

An introduction to printing with GNU/Linux

Roger Leigh¹

12th May 2003

¹Email: rleigh@debian.org

Contents

1 Introduction	4
1.1 Overview	4
1.2 Notation	4
2 Hardware	5
2.1 Types of printer	5
2.2 Media and Ink	6
2.3 Print resolution on inkjet printers	7
2.4 Connection options	7
2.5 Printer Drivers and Kernel Drivers	8
3 Kernel	9
3.1 Device drivers	9
3.1.1 Parallel port and line printer	9
3.1.2 Serial port	10
3.1.3 USB port	11
3.2 Device nodes and MAKEDEV	11
3.3 Performance tuning	12
3.3.1 tunelp	12
3.3.2 setserial	14
4 Software	14
4.1 Print spoolers	15
4.2 Common spoolers	16
4.3 Line Printing vs. Page Printing	17
4.3.1 Line Printers	17
4.3.2 Page Printers	17
4.4 Adobe PostScript	18
4.5 Ghostscript	18
4.5.1 Purpose	18
4.5.2 Dithering	20
4.5.3 IJS	20
4.5.4 Usage	22
4.6 Overview of printing with CUPS	22
4.7 Command-line interaction	25
4.8 Graphical interfaces	27
4.9 CUPS administration using the web interface	27
4.9.1 Adding a new printer	28
4.9.2 Configuring a printer	29

<i>CONTENTS</i>	2
4.9.3 Starting and stopping printing and spooling . .	29
4.9.4 Using the printer	29
4.9.5 Classes	29
4.10 CUPS configuration	30
4.11 CUPS logs	30
4.12 PPD files	31
4.13 CUPS drivers	31
4.14 LPRng <i>/etc/printcap</i>	31
5 Integration with MS Windows	32
6 Summary	33
7 Further reading	33
8 Acknowledgements	34

List of Figures

1	Kernel device drivers and software printer drivers . . .	8
2	Parallel port signal timings	13
3	A single print queue	15
4	Multiple print queues	16
5	PostScript artwork	19
6	Dither algorithms	20
7	Black and white dithering	21
8	Standard UNIX Input/Output streams	23
9	The CUPS filter chain	24
10	Controlling a print queue	26

List of Tables

1	parport and lp kernel options	9
2	serial kernel options	10
3	usb kernel options	11
4	Device naming schemes	12
5	tunelp options	13
6	setserial options	14
7	lpr options	25
8	lp options	25
9	lpadmin options	27

1 Introduction

1.1 Overview

My first experience of printing with GNU/Linux was in 1999, when I spent about two weeks attempting to set up LPRng with my EPSON Stylus inkjet printer. Most of this time was spent reading through incomplete or inaccurate documentation trying to figure out how everything fitted together, as well as manually editing `magicfilter` filter scripts. This was certainly not user-friendly, and took a lot of time I could have been spending doing more useful things, like actually doing and printing my work! Once set up, this system worked unaltered for a further three years.

Since that time, there have been great advances in ease of use, especially in distributions like *Mandrake*, *RedHat* and *SuSE*, all of which have nice graphical configuration tools. Today, there are also much better printer drivers than there were then, supporting a much wider range of printers, such as *Gimp-Print*, *hpjjs* and *Omni*, amongst many others.

This document is based on the slides I wrote for the *West Yorkshire Linux User Group* (WYLUG) meeting on the 12th May, 2003. It is a general introduction to how printing on GNU/Linux (and UNIX) systems works, with particular emphasis on CUPS.

I would appreciate any feedback from users of this document, in particular if there are any inaccuracies or mistakes that require correction.

1.2 Notation

This document uses several different typefaces to refer to different things:

<i>myfile.ps</i>	Slanted text refers to filenames, or manual pages, such as <i>ls(1)</i> .
<code>/usr/bin/ls</code>	Typewriter text refers to the names of commands, or the output from programs on a terminal.
ls -l	Bold typewriter text refers to user input.
printer	Sans-serif text refers to the name of a kernel driver or subsystem.

2 Hardware

This section covers factors to consider when purchasing a new printer, and how the printer hardware works with a GNU/Linux system.

2.1 Types of printer

There are many different types of printer, each having its own pros and cons. The print quality is often the deciding factor, but running speed, noise level, types of stationary it will handle and ongoing running costs are also important to consider.

- **Impact:** dot matrix and daisy-wheel
 - slow compared to most other types of printer.
 - dot matrix output is of poor quality, but is excellent for labels, continuous stationary and forms.
 - daisy-wheel output is of letter quality, but the character set is limited and fixed.
 - print on most types of stationary
 - expensive, but running costs are very low.
 - can be very noisy.
- **Ink-jet**
 - reasonably fast.
 - output is of intermediate to high quality, and colour is usually available.
 - excellent for photographic printing, with high quality papers and ink-sets available, including pigment inks.
 - usually use cut sheet stationary, although some models have roll feed.
 - can be very cheap, but running costs are very costly.
- **Laser**
 - very fast.

- output is of high quality, though most models only offer black and white (monochrome) output.
 - not known for good reproduction of images or half-tones, although this depends on the printer.
 - use cut sheet stationary.
 - usually expensive, with fairly low running costs, though not as cheap as a dot matrix.
- **Win-Printers** (GDI printers)
 - may or may not be supported. Like Win-Modems, they are inefficient, requiring a lot of CPU time to compensate for a lack of on-board intelligence. Flog it to an MS Windows user, and buy a real printer!
 - **Others:** e.g. dye-sublimation, linotronic,
 - these are generally expensive devices, which are used for certain specialist purposes.

A mostly comprehensive list of all printers supported by current GNU/Linux printing software is at <http://www.linuxprinting.org>, the definitive GNU/Linux printing site. This site lists all the printers known to work with GNU/Linux, and details the drivers that work with each printer, and recommends which driver should be preferred.

2.2 Media and Ink

The media (paper) and ink used make a great difference to the lifetime of a printout. This is not important for casual letters, but for many businesses, there are legal requirements to archive their files for many (at least 15) years. Many people and companies also wish to keep their work indefinitely.

Impact printers These use permanent ink or carbon ribbons. The ink or carbon is impacted upon the paper by pins or type characters like a typewriter, and hence these do not generally run, fade, or fall off the paper.

