LPRng Reference Manual

24 Sep 2004 (For LPRng-3.8.28)

Patrick A Powell

papowell@lprng.com
AStArt Technologies
6741 Convoy Court,
San Diego, CA 92111
Phone 858-874-6543
Fax 858-751-2435

LPRng Reference Manual 24 Sep 2004 (For LPRng-3.8.28)

by Patrick A Powell

The LPRng; Printing Software consists of the **LPRng** print spooler, the **ifhp** print filter, and the **LPRngTool** graphical user interface.

The **LPRng** print spooler is an enhanced, extended, and portable implementation of the Berkeley **lpr** print spooler functionality. While providing the same interface and meeting RFC1179 requirements, the implementation is completely independent and provides support for the following features: lightweight (no databases needed) lpr, lpc, and lprm programs; dynamic redirection of print queues; printer pooling and load balancing; automatic job holding; highly verbose diagnostics; client programs do not need to run SETUID root; greatly enhanced security checks; load balancing across multiple printers; and a greatly improved permission and authorization mechanism. The source software compiles and runs on a wide variety of UNIX systems, and is compatible with other print spoolers and network printers that use the **lpr** interface and meet RFC1179 requirements. Included in the **LPRng** print spooler distribution is a set of customizable banner page generation programs.

The SVR4 lp and lpstat functionality is provided by a set of emulator programs, and **LPRng** can be easily integrated with the Samba SMB support package. For users that require secure and/or authenticated printing support, **LPRng** supports SSL (using **OpenSSL**), Kerberos 5, MIT Kerberos 4 extensions to LPR, PGP, and simple MD5 based authentication. Additional authentication support is extremely simple to add.

The **ifhp** print filter converts print jobs into formats compatible with PostScript, PCL, text, and other printers and provides diagnostic and error information as well as accounting information.

The ;LPRngTool& Graphical User Interface provides a simple to use configuration and monitoring tool. It allows users to monitor printers and generate printcap entries in a simple manner, as well as providing extensive help and diagnostics.

Table of Contents

List of Tables

Preface

1. Introduction

The **LPRng** Print Spooler provides the essential printing services for UNIX and UNIX-like operating systems. It can be configured to work in small, large, or enterprise level environments. The **ifhp** Print Filter is used with **LPRng** to convert print jobs into a format compatible with a particular printer and the while it may briefly describe the **ifhp** operation, Finally, the **LPRngTool** Graphical User Interface provides an easy to use configuration and monitoring tool for the **LPRng** print spooler.

This document is the basic reference for the **LPRng** print spooler software; the **ifhp** documentation and **LPRngTool** should be consulted for details about their operation.

2. Acknowledgements

I would like to thank all of the **LPRng** users who so relentlessly tried the incredible number of permutations and combinations of printers and software, and whose requests for *just one more feature* led to the development of the software.

3. Shell Prompts

The following table shows the default system prompt and superuser prompt. The examples will use this prompt to indicate which user you should be running the example as.

User	Prompt
Normal user	`%`
`root`	`#`

4. Typographic Conventions

The following table describes the typographic conventions used in this book.

Meaning	Examples
The name of commands, files, and directories. On screen computer output.	Edit your `.login` file. Use `ls -a` to list all files. `You have mail.`

Meaning	Examples
What you type, when contrasted with on-screen computer output.	`% su Password:`
Manual page references.	Use su(1) to change user names.
User and group names	Only `root` can do this.
Emphasis	You *must* do this.
Command line variables; replace with the real name or variable.	To delete a file, type `rm filename`
Environment variables	$HOME is your home directory.

5. Notes, warnings, and examples

Within the text appear notes, warnings, and examples.

Note: Notes are represented like this, and contain information that you should take note of, as it may affect what you do.

WarningWarnings are represented like this, and contain information warning you about possible damage if you do not follow the instructions. This damage may be physical, to your hardware or to you, or it may be non-physical, such as the inadvertent deletion of important files.

Examples are represented like this, and typically contain examples you should walk through, or show you what the results of a particular action should be.

Chapter 1. Introduction

Printing is one of the essential services provided by computer systems. Users want reliable and easy to use methods of printing that require a minimum amount of effort to used and understand. On single user systems with a directly attached printer they perceive that the printing process is simply a matter of *storing* or *spooling* a file, and then transferring it to the printer in a timely manner. In the classical *multi-user* systems, each user expects to share a common printer with one or more users; the print *spooling* system provides arbitration and sharing of the printer among the various users. In a *network* based multi-user system, there may be one or more printers shared by multiple users on many different systems. The print *spoolers* will need to cooperate to provide print services to the users in a simple an predictable manner.

1.1. What is LPRng?

The **LPRng** print spooler software was developed to be robust, reliable, secure, scalable, and portable. It has been used since 1988 in extremely demanding academic printing environments such as University of Minnesota, MIT, and Rutgers, commercial companies such as Dow Jones and Abbot Pharmaceuticals, as well as being distributed with Linux, FreeBSD, and other systems. Each of these environments has a unique set of problems, demanding various configuration and administrative capabilities. For example, the simple single user system with a single or limited number of printers requires easy configuration and simple diagnostic procedures, while the network based printing system requires highly robust error logging, authentication, and failover support. **LPRng** provides a highly flexible configuration system that allows it to perform optimally in all of these environments.

The **LPRng** software has three components: the **lpd** print spooler and the user client applications **lpr**, **lpq**, **lprm**, etc.; the IFHP print filter (**ifhp**) which is used to convert jobs into a suitable for a particular printer, and the the LPRngTool Graphic User Interface (**lprngtool**) which provides a simple and easy to use configuration and monitoring tool for the **LPRng** print spooler.

LPRng mimics many of the features of the *vintage* or *legacy* Berkeley (University of California - Berkeley) Line Printer (LPR) package found on Berkeley derivatives of the Unix operating system. **LPRng** will print a document with little or no knowledge of the content or special processing required to print the document on a stand-alone machine or in a distributed printing environment. New (as compared to Berkeley LPR) features include: lightweight **lpr**, **lpc** and **lprm** programs, dynamic redirection of print queues, automatic job holding, highly verbose diagnostics, load balancing queues; enhanced security (SUID not required in most environments), and easy configuration.

LPRng started life at the University of Waterloo in 1986 as PLP (Public Line Printer), a replacement for the original BSD **lpd** code. This was a one-shot effort by the author, Patrick Powell, to develop freely redistributed code without the restrictions of the BSD/AT&T license and would allow non-licensed sites to fix and patch problems. From 1988 to 1992 individuals and groups added features, hacked, slashed, and modified the PLP code, coordinated largely by Justin Mason (<jmason@iona.ie>) who started the **LPRng** mailing list.

In 1992 while at San Diego State University Prof. Powell redesigned and reimplemented the PLP code and named the result **LPRng**. The goals of the **LPRng** project were to build a server system that was as close to user abuse proof as possible, that would provide services limited only by the inherent capacities

of the support system, RFC1179 compliant, and with extensive debugging capabilities to allow quick and easy diagnostics of problems.

In 1999 the code base for **LPRng** was again reorganized in order to provide a common method for running on non-UNIX platforms such as Microsoft Windows NT, Apple Rhapsody, and embedded systems.

As a side effect of this work, many security problems that could develop were identified and steps taken to ensure that they were not present in **LPRng**. For example, **LPRng** clients such as lpr, lprm, lpc, and lpq can run as ordinary users programs, the lpd server can run as a non-root user once a network port has been opened, and all text formatting operations done by **LPRng** use a very restricted and highly secure version of the **snprintf** function.

1.2. Additional Resources

The main **LPRng** documentation is the LPRng Reference Manual, which is available in several formats. Information about **LPRng** and the latest release can be found on the **LPRng** web page http://www.lprng.com/LPRng.html

The **ifhp** documentation is the IFHP-HOWTO, which is available in the **ifhp** distribution. Information about **ifhp** and the latest release can be found on the **LPRng** web page http://www.lprng.com/LPRng.html

There is also a mailing list at lprng@lprng.com (mailto:lprng-request@lprng.com?subject=subscribe). To post to the list you must subscribe by sending send an email to lprng-request@lprng.com (mailto:lprng-request@lprng.com?subject=subscribe), with the message subject or body containing the word ‘subscribe’ or ‘help’.

Several presentations of **LPRng** and print spooling software have been made at the Large Installation System Administrator (LISA) conferences. The presentation at the LISA 98 conference is in the PowerPoint file `LISA98.ppt` in the **LPRng** distribution documentation.

1.3. Frequently Asked Questions

There are a list of Frequently Asked Questions that appear regularly on the **LPRng** mailing list. See The Most Frequently Asked Questions.

1.4. License, Copyright, and Disclaimer

The **LPRng** Print Spooler and the ifhp Print Filter software are distributed under the GNU Public License (GPL) and the Artistic License (license.txt). Users can choose to redistribute or use the software under a license that is appropriate for their purpose. Other licenses and distribution agreements are available by contacting AStArt Technologies (http://www.astart.com) for information.

THE MATERIAL IN THESE SOFTWARE PACKAGES AND DOCUMENTS IS PROVIDED WITHOUT FEE AND AS-IS WITH NO WARRANTY REGARDING FITNESS OF USE FOR ANY PURPOSE. THE AUTHOR AND ALL CONTRIBUTORS ARE NOT LIABLE FOR ANY DAMAGES,

DIRECT OR INDIRECT, RESULTING FROM THE USE OF THE SOFTWARE OR ANY INFORMATION PROVIDED IN THIS DOCUMENT.

1.5. Commercial Support

AStArt Technologies (http://www.astart.com) provides commercial support and enhancements for the **LPRng** and other network software. AStArt provides network and system consulting services for UNIX and NT systems, as well as real time and network software.

1.6. Web Site

Web Page: http://www.lprng.com

1.7. FTP Sites

Main FTP Site:

ftp://ftp.lprng.com/pub/LPRng (US)

Mirrors:

ftp://ftp.u-aizu.ac.jp/pub/net/lpr/LPRng (JA)
ftp://ftp.cs.columbia.edu/pub/archives/pkg/LPRng (US)
ftp://ftp.cise.ufl.edu/pub/mirrors/LPRng (US)
ftp://ftp.cs.umn.edu/pub/LPRng (US)
ftp://uiarchive.uiuc.edu/pub/ftp/ftp.lprng.com/pub/LPRng (US)
ftp://ftp.sage-au.org.au/pub/printing/spooler/lprng/ (AU)
ftp://mirror.aarnet.edu.au/pub/LPRng/ (AU/NZ)
http://mirror.aarnet.edu.au/pub/LPRng/ (AU/NZ)
ftp://sunsite.ualberta.ca/pub/Mirror/LPRng (CA)
ftp://ftp.informatik.uni-hamburg.de/pub/os/unix/utils/LPRng (DE)
ftp://ftp.uni-paderborn.de/pub/unix/printer/LPRng (DE)
ftp://ftp.mono.org/pub/LPRng (UK)
ftp://ftp.iona.com/pub/plp/LPRng (IE)
ftp://uabgate.uab.ericsson.se/pub/unix/LPRng (SE)

1.8. Mailing List

To join the **LPRng** mailing list, please send mail to lprng-request@lprng.com (mailto: lprng-request@lprng.com) with the word 'subscribe' in the BODY.

The **LPRng** mailing list is archived on http://www.findmail.com/list/lprng

1.9. PGP Public Key

The **LPRng** distributions have an MD5 checksum calculated, which is then signed with a PGP public key. Here is the key for validating the checksums:

```
Type Bits/KeyID    Date       User ID
pub  1024/00D95C9D 1997/01/31 Patrick A. Powell \
   <papowell@lprng.com>

-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: 2.6.3i

mQCNAzLygTQAAAEEANBW5fPYjN3wSAnP9xWOUc3CvsMUxjip0cN2sY5qrdoJyIhn
qbAspBopR+tGQfyp5T7C21yfWRRnfXmoJ3FVtgToAsJUYmzoSFY08eDx+rmSqCLe
rdJjX8aG8jVXpGipEo9U4QsUK+OKzx3/y/OaK4cizoWqKvy1l4lEzDsA2VydAAUT
tCdQYXRyaWNrIEEuIFBvd2VsbCA8cGFwb3dlbGxAYXN0YXJ0LmNvbT6JAJUDBRA0
XonoiUTMOwDZXJ0BAQ2cBAC7zU9Fn3sC3x0USJ+3vjhg/qA+Gjb5Fi1dJd4solc4
vJvtf0UL/1/rGipbR+A0XHpHzJUMP9ZfJzKZjaK/d0ZBNlS3i+JnypypeQiAqo9t
FV0OyUCwDfWybgAORuAa2V6UJnAhvj/7TpxMmCApolaIb4yFyKunHa8aBxN+17Ro
rrQlUGF0cmljayBBLiBQb3dlbGwgPHBhcG93ZWxsQHNkc3UuZWR1PokAlQMFEDLy
gTSJRMw7ANlcnQEBYBYD/0zTeoiDNnI+NjaIei6+6z6oakqO70qFVx0FG3aP3kRH
WlDhdtFaAuaMRh+RItHfFfcHhw5K7jiJdgKiTgGfj5Vt3OdHYkeeh/sddqgf9YnS
tpj0u5NfrotPTUw39n6YTgS5/aW0PQfO9dx7jVUcGeod1TGXTe9mIhDMwDJI4J14
=3Zbp
-----END PGP PUBLIC KEY BLOCK-----
```

1.10. References and Standards

The following references and standards have been used in the development of the **LPRng** software.

1.10.1. RFCs

During the early development of the Internet developers did not want to go through the laborious process of developing formal standards and applying to a standards body such as the EIA, IEEE, or ISO. Instead, they called the standards documents they developed [Requests for Comments]. These soon became *de facto* standards, and with the formal acceptance of the TCP/IP protocol as a network standard, *de jure* as well.

You can get copies of the RFCs from literally hundreds of network sites, including http://www.isi.edu, http://www.faqs.org/rfcs, NIS.NSF.NET (ftp://NIS.NSF.NET), RFC.JVNC.NET (ftp://RFC.JVNC.NET), or FTP.ISI.EDU (ftp://FTP.ISI.EDU).

The [RFC1179 - Line Printer Daemon Protocol] describes the protocol used to transfer jobs from client program to print server. See RFC1179 for more a discussion of this protocol and more details about the RFC.

1.10.2. PostScript

PostScript is one of the *de facto* standards for print jobs. The Adobe Corporation (http://www.adobe.com) provides an excellent set of references for the PostScript language. They have made many of these available for downloading from their web sites or have published them in book form.

The [PostScript Language Reference Manual] contains a great deal of technical information about the PostScript Language, and is the language reference manual.

The [PostScript Language Tutorial and Cookbook] is a very nice and easy to read introduction to PostScript programming, and has some very useful utilities. Combined with GhostScript and the gv display program you can very easily learn to write your own small PostScript programs, and more importantly, can learn to understand the contents of PostScript files.

The [PostScript Language Program Design] is the companion to the [PostScript Language Tutorial and Cookbook], and has more complex examples of PostScript programs. More importantly, it also introduces, although without explanation, the PostScript Document Structuring Conventions described in Appendix G of the The [PostScript Language Reference Manual]. This alone makes it useful.

1.10.3. HP PCL 5

The Hewlett-Packard (HP) PCL Printer Language is the second de-facto standard for print jobs. Currently, Hewlett-Packard makes documentation for PCL available through their [Developer Program]. You will need to register and then search their site for the [PCL 5 Printer Language Reference Manual].

1.10.4. HP PJL

The Hewlett-Packard (HP) Printer Job Language is used to control various features of HP printers. The [Printer Job Language Reference Manual] is also available from Hewlett-Packard (http://www.hp.com) through their [Developer Program].

1.10.5. PDF

The Portable Document Format (`pdf`) was developed by Adobe to be a more useful method of distributing documentation for view by online systems and software. The [Portable Document Format Reference Manual] is available from Adobe (http://www.adobe.com). While `pdf` is not used directly as a print job language, it is one of the more common formats for files that need to be printed. It can be converted to PostScript by most `pdf` viewers such as GhostScript and Adobe Acrobat.

Chapter 2. Installation

The basic components of the **LPRng** system are the executables and the database files. This section deals with generating and installing the executable files.

2.1. Getting Source Code and Support Programs

1. Obtain the latest or stable version of the **LPRng** source code from a **LPRng** FTP Site.
2. Obtain the latest or stable version of the **ifhp** filter source code from a **LPRng** FTP Site. This filter is used to support PostScript, PCL, and text printers.
3. Obtain the following GNU programs from one of the many GNU Software Mirror Sites (http://www.gnu.org) and install them. See the directions in the GNU Zip distribution for details.

   GNU **gzip** Compression Utility

   Used to generate the compressed **LPRng** distribution.

   GNU **tar** Archive Utility

   GNU (http://www.gnu.org) **tar** supports **gzip** compression and decompression and is used to generate the **LPRng** distribution.

   GNU **make**

   LPRng requires GNU (http://www.gnu.org) **make** for configuration and installation.

   GNU **gcc** Compiler or ANSI C Compiler

   LPRng requires and ANSI C compiler. If you do not have an ANSI C compiler then please use the GNU (http://www.gnu.org) **gcc** compiler.

4. Solaris Sparc and X86 Binaries for GCC and Make can be obtained from http://sunfreeware.com/.
5. While the following are not essential to **LPRng** they are used by the **ifhp** filter.

   file - File Identification Utility

   The Open Source **file** utility by Ian F. Darwin can be obtained from ftp://ftp.astron.com/pub/file/. or ftp://ftp.lprng.com/pub/LPRng/UNIXTOOLS/file/. This is a greatly improved version of the original UNIX file utility and may be used by the **ifhp** filter to do file recognition.

   gs - GhostScript

   GhostScript can be obtained from http://www.cs.wisc.edu/~ghost/ or http://www.ghostscript.com. GhostScript is a PostScript interpreter that allows you to translate

PostScript to various printer compatible formats such as PCL, as well as displaying the code on a terminal. You might also want to get the PDF extensions that allows GhostScript to read and print PDF files.

gv - GhostView

Of course you will want to get the **gv** program that uses GhostScript to display PostScript on an X terminal. It can be obtained from http://wwwthep.physik.uni-mainz.de/~plass/gv/

2.2. PATH Environment Variable and Utilities

Make sure that directory where you you have installed the GNU tools is one of the first entries in the shell search PATH environment variable. For example, if you have installed the utilities in the (default) directory `/usr/local/bin`, this should be one of the first entries in the PATH. For example:

```
PATH=/usr/local/bin:/bin:/sbin:/usr/bin
```

If you are compiling the distribution on a Solaris system you will need to use the utilities in the `/usr/ccs/bin` directory. This directory must be in the PATH *after* the directory where the GNU utilities have been installed. For example:

```
PATH=/usr/local/bin:/bin:/sbin:/usr/bin:/usr/ccs/bin
```

Check to see that the GNU **make** utility and not the default OS **make** is being used by default. Use `make -v` to determine what version you are using:

```
h4: {1} % make -v
GNU Make version 3.78, by Richard Stallman and Roland McGrath.
Copyright (C) 1988-1999
```

2.3. Network Mounted File System and Spool Directories

WarningIt is strongly recommended that the print spool directories and essential printer configuration file files should *NOT* be in an NFS mounted file system or on a file system which is NFS exported.

The **LPRng lpd** print server makes heavy use of file locking to coordinate and synchronize process activities. Given the historical and ongoing problems of file locking on and NFS mounted file system, not to mention the high overhead for doing file locking in a distributed environment, it is strongly recommended that spool directories *NOT* be on an NFS mounted partition or on a partition that is NFS exported.

This warning extends to other network file systems as well.

In addition, the configuration file such as `/etc/printcap` must be available or printing will not function. Care should be taken to ensure that these files are stable and available at all times.

2.4. Daemon User and Daemon Group

The **lpd** server is started at system initialization time and initially runs as ROOT (Effective UID 0). It performs all file and other operations with *non-privileged* user `daemon` Effective UID and group `daemon` Effective GID, and does a `setuid()` to these UID and GID values when running programs. The client programs such as **lpr** operate with the effective user IDs of the user which started them.

Most UNIX systems already have user `daemon` and group `daemon`, or a similar ones. If suitable user and group IDs are not present then the appropriate system administration tools should be used to create them. The configuration `--with-userid=UID` and `--with-groupid=GID` can be used to specify the user and group IDs. The user ID must *not* have login capability.

2.5. Configuration

The **LPRng** package consists of the following executables and configuration files:

- **lpd** - the **lpd** print server program
- **lpr**, **lpq**, **lprm**, **lpc**, and **lpstat** client programs for printing, status queries, job removal, server configuration, and System V **lpstat** emulation respectively.
- `printcap` print queue database file which is used by all the server and client programs
- `lpd.conf` **LPRng** configuration options which is used by all the server and client programs
- `lpd.perms` permission information which is used by the **lpd** server to control user actions.

LPRng uses the **configure** script generated by the GNU (http://www.gnu.org) **autoconf** utility to generate a set of `Makefiles`. These are used by GNU (http://www.gnu.org) **make** to compile and install the **LPRng** software. The following `Makefile` variables and values are set by **configure** to specify the location of the **LPRng** software:

Configure Variable	Default Value	Expanded Default Value	Override
`${prefix}`	`/usr/local`	`--prefix=PATH`	

Configure Variable	Default Value	Expanded Default Value	Override
`${exec_prefix}`	`${prefix}`	`/usr/local`	`--execprefix= PATH`
`${bindir}`	`${exec_prefix}/bin`	`/usr/local/bin`	`--bindir= PATH`
`${sbindir}`	`${exec_prefix}/sbin`	`/usr/local/sbin`	`--sbindir= PATH`
`${libexecdir}`	`${exec_prefix}/libexec`	`/usr/local/libexec`	`--libexecdir= PATH`
`${sysconfdir}`	`${prefix}/etc`	`/usr/local/etc`	`--sysconfdir= PATH`
`${mandir}`	`${prefix}/man`	`/usr/local/man`	`--mandir= PATH`

These are used to install the following files:

Configure Variable	Files
`${bindir}`	`lpr, lprm, lpq, lpstat`
`${sbindir}`	`lpc, checkpc, lpd`
`${libexecdir}/filters`	`lpf, pclbanner, psbanner, lpbanner`
`${sysconfdir}`	`lpd.conf, lpd.perms, printcap`
`${mandir}/man[1-9]`	`man pages`

You can set explicit values for the paths by using the override `--name=PATH`. For example:

```
./configure --prefix=/usr --sysconfdir=/etc \
    --mandir=/usr/share/man
```

Variable	Value	Files
${bindir}	/usr/bin	/usr/bin/{lpr,lprm,lpq,lpstat}
${sbindir}	/usr/sbin	/usr/sbin/{lpc,checkpc,lpd}
${libexecdir}/filters	/usr/libexec/filters	/usr/libexec/filters{lpf, pclbanner, psbanner, lpbanner}
${sysconfdir}	/etc	/etc/{lpd.conf,lpd.perms,printcap}
${mandir}/man[1-9]	/usr/share/man	/usr/share/man/man[1-9]/{man pages}

In addition to these standard **configure** options the following options provided.

`--disable-setuid`

Install the executables without setuid ROOT permissions. Non-setuid clients and programs are

inherently more secure than SETUID programs, and system administrators would be well advised to install them without SETUID root permissions. Please see Security Considerations for more details about this option.

`--enable-priv_ports`

Require connections to the **lpd** server to come from a privileged port (range 1-1023). By default **LPRng** will allow connections from any port. Please see Security Considerations for more details about this option.

`--disable-force_localhost`

The default **LPRng** configuration assumes that all printing will be done via a **lpd** print spooler running on the local host system. However, many larger sites prefer that all users do their printing via a few central servers, and do not run **lpd** servers on user systems. The `--disable-force_localhost` configuration sets the default value of the `force_localhost` value to `false`, by default allowing the **LPRng** clients to connect directly to **lpd** servers on remote hosts.

`--disable-require_configfiles`

By default, the **lpr**, **lpq**, **lprm**, and **lpc** clients require the `lpd.conf` and `printcap` files to be present on the localhost. The `--disable-require_configfiles` literal removes this requirement.

`--enable-kerberos`

Include support for Kerberos 5 authenticated transfers.

`--enable-mit_kerberos4`

Include support for MIT Kerberos 4 authenticated transfers.

`--disable-kerberos_checks`

Disable checks for kerberos support libraries, etc.

`--with-lpddir=DIR`

lpd executable directory (default ${sbindir}). For historical configuration compatibility.

`--with-filterdir=DIR`

Filter directory (default ${libexecdir}/filters). For historical configuration compatibility.

`--with-lpd_conf_path=PATH`

Path of `lpd.conf` file. For historical configuration compatibility.

`--with-lpd_perms_path=PATH`

Path of `lpd.perms` file. For historical configuration compatibility.

`--with-printcap_path=PATH`

Path of `printcap` file. For historical configuration compatibility.

`--with-ld_library_path=PATHLIST`

Set the LD_LIBRARY_PATH environment variable of filters to this value.

`--with-filter_path=PATHLIST`

Set the PATH environment variable of filters to this value.

`--with-userid=NAME`

Run **LPRng** as this user, default daemon

`--with-groupid=NAME`

Run **LPRng** as this group, default daemon

`--with-lockfile=PATH`

The lockfile path. This will be expanded to PATH.server or PATH.port allowing multiple **LPRng** servers to run on a single host.

`--with-filterdir=PATH`

Location of the filters installed by **LPRng**.

`--with-done_jobs=N`

retain status of last N done jobs.

`--with-done_jobs_max_age=N`

remove status of done jobs older than N seconds.

`--with-chooser_routine=NAME`

name of chooser routine provided by user

`--with-order_routine=NAME`

name of order routine provided by user

`--with-user_objs=NAME`

object file with routines provided by user

`--with-user_include=NAME`

include file with templates for routines provided by user

`--disable-strip`

Do not strip the executables before installing. For debugging and diagnostic purposes.

`--with-unixsocketpath=PATHNAME`

the pathname of the UNIX socket (off or blank to disable)

`--disable-ssl`

disable ssl support

`--with-openssl=DIR`

root location for OpenSSL

`--with-openssl-inc`

OpenSSL include files

`--with-openssl-lib`

OpenSSL library files

`--with-ssl_ca_file=FILE`

ssl Certificate Authority CERT file (default ${sysconfdir}/lpd/ssl.ca/ca.crt)

`--with-ssl_ca_key=KEY`

ssl Certificate Authority private key file (default ${sysconfdir}/lpd/ssl.ca/ca.key)

`--with-ssl_certs_dir=DIR`

ssl Certificate Authority certs working directory (default ${sysconfdir}/lpd/ssl.certs/)

`--with-ssl_server_cert=FILE`

ssl server certificate file (default ${sysconfdir}/lpd/ssl.server/server.crt)

`--with-ssl_server_password_file=FILE`

ssl server private key in password file (default ${sysconfdir}/lpd/ssl.server/server.pwd)

It is recommended that you use one of the following configurations:

1. If you already have a print spooling system installed and want to install **LPRng** for testing purposes or as an alternative to the existing system and keep your existing print spooling system, use:

```
./configure

use defaults for file locations and permissions:
  /usr/local/{bin,sbin,libexec/filters,man}
requires lpd to run on the local host
executables installed setuid ROOT
```

2. If you have manual pages in `/usr/share/man`, your existing print spooling system has executables in `/usr/bin` and `/usr/sbin`, and you want to replace your existing print spooling system, use:

```
./configure --prefix=/usr --sysconfdir=/etc \
  --mandir=/usr/share/man

executables and files in
   /usr/{bin,sbin,libexec/filters}
   /usr/share/man/man[0-9]
requires lpd to run on the local host
everything installed setuid ROOT
```

3. If you have manual pages in `/usr/share/man` and allow jobs (by default) to be sent directly to the server host (lightweight operation), use:

```
./configure --prefix=/usr --sysconfdir=/etc \
   --mandir=/usr/share/man --disable-force_localhost

executables and files in
   /usr/{bin,sbin,libexec/filters}
   /usr/share/man/man[0-9]
does not require lpd to run on the local host
everything installed setuid ROOT
```

2.6. System and User Printcap, lpd.conf, and lpd.perms files

The system `printcap` file contains the definitions of the print queues used by **LPRng**, and is located in the directory specified by the configuration `${sysconfdir}` value. For a complete description of the `printcap` file see Printcap Database. If your system does not have an existing `printcap` file then a *dummy* file similar to the following is installed by default:

```
# dummy printcap file
lp:cm=Dummy Printcap Entry:
 :lp=/dev/null
 :sd=/var/spool/lpd/%P
```

In addition to the system `printcap` file, each user can have a `${HOME}/.printcap` which contains printcap entries as well. The the user `printcap` file is in effect appended to the system printcap file, and values in the user printcap file override the system printcap file. However, in order to allow users to specify a default printer after reading the printcap file information the entries are sorted so that printcap entries defined by users come first.

The `lpd.conf` is located in the `${sysconfdir}` directory and provides configuration settings for both the **LPRng** client and server programs. For a complete description of the `lpd.conf` file see Configuration File, Defaults and Overrides. During installation the `${sysconfdir}/lpd.conf.template` is created with the default **LPRng** information and if there is not an existing `${sysconfdir}/lpd.conf` file is copied to it.

The `lpd.perms` is located in the `${sysconfdir}` directory and is only by **lpd** to determine user permissions for printing activities. For a complete description of the `lpd.perms` file see Permissions and Authentication for details. During installation the `lpd.perms.template` file is installed in the `${sysconfdir}/lpd.perms.template` and if there is not an existing `lpd.perms` file is copied to it.

By default, the **LPRng** client programs **lpr**, **lpq**, **lprm**, and **lprc** will require a `lpd.conf` and `printcap` file to be installed on the local host. You can relax this requirement by setting using the

`--disable-require_configfile` configuration option. You can also create empty files to satisfy the program requirements.

2.7. Checking System Installation with checkpc

The **checkpc** program is used to make sure that the spool directories and files used by **LPRng** have the correct permissions and are in place. By default, **checkpc** will check permissions and report if there are any problems. You should run this as `root`. For example:

```
h4: {2} # checkpc
Warning - No configuration file '/etc/lpd.conf'
Warning - No lpd only printcap file found in '/etc/lpd_printcap'
Warning -  ** cannot open '/var/run/lpd.printer' - 'Permission denied'
Warning -  bad directory - /var/spool/lpd/lp
Warning -   Printer 'lp' spool dir '/var/spool/lpd/lp' needs fixing
```

In the above example, **checkpc** has discovered that the `lpd.conf` file is missing. This is a serious problem and indicates that the software may be incorrectly installed.

The `lpd only printcap` message is usually of concern to administrators who wish to use some of **LPRng**'s more exotic configuration options. It is possible to have separate `printcap` files for client and server programs. This is useful when `printcap` files get extremely large and cuts down substantially on system management problems.

The `permission denied` message for `/var/run/lpd.printer` is serious, as the **lpd** server uses this as a lock file.

The `bad directory` message about the spool directory is usually caused by wrong permissions or a missing directory.

The `checkpc -f` (fix) option causes **checkpc** to take action to rectify errors. The `checkpc -f -V` (verbose) option causes the fixup activity to be displayed a well:

```
h4: {3} # checkpc -f -V
Checking for configuration files '/etc/lpd.conf'
  found '/usr/local/etc/lpd.conf', mod 0100644
Checking for printcap files '/etc/printcap'
  found '/usr/local/etc/printcap', mod 0100644
Checking for lpd only printcap files '/etc/lpd_printcap'
 DaemonUID 1, DaemonGID 1
Using Config file '/etc/lpd.conf'
LPD lockfile '/var/run/lpd.printer'
 Checking directory: '/var/run'
   directory '/var'
   directory '/var/run'
  checking '/var/run/lpd.printer' file

Checking printer 'lp'
 Checking directory: '/var/spool/lp'
```

```
 directory '/var'
 directory '/var/spool'
 directory '/var/spool/lp'
file 'control.lp', zero length file unchanged in 1 hours
file 'status.lp', zero length file unchanged in 1 hours
file 'status', zero length file unchanged in 1 hours
file 'log', zero length file unchanged in 1 hours
file 'acct', zero length file unchanged in 1 hours
checking 'control.lp' file
checking 'status.lp' file
checking 'status' file
cleaning 'status' file, 0K bytes: no truncation
checking 'log' file
cleaning 'log' file, 0K bytes: no truncation
checking 'acct' file
cleaning 'acct' file, 0K bytes: no truncation
```

checkpc will create the spool directories and any missing files, and fix the permissions of existing files.

2.8. Compilation and Install

Once you have decided on the configuration you want and understand what files need to be installed, then you are ready to do the install. It is extremely simple to extract the files, configure, compile and install the software:

```
h4: {4} % gunzip -c LPRng-<version>.tgz | tar xvf -
h4: {5} % cd LPRng-<version>
h4: {6} % ./configure  [ ... configuration options ]
h4: {7} % make clean all
h4: {8} % su   # you must do the following commands as root
h4: {9} # make install
h4: {10} # # if checkpc did not run, do the next command
h4: {11} # # make sure checkpc and lpq are in your paths
h4: {12} # # if using CSH, use 'rehash' as well
h4: {13} # checkpc -f
h4: {14} # # check to see if the server is running
h4: {15} # lpq
```

During installation you may get an error message indicating that the **lpd** file could not be installed or the **lpd** server could not be started. If this is the case, then carry out the appropriate steps in Updating Print Spooler Software and Startup Scripts to remove the existing print spooling software and then try to reinstall **LPRng**. You should also to check the System Specific Notes section for any system specific installation requirements.

While the **LPRng** installation scripts try hard to set up the **lpd** server in place, you will still need to reboot your host and make sure that the various printing facilities are working correctly after reboot.

2.9. Installation Problems

Read the notes for your OS in section System-dependent notes for specific installation help (if any).

The following is a list of commonly encountered problems and their solution. If these do not solve your problem, then send mail to the lprng@lprng.com mailing list. You will have to subscribe to the list in order to post to the list.

1. **Make** complains about a malformed `make` or `Makefile` file, illegal syntax in the file, or illegal entries in the file. You are most likely not running GNU Make. You *must* use GNU **make** or you should be a Unix Wizard able to master the mysteries of converting GNU Makefiles to your local system **make**. It is easier to simply install GNU **make**.
2. The C Compiler complains about missing files or has a large number of errors. Use **gcc** instead of your vendor's C compiler.

   ```
   configure --with-cc=gcc
   ```

   If there are messages about missing system files, then you most likely have an incomplete set of system `include` files, or the `include` do not properly reference other required include files, or the include files are located in an *unusual* location. If you are using **gcc** then make sure that the **gcc** was carried out correctly on your system. The easiest way to assure this is to recompile and reinstall the **gcc** compiler.
3. If you have checked your compiler installation and are still missing libraries or files then the `include` files may be in `/usr/local/include` and libraries may be in `/usr/local/include` and these directories may not searched or used by the compiler by default. This can be fixed by using the `--with-cppopts=` and `--with-ldopts=` configure options.

   ```
   configure \
     --with-cppopts="-I/usr/local/include -I/usr/include/kerberosIV" \
     --with-ldopts="-L/usr/local/lib -L/usr/lib/kerberosIV"
   ```

4. The software compiles but will not run on the system. Make sure that you have followed your system specific rules for compiling and installing `setuid ROOT` programs on your system. You may need to statically link your executables.
5. The software was compiled on one system and copied to another system, but will not run on the other system. Try compiling the software on the target system. If it compiles and runs, then you most likely have an issue with libraries or Operating System Versions.

After you have installed the LPRng software and rebooted your system, do the following commands:

```
h4: {16} # lpq
Printer: lp@astart
 Queue: no printable jobs in queue
```

If you do not get status displayed, or you get some other error message, then the following are a series of tests can use to check that **LPRng** is installed correctly.

First we will run **lpd** in the *foreground* and are used to make sure that our system configuration is correct. You will need `root` permissions to do the following steps. Stop the running currently running **lpd** process. Next, run **lpd** in foreground mode:

```
h4: {17} # ps -aux | grep lpd
daemon   240  0.0  0.0  1292  0  ??  IWs  -     0:00.00 lpd: lpd Waiting
h4: {18} # kill 240
h4: {19} # checkpc -f
h4: {20} # /usr/local/bin/lpd -F -D1
Fatal error - Another print spooler is using TCP printer port
```

If you get the above error message, then you have either not terminated the running **lpd** server, there is another process using TCP/IP port 515, or you are not starting the **lpd** server as ROOT. See the System Specific Notes for details on how to resolve these issues.

Correct the problem and then restart the server. You should see the output indicated below:

```
h4: {21} # /usr/local/bin/lpd -F -D1
1999-04-05-14:35:14.023 astart27 [2667] Waiting  lpd: LOOP START
1999-04-05-14:35:14.024 astart27 [2667] Waiting  Get_max_servers: \
   getrlimit returns 256
1999-04-05-14:35:14.024 astart27 [2667] Waiting  Get_max_servers: \
   returning 128
1999-04-05-14:35:14.025 astart27 [2667] Waiting  lpd: \
   max_servers 128, active 0
1999-04-05-14:35:14.025 astart27 [2667] Waiting  lpd: \
   starting select timeout 'yes', 600 sec
```

Now from another window do the following commands:

```
h4: {22} # lpq
Printer: lp@astart
 Queue: no printable jobs in queue
```

At this point your **LPRng** software has been installed and tested. See the Updating Print Spooler Software and Startup Scripts for details on how to automatically start **lpd** at boot time.

2.10. Updating Print Spooler Software and Startup Scripts

If you are replacing your existing print spooling spooling system, you must shut down and remove the existing print spooler software before installing the **LPRng** software. This process is fairly system dependent, and requires a small amount of system expertise.

To assist in this process the **LPRng** installation has a set of `preinstall`, `postinstall`, `preremove`, and `postremove` scripts in the distribution that may be suitable for your local system use. If these fail to work during the system installation, you will need to carry out the steps described here by hand.

2.10.1. SunOS, Linux, and BSD Derived Systems

The `SunOS`, `Linux`, and `BSD` derived systems such as `BSDi`, `FreeBSD`, `OpenBSD`, and others use a version of the *legacy* or *vintage* **lpd** print server and the **lpr**, **lprm**, **lpq**, and **lpc** client programs. By convention, most of the printing related programs are in the `/usr/bin`, `/usr/sbin`, `/usr/libexec`, and `/usr/ucb` directories.

The **lpd** print spooler is started by either the `rc` startup script or by a *startup script* file in the `/etc/rc.d/init.d` or `/etc/init.d` directory. You can first locate the startup commands as follows.

1. Use the find(1) utility to search the `/etc` directory for the file that contains the startup command.

```
h4: {23} # cd /etc
h4: {24} # find . -type f -exec grep -l lpd {} \; -print
./rc.local
```

2. Examine each of the files found find the one that starts the **lpd** print spooler. You can simply comment out the command or change it to start the **LPRng lpd** print server.

```
h4: {25} # more /etc/rc.local
if [ -f /etc/printcap  -a -f /usr/libexec/lpd ] ; then
  /usr/libexec/lpd ;
fi

--- change this to
if [ -f /etc/printcap  -a -f /usr/sbin/lpd ] ; then
  /usr/sbin/lpd ;
fi
```

3. If you have an existing `printcap` file, then you should either copy this to the location used by **LPRng** or make a symbolic link to it.

Next we kill the currently running **lpd** process.

```
h4: {26} # ps -auxw |grep lpd
papowell 23932  0.0  0.3  224  184  p3  S+  10:40AM  0:00.01 grep lpd
daemon  17763  0.0  0.2  448  120  ??  IWs  29Mar99  0:01.35 (lpd)
h4: {27} % kill 135
h4: {28} % kill 135
135: No such process
```

Next, you should remove or rename the existing print system executables. The following example shows how to use the **find** utility to track down candidates.

```
h4: {29} # find /usr -type f -name lp\*  -print >/tmp/candidates
h4: {30} # find /sbin -type f -name lp\*  -print >>/tmp/candidates
h4: {31} # cat /tmp/candidates
/usr/bin/lpunlock
/usr/bin/lpqall.faces
/usr/bin/lpq             <---- old
/usr/bin/lpr             <---- old
/usr/bin/lprm            <---- old
/usr/bin/lptest
/usr/doc/samba-1.9.18p10/examples/printer-accounting/lp-acct
/usr/man/man1/lpq.1
/usr/man/man1/lpr.1
/usr/man/man1/lprm.1
/usr/man/man1/lptest.1
/usr/man/man4/lp.4
/usr/man/man8/lpc.8
/usr/man/man8/lpd.8
/usr/sbin/lpc            <--- old
/usr/sbin/lpd            <--- old
/usr/sbin/lpf            <--- old
h4: {32} # mv /usr/bin/lpq  /usr/bin/lpq.old
h4: {33} # mv /usr/bin/lpr  /usr/bin/lpr.old
h4: {34} # mv /usr/bin/lprm /usr/bin/lprm.old
h4: {35} # mv /usr/sbin/lpc /usr/sbin/lpc.old
h4: {36} # mv /usr/sbin/lpd /usr/sbin/lpd.old
h4: {37} # mv /usr/sbin/lpf /usr/sbin/lpf.old
```

After all this, you should now run `checkpc -f` (as root) to make sure that the **LPRng** configuration is present and correctly set up, and then start **lpd** by hand. You should try to use **lpq** to see if the spool queues are present and set up correctly and the system is functional.

```
h4: {38} # checkpc -f
h4: {39} # lpd
h4: {40} # lpq
Printer: lw4@h2  'Hp : LaserWriter'
 Queue: no printable jobs in queue
 Status: job 'root@h2+884' removed at 11:27:25.864
 Filter_status: done at 11:27:25.766
h4: {41} # lpr /etc/motd
h4: {42} # lpq
Printer: lw4@h2  'Hp : LaserWriter'
 Queue: no printable jobs in queue
 Status: job 'root@h2+888' removed at 11:27:25.864
 Filter_status: done at 11:33:17.020
```

Finally, you should reboot your machine and make sure that the **lpd** print server starts correctly.

2.10.2. Solaris, HP-UX, and other SysVR4 Derived Systems

The original SysVR4 (System V, Release 4) software did not have any support for RFC1179 network printing (Berkeley **lpd**). Support for this was added in a wide variety of different ways. There are a wide range of different ways that this was done, but most are based on the following system or process structure.

The `lpsched` process (`/usr/lib/lp/lpsched/`) process performs many of the functions of the **LPRng** and BSD **lpd** server. This process is responsible for overseeing job printing and starting processes for handling the print queues on the local host. This process must be shut down and the running print spooling servers terminated before **LPRng** can be correctly installed. While there is no simple and reliable method of shutting down a running `lpsched` process and the associated network services, it is simple to *prevent* the process from being started.

The `preinstall.solaris` script is a file in the **LPRng** distribution that contains most of the commands needed to remove the Solaris System V printing software. These are explained in detail in the sections below. The procedures outlined below can be used on other SystemVR4 systems.

```
#!/bin/sh
# This is an effort to automate the setup
#  needed to install the LPRng software on the
#  Solaris OS.  This is effectively a one way path.
#  You are warned.
PATH=/etc:/usr/etc:/usr/bin:/bin:/sbin:/usr/sbin:$PATH
# remove the init.d entry and links
for i in /etc/rc*.d/*lp ; do
    b=`basename $i`;
    d=`dirname $i`;
    mv $i $d/UNUSED.$b.UNUSED
done
# rename files
renameit () {
    for i in $* ; do
        if [ -f $i -a '!' -f $i.old ] ; then
            echo "renaming $i $i.old";
            mv $i $i.old
        fi
    done
}
renameit /usr/bin/lp /usr/bin/lpstat /usr/sbin/lpadmin \
  /usr/sbin/lpfilter /usr/sbin/lpforms /usr/sbin/lpmove \
  /usr/sbin/lpshut /usr/sbin/lpsystem /usr/sbin/lpusers \
  /usr/ucb/lpc /usr/ucb/lpq /usr/ucb/lpr /usr/ucb/lprm \
  /usr/ucb/lptest /usr/lib/lp/lpsched /usr/lib/lp/lpNet
# remove the cron entry
if [ -f /var/spool/cron/crontabs/lp ] ; then
    mv /var/spool/cron/crontabs/lp \
       /var/spool/cron/UNUSED.crontabs.lp
fi
# comment out inetd.conf entry
if egrep '^printer' /etc/inetd.conf >/dev/null 2>& ; then
    mv /etc/inetd.conf /etc/inetd.conf.bak
    sed -e 's/^printer/# printer/' </etc/inetd.conf.bak \
```

```
       >/etc/inetd.conf
fi
# remove the nlsadmin entry
nlsadmin -r lpd tcp
nlsadmin -r lp tcp
echo REBOOT SYSTEM and then install LPRng
```

First, you will need to remove the `/etc/rc` startup files in the `/etc/rc*.d` directories that start the `lpsched` process; see the `init` program man page for details. You can find these files by using:

```
h4: {43} # cd /
h4: {44} # find . -type f -exec grep -l lpsched {} \; -print >/tmp/files
h4: {45} # cat /tmp/files
/etc/rc0.d/K20lp
/etc/rc2.d/K20lp
/etc/rc2.d/S80lp
/etc/init.d/lp
h4: {46} # ls -l ` cat /tmp/files `
lrwxrwxr-x 1 root bin 1 Dec 29 23:39 /etc/rc0.d/K20lp -> ../../init.d/lp
lrwxrwxr-x 1 root bin 1 Dec 29 23:39 /etc/rc2.d/K20lp -> ../../init.d/lp
lrwxrwxr-x 1 root bin 1 Dec 29 23:39 /etc/rc2.d/S80lp -> ../../init.d/lp
-rwxr--r-- 5 root sys 460 Sep 1 1998 /etc/rcS.d/K39lp
```

You can remove these files, or simply comment out all of the executable commands in the `/etc/init.d/lp` file. Next, find all of the printing related commands and rename them. For example:

```
h4: {47} # find /usr -type f -name lp\* -print >/etc/printingfiles
h4: {48} # cat /tmp/printingfiles
/usr/bin/lp
/usr/bin/lpstat
/usr/lib/lp/bin/lp.cat
/usr/lib/lp/bin/lp.set
/usr/lib/lp/bin/lp.tell
/usr/lib/lp/lpNet
/usr/lib/lp/lpsched
/usr/lib/lp/lpdata
/usr/sbin/lpadmin
/usr/sbin/lpfilter
/usr/sbin/lpforms
/usr/sbin/lpmove
/usr/sbin/lpshut
/usr/sbin/lpsystem
/usr/sbin/lpusers
/usr/ucb/lpc
/usr/ucb/lpq
/usr/ucb/lpr
/usr/ucb/lprm
/usr/ucb/lptest
h4: {49} # vi /tmp/printingfiles  # remove ones you want to save
```

```
h4: {50} # for i in ` cat /tmp/printingfiles ` ; do
>   if [ -f $i -a '!' -f $i.old ] ; then  mv $i $i.old ; fi;
> done
```

On some systems there may be a `cron` file `/var/spool/cron/crontabs/lp` which is used to to periodically update and roll over error logs. You may want to remove this file or comment out its contents.

Check the `/etc/inetd.conf` file for a line like the one below and comment it out. This line is not present on all systems.

```
printer stream tcp nowait root /usr/lib/print/in.lpd in.lpd
```

Use `nlsadmin` to force the `TCP/IP listener` to release the port, as illustrated below. This may not be present on all system.

```
h4: {51} # nlsadmin -v tcp
lpd  \x00020203000000000000000000000000  ENABLED  \
  NORPC  root  NOMODULES  /var/spool/lp/fifos/listenBSD  #
0  \x00020ACE00000000000000000000000000  ENABLED    \
  NORPC  root  NOMODULES  /usr/lib/saf/nlps_server  #
lp  NOADDR  ENABLED  NORPC  root  NOMODULES \
  /var/spool/lp/fifos/listenS5  #
h4: {52} # nlsadmin -r lpd tcp
h4: {53} # nlsadmin -r lp tcp
```

Run `pmadm -l` as shown below.

```
h2.private: {2} # pmadm -l
PMTAG    PMTYPE   SVCTAG   FLGS ID    <PMSPECIFIC>
zsmon    ttymon   ttya     u    root  /dev/term/a I - /usr/bin/login ...
zsmon    ttymon   ttyb     u    root  /dev/term/b I - /usr/bin/login ...
```

If you see `zsmon` entries for SystemV `lpsched` support, then use `pmadm -r` to remove them. These may not be present on all system. See the `pmadm` man page for details on the `-r` literal.

You must now `reboot` the host.

```
h4: {54} # shutdown -y "Whooga! Whooga! Dive! Dive! System going down."
```

When the system reboots, make sure that there is no process listening on port 515 (printer port) by using:

```
h4: {55} # telnet localhost 515
```

If you can connect, then there is a problem beyond the scope of these instructions.

Compile and/or install the **LPRng** software. Make sure that the **LPRng** startup files have been installed correctly in `/etc/init.d/lprng` and that the symbolic links to the file have been made correctly. The **LPRng** startup file will usually have the following contents and you should use the same filename formats that the **lp** startup files had for the links to the `/etc/init.d/lprng` startup file:

```
LPD_PATH=/usr/sbin/lpd
SHOWALL=-e
case "$1" in
  start)
        # Start daemons.
        /bin/echo "Starting lpd: \c"
        ${LPD_PATH}
        /bin/echo ""
        ;;
  stop)
        # Stop daemons.
        /bin/echo "Shutting down lpd: \c"
        kill -INT `ps ${SHOWALL} \
           | awk '/lpd/{ print $1;}'` >/dev/null 2>&1
        /bin/echo " server stopped";
        ;;
  *)
        echo "Usage: $0 {start|stop}"
        exit 1
        ;;
esac
```

Start the **lpd** server and then test it:

```
h4: {56} # checkpc -f
h4: {57} # /usr/sbin/lpd (or /usr/local/sbin/lpd)
h4: {58} # lpq
Printer: lp
 Queue: no printable jobs in queue
```

2.11. Emulation for UNIX SystemV lp and lpstat

Many utilities in the UNIX System V environment require the **lp**, **lpstat**, and **cancel** programs. It is almost impossible to modify these utilities, as many are *vintage* software which is unsupported or which would be too costly to update. In order to support these applications **LPRng** emulates the **lp**, **lpstat**, and **clean** programs. See the **LPRng** man pages for **lp**, **lpstat**, and **cancel** in the **LPRng** distribution for details and compatibility.

The **LPRng lpstat** emulator accepts the **lpstat** command line options returns status in a format that is close to the one that common **lpstat** implementations return. Unfortunately, due to the wide variety of

different modifications and vendor versions of **lpstat** there are slight differences between the this status and the status returned by the original **lpstat**. If this is the case, then there is little to do but to modify the source code for **lpstat** and compile a version that implements the required format.

If the **lpr** program is invoked with the name **lp**, it will simulate the **lp** options. This can be done by making a symbolic link to the **lpr** program or by making a copy of the **lpr** program with the name **lp**.

```
h4: {59} # cd /usr/bin
h4: {60} # ln -s lpr lp
h4: {61} # lp /tmp/hi
request id is root@h4+489
```

Finally, if the **lprm** program is invoked with the name **cancel** it will simulate the **cancel** command. This can be done by making a symbolic link to the **lprm** program or by making a copy of the **lprm** program with the name **cancel**.

```
h4: {62} # cd /usr/bin
h4: {63} # ln -s lprm cancel
h4: {64} # cancel 489
cancel 513
Printer lp@h9:
  checking perms 'root@h9+513'
  dequeued 'root@h9+513'
```

Many *vintage* or *legacy* applications have fully qualified paths to the **lp** and **lpstat** executables, and it may be necessary to make additional symbolic links or copies of the **LPRng** executables to satisfy their pathname requirements.

```
h4: {65} # ln -s /usr/bin/lpr /usr/ucb/lpr
```

2.12. SAMBA and LPRng

The SMB network protocol is used by many Microsoft Operating Systems to implement file and printer sharing. SAMBA is a UNIX package that implements the SMB protocol and provides a simple and easy way to import and export file systems and printer facilities. The web site for SAMBA is http://www.samba.org. The SAMBA code is extremely easy to install and the SWAT (Samba Web Administration Tool) makes configuration almost trivial.

See the SAMBA `doc/text/Printing.txt` and related documentation for details on printing. In the `samba.conf` file `[global]` section or in the SWAT page for printing configuration you need to specify the that you want to have Samba handle printing, the `print`, **lpq**, and **lprm** commands to be used when a user prints a job, asks for status, or removes a job, and a temporary directory to hold print jobs when they are submitted. The following is a simple example of to set up printing for authenticated users.

```
[printers]
    path = /var/spool/lpd/samba
    #  ---  do not use the Samba default path = /tmp
    print ok = yes
    printing = lprng
    load printers = yes
    guest ok = no
    printcap name = /etc/printcap
    print command =      /usr/bin/lpr  -P%p -r %s
    lpq command   =      /usr/bin/lpq  -P%p
    lprm command  =      /usr/bin/lprm -P%p %j
    lppause command =    /usr/sbin/lpc hold %p %j
    lpresume command =   /usr/sbin/lpc release %p %j
    queuepause command = /usr/sbin/lpc  stop %p
    queueresume command = /usr/sbin/lpc start %p
```

1. Samba will make a copy of the files to be printed in the directory specified by `path`. If the print operation fails then sometimes the print file is left in the directory.
2. The directory should have the same ownership and permissions as `/tmp`, i.e.- owner and group `root` and `bin`, with `01777` permissions, where `01000` is the sticky bit.

   A directory whose 'sticky bit' is set becomes an append-only directory, or, more accurately, a directory in which the deletion of files is re- stricted. A file in a sticky directory may only be removed or renamed by a user if the user has write permission for the directory and the user is the owner of the file, the owner of the directory, or the super-user. This feature is usefully applied to directories such as /tmp which must be publicly writable but should deny users the license to arbitrarily delete or rename each others' files.
3. The directory should be examined periodically and files older then a day should be removed. The following command can be used to do this, and should be put in a file that is periodically (one a day) executed by the **cron** facility:

   ```
   find /var/spool/lpd/samba -type f -mtime 2d -exec rm -f {} \;
   ```
4. You must specify the print method as `printing = lprng`. This will allow Samba to parse the **LPRng lpq** status format correctly.
5. You must put all of the printers which Samba has access to in the `printcap` file. Your Samba server may support reading the printcap file by using a program. In this case the printcap file entry can be one of the following:

   ```
   [printers]
     #
       printcap name = |/usr/local/libexec/filters/getpc
     # or
       printcap name = |/usr/bin/lpc client all

   #!/bin/sh
   # getpc program
   /usr/bin/lpq -as | /bin/sed -e 's/[@:].*//p'
   ```

The `lpc client all` command will generate the printcap entries for all of the printers. This was done to support Samba and other printer gateway systems. You can also use a simple script to modify the output of the printer status command as shown in the example.

6. Samba can be configured to allow guests or non-authenticated users to spool print jobs. Unfortunately, by default **lpr** will mark the jobs as submitted by the Samba server, not the remote users. To solve this problem, the `lpr -U%U@%M` option causes **lpr** to mark the jobs as submitted by user `%U` on host `%M`, instead of the Samba server process. The use of this option is restricted to root and a set of userids listed in the allow_user_setting configuration option. If the userid of the submitter is not in this list, then the option is quietly ignored. The `-U%U@M` can also be used with the other LPRng commands as well. For example:

```
[printers]
    guest ok = yes
    print command =       /usr/bin/lpr  -U%U@%M -P%p -r %s
    lpq command   =       /usr/bin/lpq  -U%U@%M -P%p
    lprm command  =       /usr/bin/lprm -U%U@%M -P%p %j
    lppause command =     /usr/sbin/lpc -U%U@%M hold %p %j
    lpresume command =    /usr/sbin/lpc -U%U@%M release %p %j
    queuepause command =  /usr/sbin/lpc -U%U@%M stop %p
    queueresume command = /usr/sbin/lpc -U%U@%M start %p
```

When Samba gets a request for print queue status, it runs the `lpq command` program and then parses the output of this command. Unfortunately, different versions of Samba have different ways of parsing the output - some are more flexible than others.

>

One of the problems that might occur is when the **LPRng** `done_jobs` feature is enabled. This causes that status of the last few jobs to be retained so that users can see what happened to their jobs. For example:

```
h110: {588} % lpq
Printer: t1@h110 'Test Printer 1'
 Queue: no printable jobs in queue
 Server: no server active
 Status: job 'papowell@h110+336' saved at 14:42:54.607
 Filter_status: FILTER DONE
 Rank   Owner/ID         Class Job Files    Size Time
done   papowell@h110+336   A   336 /tmp/hi     3 14:42:53
```

In this example, the `done` job will have its status displayed by the lpq command. However, this may confuse Samba, and it may report odd or unusual status for your jobs. If the **lpq** command reports that your job has completed but Samba reports that it is printing or is stopped, then you should disable the `done_jobs` option in the printcap entry:

```
lp:
  :done_jobs=0
  :...
```

2.13. Security Concerns

While the **LPRng** software has been written with security as the primary goal there is always the problem with undetected errors in the **LPRng** software that when exploited could compromise system security. The most serious concern is that of gaining ROOT (UID 0) permissions.

The simplest way to handle this problem is to not install LPRng with `setuid ROOT` permissions. Client programs will be able to connect to the **lpd** server. Since the **lpd** server is started by the system startup script with effective UID root, it is the only program in this suite that will have an privileged user id.

A more radical step is to run the **lpd** server as a non-privileged user entirely. However, the RFC1179 protocol specifies that the **lpd** TCP/IP port is 515 and **lpd** requires root permissions to open and bind to port 515. The **lpd** server can use the `setuid()` system call after binding to this port do drop ROOT capabilities. However, in order to fully compatible with RFC1179, **lpd** must originate connections from a *reserved* port in the range 721-731, although in practice port 1-1023 seems to be acceptable.

If inter-operability with non-**LPRng** print spoolers is not desired, then it is *trivial* to configure **LPRng** to use a non-privileged port by using the `lpd.conf` file. For example, in the `/etc/lpd.conf` file, you only need to change the indicated lines:

```
# Purpose: lpd port
#   default lpd_port=printer
lpd_port=2000
# or lpd_port=localhost%2000
```

The `lpd_port` specifies the (optional) IP address and port to which the **lpd** server binds and to which the clients will connect. **LPRng** applications will connect to port 2000 to transfer jobs and ask for status. You can also use this facility to establish a *private* set of print spoolers which can be used for testing See Testing and Diagnostic Facilities for more details.

Some *legacy* print filters are not *meta-char-escape* proof. For example, suppose that a user decided to spool a job as follows:

```
h4: {66} # lpr "-J`;rm -rf /;`" /tmp/a
```

This would create a job file with the line:

```
J`rm -rf /;`
```

and gets passed to a print filter as

```
/usr/local/printfilter  -J`rm -rf /;`
```

The observant reader will observe that the above line may have the most hideous consequences if it is processed by a shell. For this reason the **LPRng** software takes extreme precautions and *sanitizes* control file contents and file names so that they do not contain any control or metacharacters.

Finally, you can use a Unix socket (i.e. - FIFO) for connections to the server on the localhost, and disable the **lpd** listening socket by setting the `lpd_listen_port` value to `off`.

Chapter 3. System Specific Notes

The following are a set of suggestions and recommendations for specific systems.

3.1. Solaris

The Sun Microsystems Solaris printing system is derived from the System V UNIX system. Please see the Solaris, HP, and SysVR4 Derived Systems installation information for a detailed description of how to install **LPRng** and remove the Solaris Print Services.

If you want to simply forward jobs from a Solaris system to a BSD print spooling system you can use the following commands to create a printer. Check your specific system references and man pages for options:

```
h4: {67} # lpsystem -t bsd servername    # add server
h4: {68} # lpadmin -p pr -s servername -T unknown -I any # set up printer
h4: {69} # accept pr    # enable queueing
h4: {70} # enable pr    # enable printing
h4: {71} # lpstat -t    # show status
scheduler is running
system for pr: servername
lp accepting requests since Mon Aug  6 12:00:00 PST 2000
Printer: pr@servername  'Hp : Laserwriter' (printing disabled)
 Queue: 1 printable job
 Rank  Owner/ID            Class Job Files            Size Time
1      papowell@h4+207       A   207 h4.023205           3 18:24:54
```

The above commands will create the necessary directories and files for the printer. If you want to use the `lp -o option` syntax to pass options to the LPRng print spooler you will have to enable this by hand. The `/etc/printers.conf` (this may be in some other directory besides `/etc`) needs to be modified so that it allows options to be passed using the Solaris convention, which is to put them on the `S` or `O` line of the control file. For example:

```
#
# The preferred method of modifying this file is through the use of
# lpset(1M) or fncreate_printer(1M)
#
pr:bsdaddr=servername,pr,Solaris:
_default:use=pr:
```

The `bsdaddr=host,printer[,Solaris]` indicates that the entry is for a remote RFC1179 printer on server `host` with name `printer`. The `Solaris` option indicates that the Solars extensions to the RFC1179 protocol are to be used.

3.2. Linux

There is no universal way to install **LPRng** cleanly on all of the different Linux systems. The major difficulty is the fragmentation in the various libraries, location of files, and system dependencies. If the **LPRng** installation procedure does not install the software correctly, please use the following remedial steps.

1. Check the **LPRng** web site http://www.lprng.com and see if there is a Linux Release RPM or other binary distribution for your version of Linux. If not, then you will have to do a source install.
2. Obtain the source distribution and follow the instructions outlined in other sections to compile and install the software. Use the following configuration options which correspond to Linux:

```
h4: {72} # ./configure --prefix=/usr \
  --sysconfdir=/etc --mandir=/usr/share/man
h4: {73} # make clean all install
h4: {74} # checkpc -f
```

3. You may need to modify your existing printcap file by adding the `:bkf` flag as shown below, or replace the entries with other filters.

```
lp:\
  :if=/usr/local/libexec/lpr/hplaserjet3:\
  :bkf:\     # added to the printcap by hand
  :...
```

4. Test the system by printing a file. If this does not work, then you will have to determine if the problem is in the print spooling software or in the filter. See the section on **ifhp** for directions on how to replace the vendor supplied filters with **ifhp**.

3.3. AIX

This information was supplied by Dirk Nitschke (mailto:nitschke@math.unihamburg.de), as of August 1997, and describes how to install the **LPRng** package on a workstation running AIX 4.1.x and possibly 3.x.x as well. Dirk would be interested in any comments or corrections.

Printing on AIX systems is different. AIX provides a general queueing facility and printing is only one way to use it. You submit a print job to a print queue using one of the commands `qprt` or `enq`. You can use the BSD or System V printing commands **lpr** or **lp**, too. The `qdaemon` watches all (general) queues and knows how to handle your job. A (general) queue is defined in the file `/etc/qconfig`. The format of this file is different from the `printcap` format.

OK, how to replace the AIX printing system? There is no group `daemon` on AIX. Therefore you have to change the default group for file ownership and process permissions or create a `daemon` user and group. We decided to use the `printq` group; on reflection it would have been easier to have created a `daemon` group. The user `daemon` exists on AIX but we have chosen **lpd** as the user who runs **lpd** and all filters

and owns the spooling directories. You can change the values for `group` and `user` in your `lpd.conf` file or in the sources `src/common/vars.c`. This is an example for `lpd.conf`:

```
# Purpose: group to run SUID ROOT programs
#   default group=daemon
group=printq
# Purpose: server user for SUID purposes
#   default user=daemon
user=lpd
```

Compile and install the **LPRng** package. Create your `printcap`, spooling directories, accounting and logfiles and so on. Don't forget to use checkpc to make sure that all the permissions are set correctly and the necessary files are created.

Then stop all print queues defined on your workstation. Use

```
# chque -q queuename -a "up = FALSE"
```

for this (yes, blanks around = are needed).

If you have local printers attached to your system you will have an **lpd** running. Stop this daemon using SMIT (Print Spooling, Manage Print Server, Stop the Print Server Subsystem). Choosing *both* also removes **lpd** from `/etc/inittab`. Maybe it's faster to do this by hand:

```
h4: {75} # topsrc -p'pid of /usr/sbin/lpd'
h4: {76} # rmitab "lpd"
```

Now delete all print queues managed by `qdaemon` defined on your system. You can use `SMIT` for this or the commands `{mk,ch,rm}que`, `{mk,ch,rm}quedev`, `{mk,ch,rm}virprt`. The `SMIT` fast path is `smit rmpq`.

To start the new **lpd** at system startup you have to add an entry to `/etc/inittab`:

```
h4: {77} # mkitab "lpd:2:once:/full/path/lpd"
```

Some work has to be done if have have a local printer attached to your workstation. You have to create a device file like `/dev/lp0`. The `SMIT` fast path for this is `smit mkdev`. Choose `Printer/Plotter`, then `Printer/Plotter Devices`, then `Add a Printer/Plotter`. To create a parallel printer device select the following:

```
Plotter type:              opp Other parallel printer
Printer/Plotter Interface: parallel
Parent Adapter:            ppa0 Available
```

Now define the characteristics of the device:

```
Port Number: p
```

Option `p` is for parallel. Go to the field:

```
Send all characters to printer UNMODIFIED   no
```

and select `yes`! We have had a lot of trouble with `no`. This is very important! Expect erroneous output if you choose `no`. If you have already created a device file, change the characteristics! SMIT's fast path is `smit chdev`.

Finally remove all AIX printing commands like **qprt**, **lp**, **cancel**, **lpq**, and **lprm**. You will find a lot of them in `/usr/bin`. Do not remove **enq** and friends if you want to use the general queueing facility.

Now you can start your new **lpd**.

3.4. AppleTalk Support

Netatalk is used to communicate from TCP/IP to AppleTalk printers and vice versa. The Netatalk distribution FAQ is at: http://www.umich.edu/~rsug/netatalk. Also, The University of Melbourne web site http://www.cs.mu.oz.au/appletalk/ has additional tutorial and installation information. In addition, