child;
}
```

11.6. Running an `if` for remote printers with old LPDs

One oddity of older versions of lpd is that the `if` is not run for remote printers. (Versions after 0.43 or so have the change originated on FreeBSD such that the `if` is always run). If you find that you need to run an `if` for a remote printer, and it isn't working with your lpr, you can do so by setting up a double queue and requeueing the job. As an example, consider this `printcap`:

```
lj-5:\
        :lp=/dev/null:sh:\
        :sd=/var/spool/lpd/lj-5:\
        :if=/usr/lib/lpd/filter-lj-5:
```

```
lj-5-remote:sh:rm=printer.name.com:\
        :rp=raw:sd=/var/spool/lpd/lj-5-raw:
```

in light of this **filter−lj−5** script:

```
#!/bin/sh
gs <options> -q -dSAFER -sOutputFile=- - | \
        lpr -Plj-5-remote -U$5
```

The `-U` option to lpr only works if lpr is run as daemon, and it sets the submitter's name for the job in the resubmitted queue correctly. You should probably use a more robust method of getting the username, since in some cases it is not argument 5. See the man page for `printcap`.

11.7. From Windows.

Printing from a Windows (or presumably, OS/2) client to a Linux server is directly supported over SMB through the use of the SAMBA package, which also supports file sharing of your Linux filesystem to Windows clients.

Samba includes fairly complete documentation, and there is a good Samba FAQ which covers it, too. You can either configure a magic filter on the Linux box and print PostScript to it, or run around installing printer−specific drivers on all the Windows machines and having a queue for them with no filters at all. Relying on the Windows drivers may in some cases produce better output, but is a bit more of an administrative hassle if there are many Windows boxen. So try Postscript first. Modern versions of Samba should support the automagical driver download mechanism offered by Windows NT servers to deal with this problem.

With PDQ, you should configure Samba to run the pdq command with appropriate arguments instead of the lpr command that it defaults to running. I believe that Samba will run pdq as the proper user, so it should work well this way. There are several Samba options that you should adjust to do this:

printcap

: This should point to a "fake" printcap you whip up listing available printers. All you need is a short and long name for each printer, one per line:

```
lp1|Printer One
lp2|Printer Two
lp3|Printer Three
```

The short name will be used as the printer name for the print command:

print command

: This will need to be set to something like `pdq -P %p %s ; rm %s`.

lprm command

: There doesn't seem to be a good value for this setting at the moment. PDQ's queued jobs will expire after a time, so if the printer is totally gone there's no problem. If you just change your mind, you can use **xpdq** to cancel jobs, but this is inconvenient from Windows. Just put a do−nothing command like **true** for now. If you use **lpd** or LPRng as the back−end, then a suitable **lprm** command should work. I'm not sure how Samba would identify the lpr queue entry number for a pdq−submitted job.

lpq command

Again, PDQ doesn't offer a good value to put here. Distributed systems don't offer a sensible way to see the queue, but samba−centric centralized server systems want to have a queue worth examining. Just put a do−nothing command like **true** for now. If you use LPD or LPRng as the back−end, then a suitable **lpq** command should work; you just won't see jobs until they're done being filtered by PDQ.

11.8. From an Apple.

Netatalk supports printing from Apple clients over EtherTalk. See the Netatalk HOWTO Page for more information.

11.9. From Netware.

The ncpfs package includes a daemon named pserver which can be used to privide service to a NetWare print queue. From what I understand, this system requires a Bindery−based NetWare, ie 2.x, 3.x, or 4.x with bindery access enabled.

For more information on ncpfs and it's pserver program, see the ncpfs FTP site.

11.10. Networked Printer Administration

Most networked printers support some method of remote administration. Often there are easy−to−use web pages for configuration. More usefully, there is often support for SNMP management. Typically you can find out interesting information on printer status like ink and paper levels, print volumes, and so forth, and you can usually change certain settings. SNMP printer control, and a number of other printing−related things, are being standardized by the IEEE's Printer Working Group

11.10.1. npadmin

Npadmin is a command−line program which offers an interface to the common SNMP functionality of networked printers. It implements the standard Printer MIB, as well as a few vendor−proprietary schemes used mainly for older devices. Both printer−discovery style actions and various printer status queries are supported.

npadmin has an excellent man page, and precompiled packages are distributed for a number of RPM and dpkg based distributions.

11.10.2. Other SNMP tools

Besides npadmin, there are a number of SNMP tools that will be useful. **snmptraplogd** can log SNMP trap events. This is useful for observing printer jams, out of paper events, etc; it would be straightforward to retransmit certain events to a pager, or to send an email.

While npadmin provides simplified support for many network printers' SNMP interfaces, some printers may have vendor extensions which npadmin doesn't know about. In this case, you can use the CMU SNMP tools, which support arbitrary SNMP GET and SET operations, as well as walks and the like. With these, and a bit of work, you can make use of any SNMP feature offered by your printer's MIB. You may need to obtain a MIB from your vendor to figure out what all the variables are; sometimes vendors think that people actually use the proprietary tools they ship.

VA Linux's **libprinterconf** includes code to perform network printer discovery. Printers are identified against a compiled−in library of printer signatures; at the moment the library is not large, but does cover many common networked printer models.

12. Windows−only printers

As I discussed earlier, some printers are inherently unsupported because they don't speak a normal printer language, instead using the computer's CPU to render a bitmap which is then piped to the printer at a fixed speed. In a few cases, these printers also speak something normal like PCL, but often they do not. In some (really low−end) cases, the printer doesn't even use a normal parallel connection but relies on the vendor's driver to emulate what should be hardware behaviour (most importantly flow control).

In any case, there are a few possible workarounds if you find yourself stuck with such a lemon.

12.1. The Ghostscript Windows redirector

There is now a Windows printer driver available (called **mswinpr2**) that will run a print job through Ghostscript before finally printing it. (Rather like an `if` filter in Unix's LPD). There is also a new Ghostscript driver which will print using Windows GDI calls. Taken all together, this allows a Windows machine to print PostScript to a Windows−only printer through the vendor's driver.

If you get that working, you can then follow the instructions above for printing to a Windows printer over the network from Linux to let Unix (and other Windows, Mac, etc) hosts print to your lemon printer.

12.2. HP Winprinters

Some HP printers use "Printing Performance Architecture" (marketingspeak for "we were too cheap to implement PCL"). This is supported in a roundabout way via the pbm2ppa translator written by Tim Norman. Basically, you use ghostscript to render PostScript into a bitmapped image in pbm format and then use pbm2ppa to translate this into a printer−specific ppa format bitmap ready to be dumped to the printer. This program may also come in ghostscript driver format by now.

The ppa software can be had from the ppa home page; pbm2ppa supports some models of the HP 720, 820, and 1000; read the documentation that comes with the package for more details on ppa printer support.

12.3. Lexmark Winprinters

Most of the cheap Lexmark inkjets use a proprietary language and are therefore Winprinters. However, Henryk Paluch has written a program which can print on a Lexmark 7000. Hopefully he'll be able to figure out color and expand support to other Lexmark inkjets. See here for more info.

Similarly, there are now drivers for the 5700, 1000, 1100, 2070, 3200, and others. See the supported printers listing above, and my web site, for more information on obtaining these drivers.

13. How to print to a fax machine.

You can print to a fax machine with, or without, a modem.

13.1. Using a faxmodem

There are a number of fax programs out there that will let you fax and receive documents. One of the most powerful is Sam Leffler's HylaFAX. It supports all sorts of things from multiple modems to broadcasting.

SuSE ships a Java HylaFax client which allegedly works on any Java platform (including Windows and Linux). There are also non−Java fax clients for most platforms; Linux can almost certainly handle your network faxing needs.

Also available, and a better choice for smaller installations, is **efax**, a simple program which sends and receives faxes. The getty program **mgetty** can receive faxes using **efax** (and do voicemail or interactive logins).

13.1.1. Faxing from PDQ

PDQ doesn't ship with a fax interface declaration, but here's a simple one (which is only partly tested):

```
interface efax-0.1 {
   help "This interface uses the efax package's fax program to send a
         fax.  You should first get efax's \"fax send\" working by
         itself by editing the file /etc/efax.rc and testing.  Connect
         this interface to a generic postscript driver to define a
         fax machine \"printer\"".

   requires { "efax" "fax" }

   # Making phone number required means that the add printer wizard
   # will demand a phone number at add printer time.  This is
   # undesirable, so it isn't explicitly required, even though it is
   # logically required.  The send_exec script checks for the number.
   # You could skip the wizard by adding this printer by hand to
   # .printrc, mark this as required, and it might then prompt?
   argument {
      var = "PHONE_NUMBER"
      desc = "Phone Number"
      help = "The phone number to dial.  Prefixes like 9 ought to be
              defined in your /etc/efax.rc file."
   }

   option {
      var = "RESOLUTION"
      desc = "Fax resolution"
      default_choice = "high"
      choice "low" {
         value = "-l"
         desc = "Low"
         help = "Low resolution on a fax is 96lpi."
      }
```

```
      choice "high" {
         value = ""
         desc = "High"
         help = "High resolution on a fax is 192lpi."
      }
   }

   # If you don't specify a phone number the job just fails, and
   # the only way to figure this out is to look at the error message
   # at the bottom of the job details.  Hmm.
   send_exec {
     if [ "x$PHONE_NUMBER" != "x" ]
     then
          fax send $RESOLUTION $PHONE_NUMBER $INPUT
     else
          echo 'You must specify a phone number!'
          false
     fi
   }
}
```

13.2. Using the Remote Printing Service

There is an experimental service offered that lets you send an email message containing something you'd like printed such that it will appear on a fax machine elsewhere. Nice formats like postscript are supported, so even though global coverage is spotty, this can still be a very useful service. For more information on printing via the remote printing service, see the Remote Printing WWW Site.

13.3. Commercial Faxing Services

A number of companies operate web−based faxing services. EFax, in particular, offers free inbound faxes (to your own dedicated fax number, no less) via email, and fax transmission for a fee. Other companies offer similar services.

14. How to generate something worth printing.

Here we get into a real rat's−nest of software. Basically, Linux can run many types of binaries with varying degrees of success: Linux/x86, Linux/Alpha, Linux/Sparc, Linux/foo, iBCS, Win16/Win32s (with dosemu and, someday, with Wine), Mac/68k (with Executor), and Java. I'll just discuss native Linux and common Unix software.

14.1. Markup languages

Most markup languages are more suitable for large or repetitive projects, where you want the computer to control the layout of the text to make things uniform.

nroff

This was one of the first Unix markup languages. Man pages are the most common examples of things formatted in *roff macros; many people swear by them, but nroff has, to me at least, a more arcane syntax than needed (see Figure 9), and probably makes a poor choice for new works. It is worth knowing, though, that you can typeset a man page directly into postscript with groff. Most man commands will do this for you with **man −t foo | lpr**.

Figure 9. Example of roff Input

```
.B man
is the system's manual pager. Each
.I page
argument given to
.B man
is normally the name of a program, utility or function.
The
.I manual page
associated with each of these arguments is then found and
displayed. A
.IR section ,
if provided, will direct
.B man
to look
only in that
.I section
of the manual.
```

TeX

TeX, and the macro package LaTeX, are one of the most widely used markup languages on Unix. Technical works are frequently written in LaTeX because it greatly simplifies the layout issues and is *still* one of the few text processing systems to support mathematics both completely and well. TeX's output format is `dvi`, and is converted to PostScript or Hewlett Packard's PCL with **dvips** or **dvilj**. If you wish to install TeX or LaTeX, install the whole teTeX group of packages; it contains everything. Recent TeX installations include pdfTeX and pdfLaTeX, which produce Adobe PDF files directly. Commands are available do create hyperlinks and navigation features in the PDF file.

Figure 10. Example of LaTeX Input

```
\subsubsection{NAT}

  Each real server is assigned a different IP address, and the NA
  implements address translation for all inbound and outbound
  packets.

  \begin{description}
  \item[Advantage] Implementation simplicity, especially if we
        already implement other NAT capabilities.

  \item[Disadvantage] Return traffic from the server goes through
        address translation, which may incur a speed penalty.  This
        probably isn't too bad if we design for it from the
        beginning.

  \item[Disadvantage] NAT breaks the end-to-end semantics of normal
        internet traffic.  Protocols like ftp, H.323, etc would
        require special support involving snooping and in-stream
        rewriting, or complete protocol proxying; neither is likely
        to be practical.
  \end{description}
```

SGML

There is at least one free SGML parser available for Unix and Linux; it forms the basis of Linuxdoc−SGML's homegrown document system. It can support other DTD's, as well, most notably DocBook. This document is written in DocBook−DTD SGML; see Figure 11 for an example.

Figure 11. Example of DocBook SGML

```
<VarListEntry>
 <Term>SGML</Term>
 <ListItem>
  <Para>
   There is at least one free SGML parser available for Unix
   and Linux; it forms the basis of Linuxdoc-SGML's homegrown
   document system.  It can support other DTD's, as well, most
   notably DocBook.  This document is written in DocBook-DTD
   SGML.
  </Para>
 </ListItem>
</VarListEntry>
```

14.2. WYSIWYG Word Processors

There is no shortage of WYSIWYG word processing software. Several complete office suites are available, including one that's free for personal use (StarOffice).

StarOffice

Sun Microsystems is distributing StarOffice on the net free for Linux. This full−blown office suite has all the features you'd expect, including both import and export of Microsoft Office file formats (including Word documents). There's a mini−HOWTO out there which describes how to obtain and install it. It generates PostScript, so should work with most any printer that works otherwise on Linux.

WordPerfect

Corel distributes a basic version of WordPerfect 8 free for Linux, and sells various packages of Word Perfect Office 2000 (which includes WordPerfect, Corel Draw and Quattro Pro Versions 9). The Linux WordPerfect Fonts and Printers page has information about configuring WordPerfect for use with either Ghostscript or its built−in printer drivers (which are apparently identical the DOS WordPerfect drivers, if your printer's driver isn't included in the distribution).

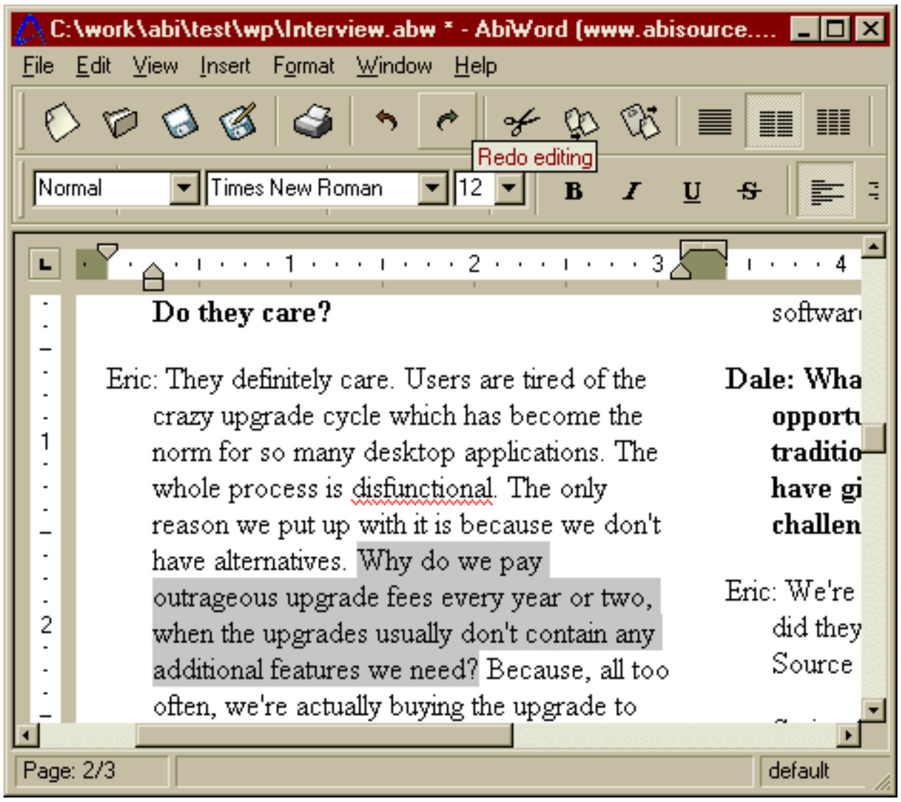
Applix

Applix is a cross−platform (ie, various Unices, Windows, and others) office suite sold by the Applix company. Red Hat and SuSE sold it themselves when it was the only game in town; now sales have reverted to Applix. This is the only native Unix−style application suite; it probably fits in better with the Unix way of doign things.

AbiWord

AbiWord is one of several GPL WYSIWYG word processor projects; this one has produced a very nice word processor based on an XML format. It is capable of Word file import. AbiWord is still a work in progress, although it is useful for small things now.

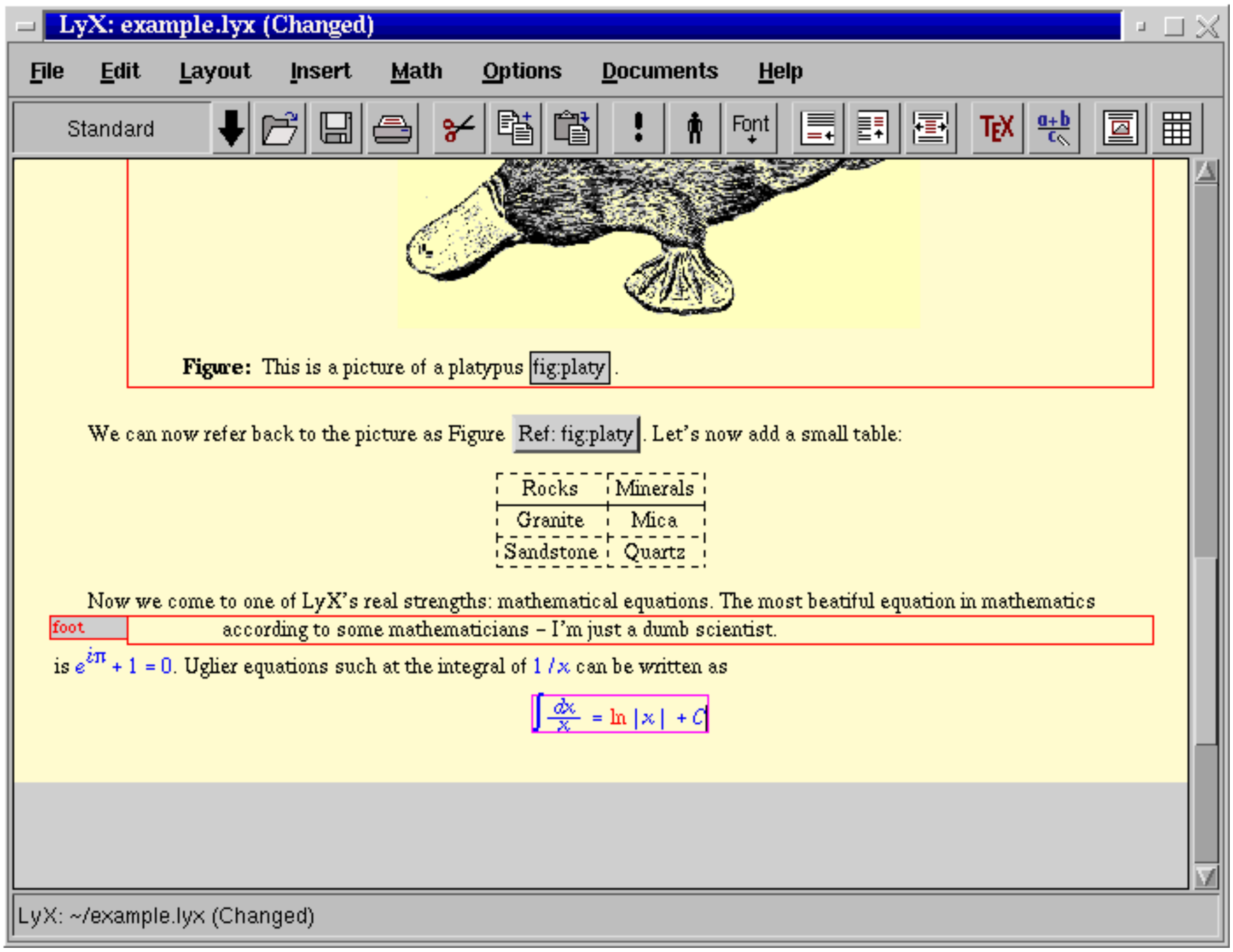
Figure 12. AbiWord



LyX

LyX is a front–end to LaTeX which looks very promising. See the LyX Homepage for more information. There is a KDE–styled version of LyX, called Klyx; the author of LyX and the instigator of KDE are the same person.

Figure 13. LyX



Maxwell

Maxwell is a simple MS RTF–format based word processor which started as a commercial product but is now distributed under the GPL.

Other vendors should feel free to drop me a line with your offerings.

15. Printing Photographs

There are many details to getting decent photo output from common printers. If you haven't bought a photo printer yet, see the photo–related tips in Section 5.4.

15.1. Ghostscript and Photos

Ghostscript has some difficulties rendering color photographs through most drivers. The problems are several:

- Many drivers have poorly tuned color support. Often the colors don't match the Windows driver output or the screen. OTOH, all drivers, and Ghostscript as a whole, have readily adjustable color support; the "Gamma" settings (see Section 10.2.2) are one thing to play with, and there are others documented in Ghostscript's `Use.htm` documentation file.
- I'm only aware of one Ghostscript driver with support for 6 and 7 color printing; it's in beta at the moment and supports most Epson Stylus Photo models. It is rumoured to produce better color than the Windows driver (!). The Ghostscript driver core itself provides no support for non CMYK or RGB colors; arguably, some work to put that there is needed.
- Ghostscript often ends up dithering coarsely, or generating printouts with artifacts like banding. The dithering can usually be corrected; see Section 10.2.3, and read the documentation for your driver.

You should be able to correct some of these problems by tuning Ghostscript; see Section 10 for more information on how to do this. Fiddling with Ghostscript options is much easier if you declare them as options in your spooling system.

That said, the obvious solution for now is to use non–Ghostscript software for printing photos, and indeed, such things do exist. The main contender is the print plugin in the Gimp, which supports pixel–for–pixel printing on Epson Styluses and Postscript printers (with basic PPD support). That Epson Stylus portion of that driver is available for Ghostcript, as well, as the **stp** driver. Also possible to use for this purpose are the assorted external pnm–to–foo programs used to print on printers like the cheap Lexmarks; these print attempt to print pixmaps pixel–for–pixel.

The best solution, of course, is to buy a Postscript printer; such printers can usually be completely controlled from available free software, and will print to the full capability of the printer.

15.2. Paper

Color inkjets are extremely dependent on the paper for good output. The expensive glossy coated inkjet papers will allow you to produce near–photographic output, while plain uncoated paper will often produce muddy colors and fuzzy details. Nonglossy coated inkjet papers will produce results in between, and are probably best for final prints of text, as well. Stiffer glossy coated "photo" papers will produce similar output to lighter–weight glossy papers, but will feel like a regular photo.

15.3. Printer Settings

For photo output on most color inkjets, you should use the most highly interlaced (and slowest) print mode; otherwise solid regions may have banding or weak colors. Generally with Ghostscript this is what will happen when you pick the highest resolution. With Postscript printers, you may need to add a snippet to the prologue based on the settings available in the PPD file. The Gimp's PPD support doesn't include (printer−specific) print quality settings, but I added one in an ugly way for my own use; contact me if you'd like that. If you use PDQ or CUPS, you can easily control all the printer settings you need. VA Linux's **libppd** and the GPR front−end can also add these options for Postscript printers.

15.4. Print Durability

Color inkjet printouts usually fade after a few years, especially if exposed to lots of light and air; this is a function of the ink. Printers with ink−only consumables like the Epsons and Canons can buy archival inks, which are less prone to this problem. Newer printers often use pigment−based inks, which don't fade as much as the older dye−based ink did. No inkjet output is really particularly good for long−term archival use. Write the bits to a CD−R and store that instead.

15.5. Shareware and Commercial Software

There's a program called xwtools which supports photo printing with all the bells and whistles on an assortment of Epson, HP, and Canon printers. Unfortunately, it was written under NDA, so comes without source. Unless you use it for the Epson Stylus Color 300 on Linux x86, it costs E15 for personal use; commercial pricing is unknown.

The ESP Print Pro package from Easy Software supports some printers which might otherwise be unsupported. These drivers are not reported to be very well−tuned for photos, but they do work.

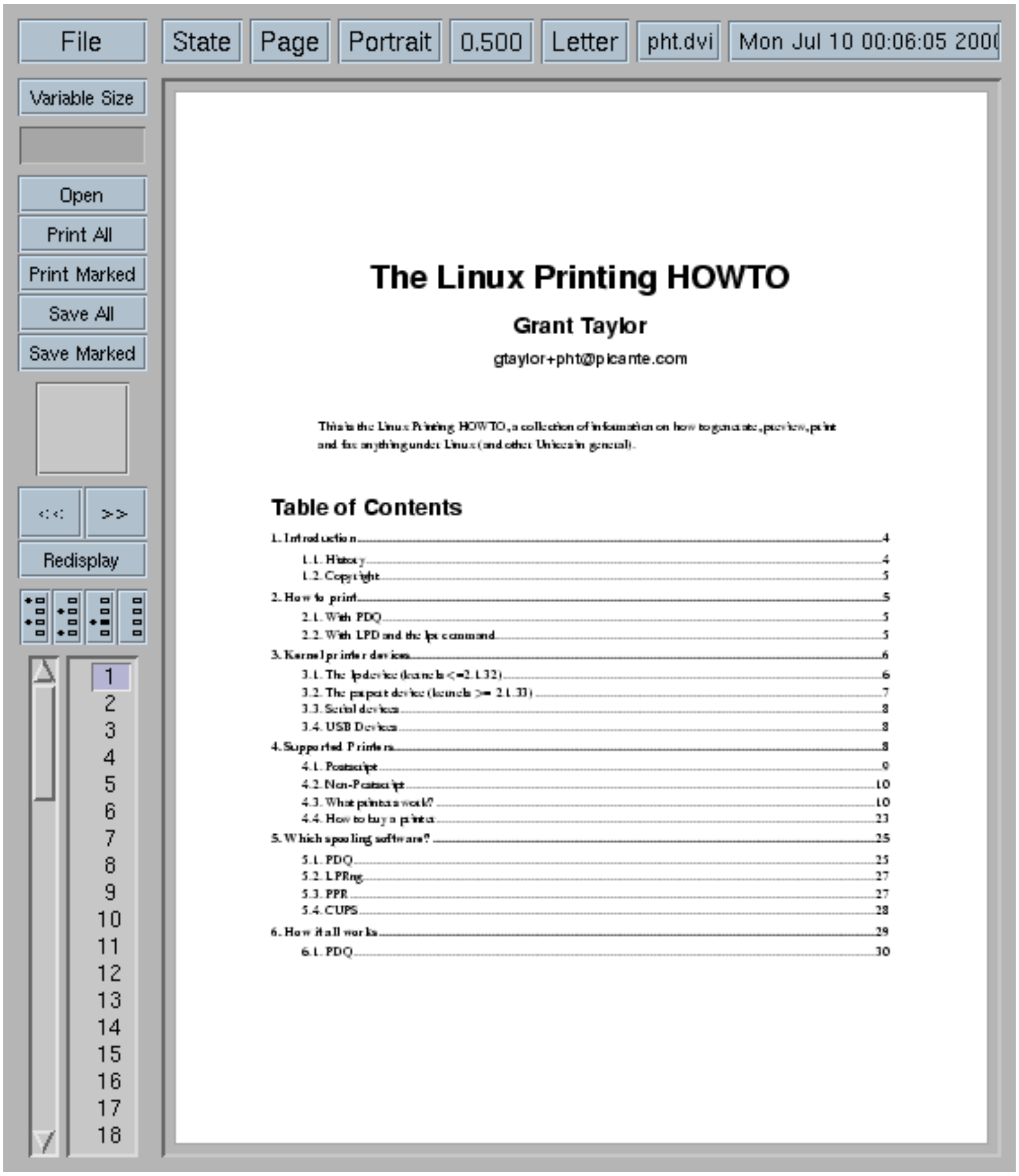
16. On−screen previewing of printable things.

Nearly anything you can print can be viewed on the screen, too.

16.1. PostScript

Ghostscript has an X11 driver best used under the management of the PostScript previewer gv. The latest versions of these programs should be able to view PDF files, as well. Note that gv has replaced the older previewer "Ghostview"; the new user interface is mch prettier and featureful that ghostview's plain old Athena GUI.

Figure 14. Gv



16.2. TeX dvi

TeX DeVice Independent files may be previewed under X11 with xdvi. Modern versions of xdvi call ghostscript to render PostScript specials.

A VT100 driver exists as well. It's called `dgvt`. `Tmview` works with Linux and svgalib, if that's all you can do.

16.3. Adobe PDF

Adobe's Acrobat Reader is available for Linux; just download it from the Adobe web site.

You can also use xpdf, which is free software, and I believe **gv** supports viewing PDF files with gs under X11.

17. Serial printers under lpd

Serial printers are rather tricky under lpd.

17.1. Setting up in printcap

Lpd provides five attributes which you can set in */etc/printcap* to control all the settings of the serial port a printer is on. Read the *printcap* man page and note the meanings of *br#*, *fc#*, *xc#*, *fs#* and *xs#*. The last four of these attributes are bitmaps indicating the settings for use the port. The *br#* atrribute is simply the baud rate, ie `` `br#9600' ``.

It is very easy to translate from stty settings to printcap flag settings. If you need to, see the man page for stty now.

Use stty to set up the printer port so that you can cat a file to it and have it print correctly. Here's what `*stty −a*' looks like for my printer port:

```
dina:/usr/users/andy/work/lpd/lpd# stty -a < /dev/ttyS2
speed 9600 baud; rows 0; columns 0; line = 0;
intr = ^C; quit = ^\; erase = ^?; kill = ^U; eof = ^D; eol = <undef>;
eol2 = <undef>; start = ^Q; stop = ^S; susp = ^Z; rprnt = ^R; werase = ^W;
lnext = ^V; min = 1; time = 0;
-parenb -parodd cs8 hupcl -cstopb cread -clocal -crtscts
-ignbrk -brkint -ignpar -parmrk -inpck -istrip -inlcr
-igncr -icrnl ixon -ixoff -iuclc -ixany -imaxbel
-opost -olcuc -ocrnl -onlcr -onocr -onlret -ofill -ofdel nl0 cr0 tab0
bs0 vt0 ff0
-isig -icanon -iexten -echo -echoe -echok -echonl -noflsh -xcase
-tostop -echoprt -echoctl -echoke
```

The only changes between this and the way the port is initialized at bootup are `-clocal`, `-crtscts`, and `ixon`. Your port may well be different depending on how your printer does flow control.

You actually use stty in a somewhat odd way. Since stty operates on the terminal connected to it's standard input, you use it to manipulate a given serial port by using the `<' character as above.

Once you have your stty settings right, so that `*cat file > /dev/ttyS2*' (in my case) sends the file to the printer, look at the file /usr/src/linux/include/asm–i386/termbits.h. This contains a lot of #defines and a few structs (You may wish to cat this file to the printer (you do have that working, right?) and use it as scratch paper). Go to the section that starts out

```
/* c_cflag bit meaning */
#define CBAUD   0000017
```

This section lists the meaning of the *fc#* and *fs#* bits. You will notice that the names there (after the baud rates) match up with one of the lines of stty output. Didn't I say this was going to be easy?

Note which of those settings are preceded with a – in your stty output. Sum up all those numbers (they are octal). This represents the bits you want to clear, so the result is your *fc#* capability. Of course, remember that you will be setting bits directly after you clear, so you can just use `` `fc#0177777' `` (I do).

Now do the same for those settings (listed in this section) which do not have a – before them in your stty output. In my example the important ones are CS8 (0000060), HUPCL (0002000), and CREAD (0000200). Also note the flags for your baud rate (mine is 0000015). Add those all up, and in my example you get 0002275. This goes in your *fs#* capability (`` `fs#02275` ``' works fine in my example).

Do the same with set and clear for the next section of the include file, "c_lflag bits". In my case I didn't have to set anything, so I just use `` `xc#0157777` ``' and `` `xs#0` ``'.

17.2. Older serial printers that drop characters

Jon Luckey points out that some older serial printers with ten–cent serial interfaces and small buffers *really* mean stop when they say so with flow control. He found that disabling the FIFO in his Linux box's 16550 serial port with `setserial` corrected the problem of dropped characters (you apparently just specify the uart type as an 8250 to do this).

18. What's missing?

Many of the parts for a complete printing system do not exist yet. Projects are underway to address most of these, although most have not yet produced running useful code, and efforts to standardize the necessary protocols and APIs are in their infancy.

18.1. Plumbing

There's a general problem with getting all the parts to talk to one another; especially in a spooler−independent way. This problem manifests itself most noticably in the pathetic application support for control over all the "usual" printing features. There is simply no way for an application writer to get information about printers, jobs, etc; no standardized way to submit jobs; no good way to get job status back; nor even really a standardized way to generate print data (although most of the new desktop systems offer desktop−specific facilities for doing this).

Work to define a sensible API for applications to use for printing will undoubtedly center around Corel's sysAPS library, which provides a rudimentary implementation of several queueing and printer information features.

18.2. Fonts

Font handling on free systems is rather awkward. The display, the printer, the application, and the data files should ideally all have access to the same fonts. Unfortunately this is simply not the case. Plans are afoot to remove font handling from the X server, simplifying part of the problem, but good printer font to application font mapping is still a problem. No project really seems to be underway to address this; currently application writers simply embed their own fonts into printed data.

18.3. Metadata

Applications or spoolers need to learn about printer and driver properties somehow. The current standardized scheme, implemented on Windows, the Mac, and in CUPS, is to use Postscript Printer Description files to drive a programatic interface and user interface. This had trouble for non−Postscript printers, for obvious reasons, so the IEEE's Printer Working Group has a project to specify "Universal Printer Driver Format", or UPDF, files. Thus far they have constructed a sample file in an XML format. The sample file strongly resembles a PPD file, and is missing all sorts of driver and platform specific information; so much so that UPDF is currently not useful. IBM has a fully parameterized driver architecture for OS/2 which is available as free software; once this is released it is bound to be a useful source of ideas or code, and possibly a good enough system to just use outright. Even this system, however, provides no defined mechanism for communicating interesting properties from the driver space up to the application. Some XML format, and/or an API for fetching assorted properties, is bound to appear at some point.

18.4. Drivers

The state of free software drivers is rather poor. Fortunately, several projects are underway to correct this, and impressive results can now be had on printers using that code. The eventual goal seems to be to provide both good drivers and a good framework for the frequently duplicated (and hard!) parts of driver code dithering, for example to be shared.

Printer vendor cooperation will be an important part of achieving this goal. Vendors currently do not provide the minimum documentation necessary to operate their devices well. At the Printing Summit 2000, many vendors were present, and some small headway was made on this point. Vendors are mainly concerned with keeping the dithering and related algorithms secret; these software components are what produces such remarkable inkjet output, and the vendors are of course competing. Those vendors present at the summit should now have a clearer picture of how free software works and what we want from them. This isn't much; bt it sets the stage for future progress.

19. Credits

Special thanks to Jacob Langford, author of `pdq`, who finally gave us something better than the smattering of scripts globbed onto a 20 year old overgrown line−printer control program.

The `smbprint` information is from an article by Marcel Roelofs <`marcel@paragon.nl`>.

The `nprint` information for using Netware printers was provided by Michael Smith <`mikes@bioch.ox.ac.uk`>.

The serial printers under lpd section is from Andrew Tefft <`teffta@engr.dnet.ge.com`>.

The blurb about gammas and such for gs was sent in by Andreas <`quasi@hub−fue.franken.de`>.

The two paragraphs about the 30 second closing_wait of the serial driver was contributed by Chris Johnson <`cdj@netcom.com`>.

Robert Hart sent a few excellent paragraphs about setting up a print server to networked HPs which I used verbatim.

And special thanks to the dozens upon dozens of you who've pointed out typos, bad urls, and errors in the document over the years.