Ink-jet printers The ink used is usually water-based, so can run if damp. The inks are also prone to fading in sunlight and over time. Special archival inks are available, and also permanent inks. These are expensive, and the solvent can evaporate easily, clogging the print-head.

Laser printers Toner is heat-sealed to the paper, and will not run. However, it is prone to flake off, especially if the paper is creased or abraded, so may not be suited for very long-term archival purposes.

In conclusion, in order of *decreasing* durability:

- impact
- laser
- ink-jet

This is, of course, very general. The durability is highly dependent on the specific printer, paper and ink used.

2.3 Print resolution on inkjet printers

In general, the higher the resolution, the higher the print quality. However, larger resolutions use greater quantities of ink and are much slower.

The resolution chosen is a tradeoff between print quality and print speed.

- Text is fine at 360 or 720 DPI.
- Using plain paper, there little improvement in quality over 720 DPI.
- For high-resolution graphics and photo printing, use special photo papers.

2.4 Connection options

Printers must have some method of communication with a computer, either directly through an interface on the computer, or through a network connection in the case of high-end networked printers. Common methods are:

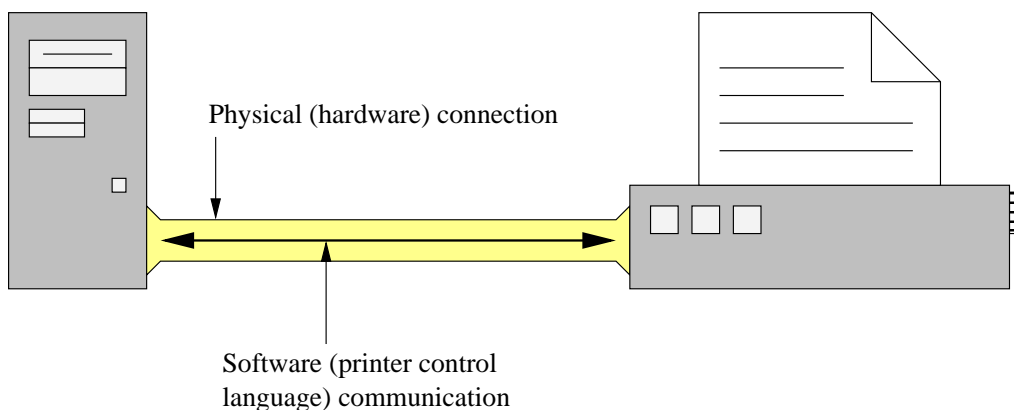


Figure 1: Kernel device drivers and software printer drivers

- Parallel
- Serial
- USB
- FireWire
- Network (LPD/IPP/SMB)

Serial interfaces are common on typewriters and old dot matrix printers, while parallel interfaces are in wide use on all types of printer from dot matrix to lasers. On modern printers, a USB interface is increasingly common, sometimes alongside a parallel interface. Some high-end inkjet printers apparently have FireWire interfaces.

Network printers generally run a *print daemon*, just like a print server. This is usually a UNIX LPD (*Line Printer Daemon*), although nowadays IPP (*Internet Printing Protocol*) may also be seen. SMB (*Server Message Block*) may also be used on Windows networks.

2.5 Printer Drivers and Kernel Drivers

There is often confusion about what a “printer driver” actually is. The confusion is not helped by MS Windows calling lots of things “drivers” that really aren’t.

A kernel driver drives the physical hardware communications device (parallel, USB etc.). The “printer driver” produces the control codes that travel across the communications link to the printer.

“CML” option	menuconfig option
CONFIG_PARPORT	Parallel port support→Parallel port support
CONFIG_PARPORT_PC	Parallel port support→PC-style parallel port
CONFIG_PARPORT_1284	Parallel port support→IEEE 1284 transfer modes
CONFIG_PRINTER	Character devices→Parallel printer support

Table 1: parport and lp kernel options

This scheme is illustrated in Figure 1. The “printer driver” is typically a user-level (non-kernel) program or programs.

3 Kernel

Before any software can make use of an external device, the kernel must have a driver for that device. For printers, the kernel must have support for the connection method one is using (e.g. parallel, USB), either built-in statically or available as a module.

The device may require further tweaking to make it work, or perform optimally.

Most modern GNU/Linux distributions come with a generic kernel with modular support for all the hardware that will typically be used. This section covers the necessary options needed if building a kernel “by hand”, which can be skipped if one does not wish to build a kernel.

3.1 Device drivers

3.1.1 Parallel port and line printer

The parallel port line printer device requires several kernel drivers, shown in Table 1. The parport driver provides generic parallel port support, and the lp device provides support for line printers attached to a parallel port.

The modules stack like this:

“CML” option	menuconfig option
CONFIG_SERIAL	Character devices→Standard/generic [...] serial support

Table 2: serial kernel options

lp	parport_lowlevel
	parport

The lp device *requires* the parport driver to be loaded first, so it can use its services. parport_lowlevel is *platform-specific* parallel port support, required by the generic parport layer. This requires that the platform-specific driver is *aliased* to parport_lowlevel (see below).

I/O base and IRQ setup To enable interrupt-driven (as opposed to the default, polling) mode, one needs to add some entries to */etc/modules.conf*. There may be distribution-specific ways to modify this file. On Debian systems, edit */etc/modutils/arch/i386* for i386 machines, and run */sbin/update-modules*.

```
alias parport_lowlevel parport_pc
options parport_pc io=0x0378 irq=auto
```

The first line above makes a request for the parport_lowlevel module refer to the parport_pc module instead. The second line enables interrupt-driven mode automatically, using 0x0378 as the I/O base address (*Documentation/parport.txt* lists the common addresses). On the author’s system, this is assigned to IRQ 7, which is the most common IRQ for a parallel port.

Note: 2.0 kernels do not support bidirectional communications. They do not have a parport layer either. 2.2 kernels use the option CONFIG_PRINTER_READBACK instead of CONFIG_PARPORT_1284 for bidirectional mode.

3.1.2 Serial port

Serial communications are very simple to set up, since there is just one options to enable, shown in Table 2.

“CML” option	menuconfig option
CONFIG_USB	USB support→Support for USB
CONFIG_USB_DEVICEFS	USB support→Preliminary USB device filesystem
CONFIG_USB_[EUO]HCI	USB support→USB Host Controller Drivers
CONFIG_USB_PRINTER	USB support→USB Printer support

Table 3: usb kernel options

3.1.3 USB port

USB is quite complex, and requires a module specific to the system’s *host controller*. Table 3 lists all the options required for USB printer support. `CONFIG_USB_DEVICEFS` is optional, but highly recommended.

USB support in Linux is fairly new, and there are still teething problems. For example, if the modules are inserted in the wrong order, or one is missed out (I have not yet identified the cause of the problem), data sent to `/dev/usb/lp0` will not be printed, or generate an error, it will just be silently discarded. This can make USB problems difficult to diagnose if one is unaware of this silent data loss. Using `usbmgr` is an easy solution.

usbmgr The `usbmgr` daemon uses the **usbdevfs** filesystem to handle hot-plug events and insert and remove the correct modules. `usbmgr` requires `/etc/fstab` to have the following entry:

```
# <fs> <mount point> <type> <options> <dump> <pass>
none /proc/bus/usb usbfs defaults 0 0
```

3.2 Device nodes and MAKEDEV

A device node must exist for each device. The device naming for common device types is shown in Figure 4.

If the device nodes don’t exist, then `MAKEDEV` can create them. For example, to create the USB device nodes (as the root user):

```
# cd /dev
# ./MAKEDEV usb
```

Device	Node name
Parallel	<code>/dev/lp0 ... /dev/lpn</code>
Serial	<code>/dev/ttyS0 ... /dev/ttySn</code>
USB	<code>/dev/usb/lp0 ... /dev/usb/lpn</code>
Network	N/A

Table 4: Device naming schemes

3.3 Performance tuning

Although the default behaviour of the device drivers means that they will work perfectly in nearly all cases, it might be the case that some “tweaking” is required before they work. In addition, it is possible that the performance is non-optimal, and can be increased by a significant amount.

3.3.1 `tunelp`

The `tunelp` command may be used to fine-tune the operation of the `lp` device. Parallel ports work by sending 8 bits in parallel. There is a complex pattern of signals used to send each 8 bits of data, illustrated in Figure 2. The timing requirements may be different for other printers, and the speed of data transmission may be increased by shortening the delay timings, providing the printer hardware will allow it.

`tunelp` may be used to set various timing options for each `lp` device. A list of the most commonly-used options is given in Table 5; see the *tunelp(8)* manual page for a complete description. Note that on modern kernels (2.2 or newer), the `-i` option does not work (this is now part of the `parport` layer). Other options change the behaviour of the device when errors occur.

Interrupt-driven operation imposes less CPU overhead, but can flood the system with interrupts if used heavily. If running a printer continuously, polling is better, if slower, but will waste CPU cycles otherwise.

Some printers will need the options tweaking to work, but most will work just fine without.

My OKI Microline 321 Elite (9-pin dot matrix printer) wouldn't work with Linux, even with `tunelp` tweaking, but worked without any problems using 4.6-FreeBSD and MS Windows98. This was

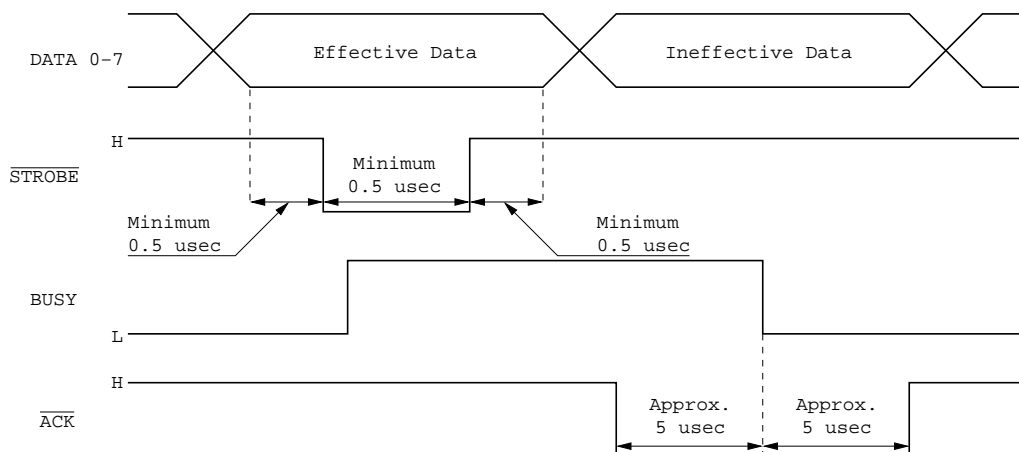


Figure 2: Parallel port signal timings. When the computer tries to send some data to the printer, there is a set sequence of events. The computer sets the 8 data pins, and then brings the $\overline{\text{STROBE}}$ line low. The printer responds by taking BUSY from low to high, then bringing the $\overline{\text{ACKNOWLEDGE}}$ line low. Note the timing requirements.^a

^aThese timings are for an EPSON FX-compatible dot-matrix printer. Modern printers may have different requirements for high-speed data transfer rates.

Option	Description
-i	IRQ
-c	Times to try sending before sleeping
-t	Wait time if no char taken
-w	μ secs to wait while strobing
-a on off	Abort on printer error
-o on off	Check device is OK on open().
-s	Get printer status
-T	Trust IRQ (interrupt→immediate char send)
-r	Reset port

Table 5: tunelp options

Option	Description
IRQ	Interrupt Request Number
I/O Port	Base I/O Port
Baud rate	Transmission data rate
Delay timings	Various delay parameters
Port locking	Prevent opening, or notify other processes
Latency minimisation	Utilises more CPU cycles

Table 6: setserial options

due to the experimental parport options `CONFIG_PARPORT_PC_FIFO` and `CONFIG_PARPORT_PC_SUPERIO`. After removing these options, the printer worked perfectly.

3.3.2 setserial

Like the `lp` device, the `serial` device may also be fine-tuned. The most commonly-used options are listed in Table 6. These options may also be listed in `/etc/serial.conf` and loaded at boot-time by `/etc/init.d/setserial`. For example:

```
/dev/ttyS0 uart 16550A port 0x03f8 irq 4 baud_base \
 115200 spd_normal skip_test session_lockout
/dev/ttyS1 uart 16550A port 0x02f8 irq 3 baud_base \
 115200 spd_normal skip_test session_lockout
```

4 Software

Once the kernel configuration is complete, user-level programs can communicate with the printer. The first step is to check that the kernel device driver for the printer is working.

A line printer can be tested like this:

```
# echo 'Testing, testing, 1 2 3' > /dev/lp0
```

or like this:

```
# cat testfile > /dev/lp0
```

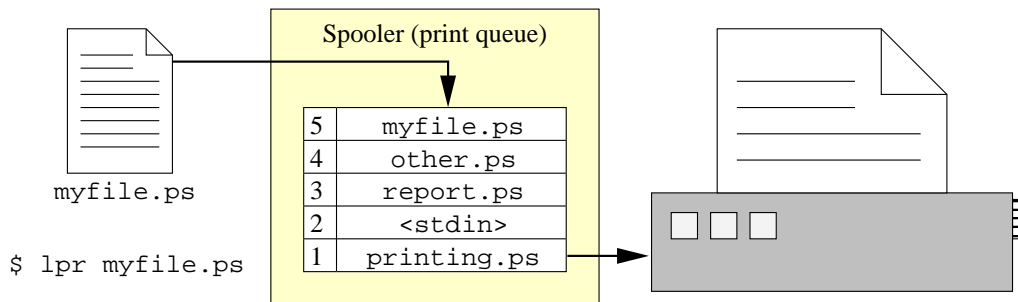


Figure 3: A single print queue

Both of these commands should result in some output from the printer within a few seconds. A page printer will need a form feed code (FF, ASCII 0x0C) to make it print and eject the page.

4.1 Print spoolers

One can send data for printing directly to the device node, but this causes several problems:

- While waiting for the printing to complete, most programs will “block”, becoming unusable.
- More than one program might try to send data at the same time.
- More than one user might want to use the printer.
- Allowing users to use the device node could have security and resource sharing issues.

Print “spoolers” were created to solve these problems.

A **spooler** queues up the **print jobs** (files to be printed), then prints each in turn. An example of how a simple spooler works is shown in Figure 3. (“SPOOL” stands for “Simultaneous Peripheral Operation On-Line”, although this is thought to have been invented after the word itself!)

With a print spooler, printing is “instant”, since the program doesn’t have to wait for the printing to complete. This was even a feature of many DOS programs, e.g. DR GEM and MS Windows, and word processors such as Locomotive LocoScript.

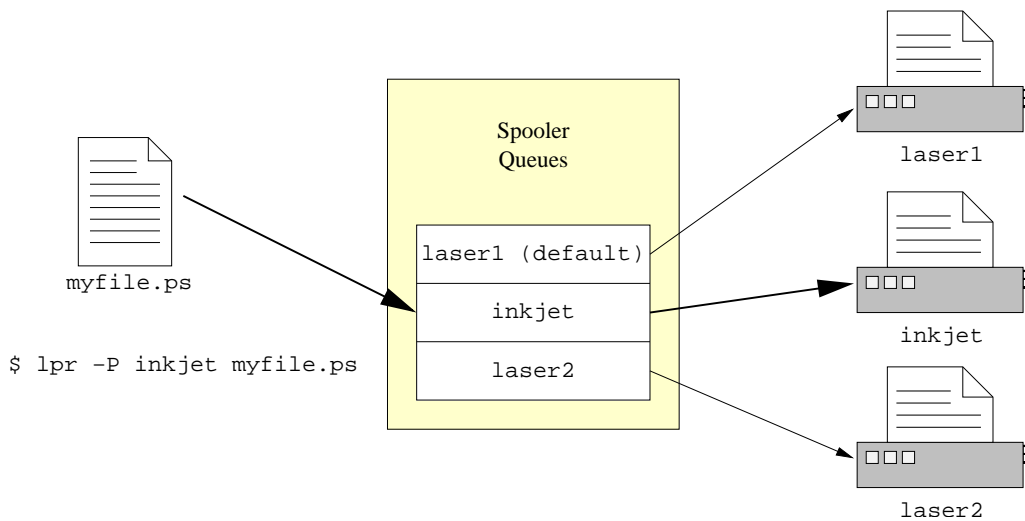


Figure 4: Multiple print queues

On a multiuser system, spooling allows efficient sharing of resources, since a printer can be used simultaneously by many users, and they can be located in central areas. Advanced spoolers allow diversion of jobs to different printers, remote printing, disabling of printing and queueing, and a host of other features.

Spoolers can consist of multiple **queues**, as shown in Figure 4. Each queue has a name associated with it, and by sending the print job to a queue, it will be directed to a particular printer. If no queue is specified, the job will be sent to the **default** print queue.

4.2 Common spoolers

There are many different spooling systems available. These are the most commonly used:

- CUPS

The *Common UNIX Printing System* is the default spooler on most modern GNU/Linux distributions. It is by far the most user-friendly, and the easiest to set up.

- LPRng

“LPR-next generation” is a complete rewrite of LPD. It is robust and widely used, and should be used in preference to any LPD variant.

- LPD

“Line Printer Daemon”. There are numerous variations of LPD, all of which have various deficiencies and security issues. Use LPRng instead.

If you are unsure of which system to use, I would recommend CUPS because of its ease of set up and user-friendliness. If you wish to use an LPD spooler, I would recommend LPRng.

4.3 Line Printing vs. Page Printing

Printers print in either a line- or page-oriented manner. Originally, all printers were line printers. Today, page printers are the norm.

4.3.1 Line Printers

These print on a character-by-character or line-by-line basis, like a typewriter or TTY. This includes all impact, and most ink-jet printers.

The ESC/P language used by Epson (and compatible) printers falls into this category. It can change the character pitch, line spacing, bold/italics/underline and sub- and super-script attributes. However, it also includes simple bitmap graphics modes.

Most DOS word processors were geared to line printer output.

The ESC/P2 language used by Epson *Stylus* printers has much more complex graphics capabilities, and is a new version of ESC/P with many new commands. This enables printing at very high resolutions, using multiple colours and drop sizes. However, some processing is offloaded to the computer, such as softweaving (calculating the order in which the lines of the image will be printed on the paper).

4.3.2 Page Printers

The smallest unit of output is one page. An image of a page is built up, then output in one go. This includes all laser printers, and some high-end ink-jet printers.

A **Page Description Language** is used to describe what is to be put on each page. Adobe *PostScript* (PS) and HP *Printer Control Language* (PCL) are Page Description Languages commonly used by page printers.

4.4 Adobe PostScript

PostScript is a full (stack-based) programming language, used to describe the appearance and placement of text and graphical lines, shapes and images on the printed page. It is device-independent—any PostScript printer will be able to print a normal PostScript file.

Figure 5 is an example of what can be done with PostScript. The picture is made entirely with PostScript line drawing and shading commands.

The language is interpreted, and the interpreter is usually a Raster Image Processor (RIP). The RIP converts the PostScript into a large picture—a bitmap or pixel map (pixmap).

Hardware RIP PostScript printers have a built-in (hardware) RIP; they can print the PostScript directly.

Software RIP Other printers cannot print PostScript. They need it converting into a language they can understand, such as ESC/P2 or PCL. On GNU/Linux, GNU Ghostscript is used for this task.

4.5 Ghostscript

4.5.1 Purpose

GNU Ghostscript is a “software RIP”. It converts PostScript into other formats and languages, such as:

- Bitmap or pixmap graphics
- Printer control languages, e.g. ESC/P2 or PCL

It can be thought of as two separate parts:

PS→raster image	raster→printer format
-----------------	-----------------------

The first half converts the PostScript into a raster image (a big bitmap/pixmap picture). The second half translates this picture into a printer control language, graphic image format, or whatever one likes, depending upon which driver one selected. All of the available drivers can be listed with the command:

```
$ gs -h
```



Figure 5: PostScript artwork. This picture is constructed using PostScript line drawing and shading commands.

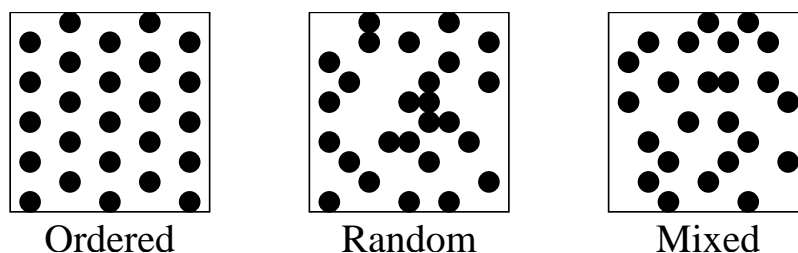


Figure 6: Dither algorithms. All of the above patterns have exactly the same number of dots. Ordered dithers use ordered placement of dots to represent different shades, but the regularity is not pleasing to the eye. Truly random dithers introduce “clumps” and “gaps” which are further dissatisfying. The best dithers are random, but take care to remove clumps and gaps, while not introducing too much order.

Ghostscript traditionally has all its drivers built in, as well as being licensed from Aladdin Enterprises. The licensing arrangements, while Free, made it hard to improve, or add new drivers, which generally required manually patching the source.

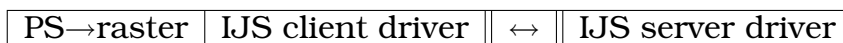
4.5.2 Dithering

Dithering converts shades of many colours into just a few. For example, most colour inkjet printers have just four (cyan, yellow, magenta and black(keyline)). This allows just a few colours to make many, by clever patterning of the coloured dots. Some of the different ways of dithering are shown in Figure 6. An example of black and white dithering of a simple coloured image is shown in Figure 7.

Because the standard Ghostscript dither algorithms are lacking, drivers like *Gimp-Print* use their own, much better, equivalents.

4.5.3 IJS

The **IJS** (InkJet Server) driver has been developed to allow the printer driver to run as a co-process:



IJS is a client-server raster image transport protocol, used for communication between the two processes. Ghostscript is the IJS



Figure 7: Black and white dithering. The colours and shades of the top picture are reduced to patterns of black dots in the bottom picture.

client, and the printer driver program is the *server*. The client and server communicate using UNIX pipes, like pipes in a shell script, talking the IJS protocol.

This “fixes” the problem of Ghostscript being a monolithic program, allowing new drivers to be easily added and updated. `libijs` is MIT-X11 licensed, allowing use in both Free and proprietary software. IJS servers in common use are `hpijs` and `ijsgimpprint` (which supersedes the Ghostscript `stp` driver).

4.5.4 Usage

Examples of usage:

```
$ /usr/bin/gs -sDEVICE=stcolor -r360 -q -dSAFER
-dNOPAUSE -sOutputFile=/tmp/tiger.raw tiger.ps
GS>quit
```

This uses the `stcolor` driver to print the PostScript file `tiger.ps` at 360 DPI, and save the ESC/P2 output in the file `/tmp/tiger.raw`. This could then be copied directly to the printer device.

```
$ /usr/bin/gs -sDEVICE=ijjs
-sIjsServer=/usr/bin/ijsgimpprint
-sDeviceManufacturer=EPSON -sDeviceModel=escp2-c60
-sPAPERSIZE=a4 -sIjsParams='Quality=360sw' -dSAFER
-dNOPAUSE -sOutputFile=/tmp/tiger.raw tiger.ps
GNU Ghostscript 6.53 (2002-02-13) Copyright (C) 2002
artofcode LLC, Benicia, CA. All rights reserved. This
software comes with NO WARRANTY: see the file COPYING
for details.
GS>quit
```

This command does exactly the same thing, but instead uses the `ijjs` driver, coupled with the `ijsgimpprint` server driver. This produces a much higher quality printout.

4.6 Overview of printing with CUPS

The original print spoolers did not do much other than store the job, then send it to the printer. Nowadays, spoolers are used to do complex filtering of print jobs, enabling non-PostScript printers to print PostScript.

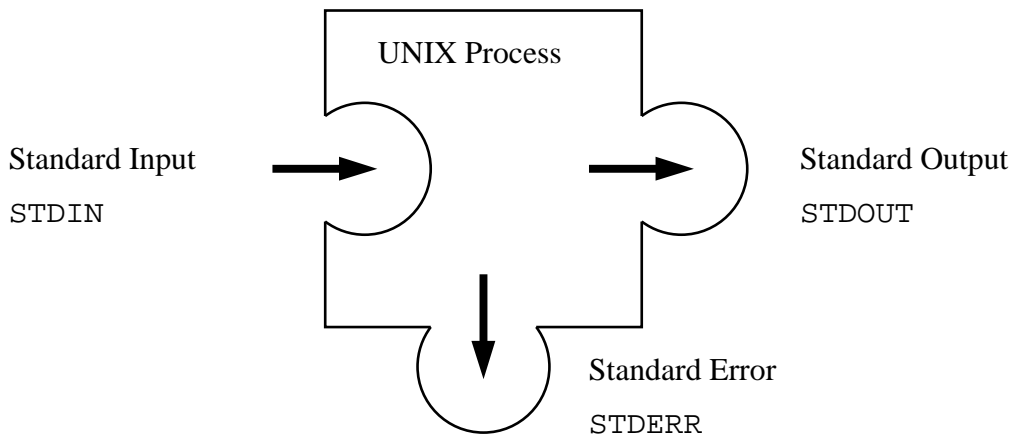
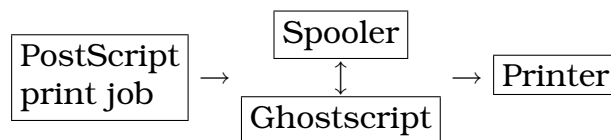


Figure 8: Standard UNIX Input/Output streams. A UNIX process is shown as a jigsaw puzzle piece. The “hole” represents input whilst the “tabs” represent output. Many processes can be connected together by joining them up where they will fit together.

Printing to a PS printer The spooler queues the job, then sends it straight the the printer.



Printing to a non-PS printer The spooler queues the job as before, but also sends it through Ghostscript, before sending the Ghostscript output to the printer.



Printing with CUPS Every UNIX process has three file descriptors (I/O streams), as illustrated in Figure 8. For each print job, CUPS runs a “filter chain”, shown in Figure 9. This converts a PostScript print job to a raster image, then converts the image to printer commands, then outputs the print data. Each part of the chain does a specific task:

cupsd The CUPS daemon receives and spools the print job.

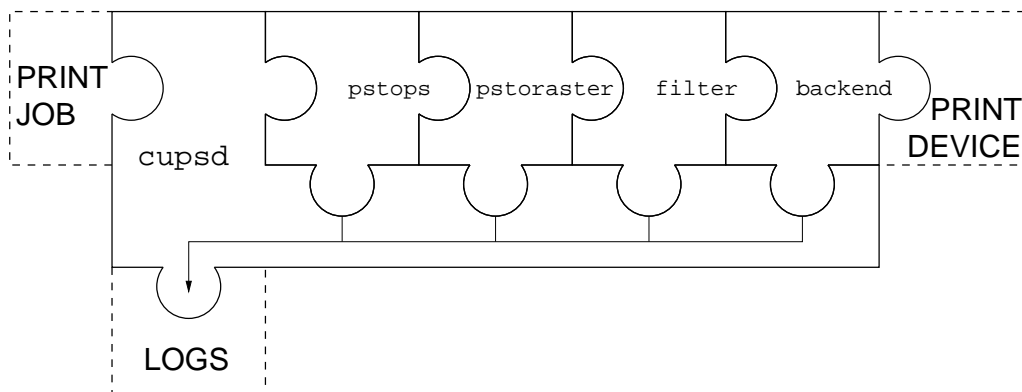


Figure 9: The CUPS filter chain. CUPS spools incoming print jobs. When they are to be printed, `cupsd` sets up a *filter chain* consisting of several programs connected by pipes (the standard output of one connected to the standard input of the next; see Figure 8). The standard error of each is connected to `cupsd`, which filters the error messages according to the `LogLevel` setting and stores them in its log files. The final process in the filter chain, the `backend`, talks to the printer itself.

pstops Counts the number of pages, and can also do *n*-up printing (printing multiple pages, reduced in size, side by side on a single page).

pstoraster Converts the PS to a raster image (the first “half” of Ghostscript).

filter Converts the raster image to the native printer language. This could be a CUPS driver (`rastertoepson`), or third-party driver like *Gimp-Print* (`rastertoprinter`), *Foomatic* (an interface to Ghostscript and IJS drivers) or *Omni*.

backend Sends the data to the printer. There is a backend for each device type, e.g. `parallel`, `usb`, `ipp` or `lpd`. They can detect error conditions, such as the printer being not ready or out of paper, and then automatically disable the queue.

The filters are located in `/usr/lib/cups/filter` and the backends are located in `/usr/lib/cups/backend`. New filters and backends can easily be added.

Option	Description
<code>-P queue</code>	Queue name
<code>-r</code>	Remove <i>myfile.ps</i> when finished (!)
<code>-o raw</code>	Print “raw”—no filtering

Table 7: `lpr` options

Option	Description
<code>-dqueue</code>	Queue name

Table 8: `lp` options

4.7 Command-line interaction

Traditionally, AT&T System V and BSD UNIX had different spoolers, which had different commands and environment variables. CUPS and LPRng include both sets, so either may be used. The System V versions are in parentheses.

Printing The `lpr` (`lp`) command queues a file for printing:

```
$ lpr -P lp myfile.ps
```

The common options for the `lpr` command are shown in Table 7.

```
$ lp -dlp myfile.ps
```

The common options for the `lp` command are shown in Table 8.

The `-P` and `-d` options are common to all the BSD and System V commands, respectively.

GNU `a2ps` can be used to convert plain text to PS for printing. This can give nicer results than just printing the plain text “as is”.

View status The `lpq` (`lpstat`) commands show the files which have been queued, along with the owner (username) of the job and the job ID.

```
$ lpq
```

```
Rank    Owner   Job   File(s)      Total Size
active  roger   268   myfile.ps    141312 bytes
```

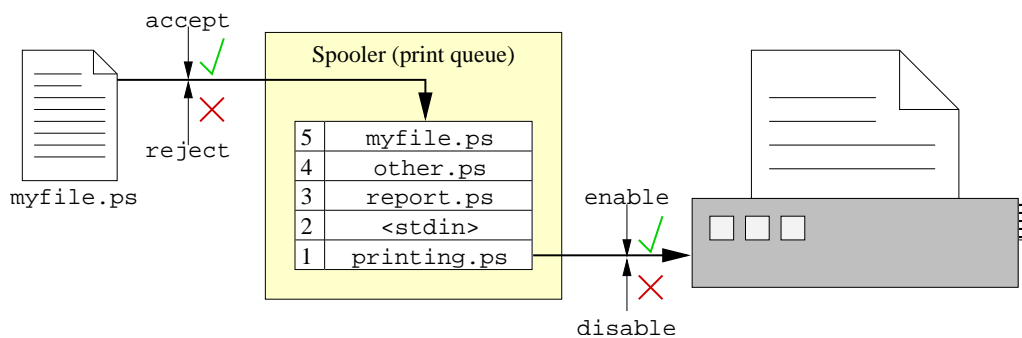


Figure 10: Controlling a print queue. The `accept` and `reject` commands enable and disable job queuing for a destination, respectively. `enable` and `disable` start and stop a printer or class, respectively.

```
$ lpstat
```

```
c60-268 roger 141312 Wed May 7 16:02:57 2003
```

Remove/Cancel jobs The `lprm` (`cancel`) commands remove a job from the print queue. They take the job ID from `lpq` (`lpstat`) as the job to delete.

```
$ lprm 268
```

```
$ cancel 268
```

```
$ cancel -dlp -a
```

The latter command cancels all jobs on the `lp` queue.

Administration As part of one's role maintaining the printing system, one may need to take a printer out of service for a while, for example to add ink or toner, or to add more paper. This should not cause disruption to users of the system, who do not want to lose their printing. Figure 10 illustrates the use of the `accept`, `reject`, `enable` and `disable` commands to control spooling and printing.

The `lpadmin` tool may be used to configure CUPS printers and classes, and set the default printer and class. It is used like this:

```
$ lpadmin [options] destination
```

Table 9 lists the commonly used options. Most of these have equivalents in the web interface (see below), apart from setting the default queue, and setting per-user quotas.

Option	Description
-p <i>printer</i>	Configure (add) a printer
-D <i>info</i>	Textual description of printer
-L <i>loc</i>	Textual location of printer
-P <i>PPD</i>	PPD file for printer
-d <i>queue</i>	Default queue name
-E	Enable printer and accept jobs
-c <i>class</i>	Add printer to <i>class</i>
-r <i>class</i>	Remove printer from <i>class</i>
-o job-k-limit= <i>value</i>	Set the per-user data size limit
-o job-page-limit= <i>value</i>	Set the per-user page limit
-o job-quota-period= <i>value</i>	Set the quota period (seconds)

Table 9: lpadmin options

4.8 Graphical interfaces

There are several graphical `lpr/lpq/lprm` replacements:

- KPrinter
- qtcups
- XPP
- GTK-lp

All of these are approximately equivalent in functionality; they differ in which GUI toolkits were used in writing them.

`cupsdconf` is a KDE GUI tool for editing `cupsd.conf`. It plugs into the KDE “Control Centre” application, and provides a nice easy method of configuring CUPS.

4.9 CUPS administration using the web interface

CUPS uses a protocol called the *Internet Printing Protocol* (IPP). IPP is HTTP on port 631. IPP is used by all of the client applications to communicate with the CUPS daemon, `cupsd`, for example to send a print job, remove a job, and get information about the available print queues and their descriptions.

Because CUPS is an IPP server, it can also act as an HTTP (web) server. If one directs a web browser at `http://localhost:631/`,

one can administer the CUPS server using its web-based management interface. This is an alternative to using `lpadmin` and the other command-line tools.

To start administering the server, select the top link, “Do Administration Tasks”. You should be prompted for a username and password; use `root` and the password for `root`.

4.9.1 Adding a new printer

From the “Admin” page (<http://localhost:631/admin>), choose the “Add Printer” option. You will now be prompted to enter a Name, Location and Description. The Name is the print queue name, and must be a single word, preferably in lowercase. The other two items are optional, merely for informational purposes. These are the location of the printer in the building (so users can find it), and a description of the printer.

The next step is to select a Device (that is, a *backend*), which is the connection method used to connect the printer to the computer. All the available methods should be in the list.

If one is using USB, and no USB devices appear, make sure that the `usb` modules (including the host controller module), along with the printer (USB printer) module are inserted into the kernel *and* that the printer is connected and powered on. Then, restart `cupsd`. This is because `cupsd` detects which backends are available at startup, and may think that there are no USB devices present.

The next step is to select the Make (manufacturer) of the printer. If it is not listed, try a compatible model of a different make.

Next, choose the specific model of printer, or a compatible model. There may be multiple drivers for a single model, in which case check the <http://www.linuxprinting.org/> website to see which is recommended.

Now the queue is set up and ready to print!

If there are multiple queues, one can set the default queue with the command:

```
# lpadmin -d queuename
```

Now all print jobs will be sent to *queuename* unless the user specifies a different queue.

4.9.2 Configuring a printer

Select the “Printers” option from the bar at the top of the page (<http://localhost:631/printers>). All the printer queues available on the system will now be listed. Each should have a “Configure Printer” button. By selecting this, you are taken to a configuration page. On this page, you can change all of the options the printer driver provides, such as the resolution, paper size, banners, and so on. This sets the defaults for the queue, although users may override them for individual print jobs.

4.9.3 Starting and stopping printing and spooling

Spooling may be allowed or denied using the `accept` and `reject` commands, and printing may be started and stopped using the `enable` and `disable` commands. The web interface also provides equivalent control for each queue. In the “Printers” list (<http://localhost:631/printers>), each printer has buttons to start and stop the printer, and `accept/reject` jobs.

4.9.4 Using the printer

Now there is a print queue on the system, all of the command-line tools mentioned above in Section 4.7 may be used.

4.9.5 Classes

Classes are collections of print queues. They may be created with `lpadmin` (as discussed above), or with the web interface. Select the “Classes” option from the bar at the top of the page (<http://localhost:631/classes>). Like the “Printers” page, this shows the available classes, along with a similar set of options.

Choose “Add Class” to add a new class. One will be asked for a Name, Location and Description, as for adding a new printer. Next, select which print queues one wishes to be part of the class.

Now the class exists, a print job can be sent to it just like to a normal print queue. It will split all of the print jobs it receives between all the queues that are members of the class. If a printer is disabled, for example because it is out of paper, the jobs will stop being sent to that member. This means that users still get their printouts when a printer fails for some reason.

4.10 CUPS configuration

CUPS has several configuration files, each of which may be manually edited.

cupsd.conf The configuration file for `cupsd`. This file is very well commented, having a description for each configuration option, usually along with default and allowed values. If one is experiencing problems with CUPS, make sure that *LogLevel* is set to *info* or even *debug*, or else the log files may not have enough detail to show the problem.

client.conf The configuration file for the client programs like `lpr`, `lpq` and `lprm`. This file is used so that they know where the CUPS server is located (which might be another machine on the network).

printers.conf A list of all the print queues, and their status. This should not normally need to be edited by hand, because the `lpadmin` and web administration tools update it automatically.

classes.conf A list of all the classes. This has the same format as *printers.conf*, and should also not need editing by hand.

mime.convs CUPS prints PostScript. This file contains instructions for converting other file formats to PostScript, such as image files and HTML, using MIME types.

4.11 CUPS logs

During its operation, CUPS writes several log files, located under `/var/log/cups`:

access.log This file records every access to the server, both from the web interface as well as print jobs sent by `lpr` etc..

error.log This is the main CUPS log file. `cupsd` writes messages about everything it does to this file, but by lowering the severity of the *LogLevel* option, less information is recorded.

page.log This is a record of the page accounting. For implementing a different accounting policy to the one built into CUPS, this file could be used to read the accounting data.

CUPS performs its own log rotation, based on maximum log file sizes you specify in *cupsd.conf*.

4.12 PPD files

A *PostScript Printer Description* (PPD) file describes the capabilities of a printer. CUPS has a PPD file for each print queue, and uses it to find (and store) the configuration options for that printer.

CUPS uses some extensions to the PPD specification to allow non-PS printers to use PPD files. This is so CUPS knows which filters to run for each printer model.

The PPD files can be read by IPP-using applications, such as *kprinter*, over the network, allowing each print job to be customised by the user. On MS Windows, the PPD file is used in conjunction with the standard PostScript printer driver in the same way, so one can tweak all the CUPS PPD configuration options from a print dialogue!

4.13 CUPS drivers

CUPS comes with some generic printer drivers, and Easy Software also sell commercial drivers (*ESP Print Pro*).

Gimp-Print supports all Epson Stylus printers, Canon BJC, Lexmark, HP PCL laser and ink-jet printers. It allows photo-quality printing on printers that allow it.

Omni supports many older models of printer. It's the print subsystem and drivers from IBM OS/2. The print quality isn't as good as *Gimp-Print*, but it supports many more printers.

Foomatic is a spooler-independent interface to Ghostscript. It is the framework used in the current Mandrake, SuSE and RedHat printer configurators. All the Ghostscript drivers can be used via the *cupsomatic* filter.

There are many other less common drivers available in addition to these. Check <http://www.linuxprinting.org/> to find what the available drivers are for each supported printer.

4.14 LPRng /etc/printcap

LPD daemons use */etc/printcap* to hold the information about each print queue. It's roughly equivalent to *printers.conf*. This

is an example of a `printcap` entry for an Epson Stylus printer:

```
# An LPRng print queue
lp|stcolor660|Epson Stylus Color 660:\
    :lp=/dev/lp0
    :sd=/var/spool/lpd/stcolor660
    :sh:pw#80
    :pl#72
    :px#1440
    :mx#0
    :if=/etc/magicfilter/stp_660
    :af=/var/log/lp-acct
    :lf=/var/log/lp-errs
```

The filter script (`:if=/etc/magicfilter/stp_660`) performs the job of running `ghostscript` to convert the PostScript to the printer's ESC/P2 language. The `:lp` entry is the printer device node, `:sd` is the spool directory and `:mx` is the maximum job size limit (unlimited). All of the options are documented in the *printcap(5)* manual page.

5 Integration with MS Windows

Both CUPS and LPRng can be used in conjunction with Samba to allow printing from MS Windows hosts. This can even extend to automatic downloading of the correct printer drivers for MS Windows!

There are two ways of doing this:

1. Print PostScript from MS Windows, and do all the RIPing on the server, if needed.
2. Use the native MS Windows printer driver, and print "raw" to the CUPS server.

The latter method will put less load on the print server, and may also give better print quality if the MS Windows drivers are of better quality.

6 Summary

- Printing on GNU/Linux works by queueing (*spooling*) jobs, then printing them in turn. There can be more than one queue.
- PostScript is the page description language used for printing. Non-PostScript printers must use a filter to convert the PostScript to their native language.
- There are many printer drivers available. Most are based on Ghostscript, and there are increasingly more native CUPS drivers. However, it is not always easy to know which driver your printer needs, or how to set it up.
- Printing on GNU/Linux has never been easier, compared with the past. However, there is still plenty of room for improvement.

Have fun printing on GNU/Linux!

7 Further reading

- CUPS
<http://www.cups.org/>
- LPRng
<http://www.lprng.org/>
- KDEPrint and CUPS (Kurt Pfeifle's tutorial materials)
<http://www.linuxprinting.org/kpfeifle/LinuxKongress2002/Tutorial/>
- The Linux Printing HOWTO (outdated; update pending)
<http://www.tldp.org/HOWTO/Printing-HOWTO/index.html>
- The Serial HOWTO
<http://www.tldp.org/HOWTO/Serial-HOWTO.html>

8 Acknowledgements

Figure 2 on Page 13 was taken from the *User Instructions* manual for AMSTRAD DMP 3000/3160/3250DI Dot Matrix Printers. I would like to thank AMSTRAD Plc. for their permission to use the diagram in this document. The diagram was redrawn in Xfig from the original manual.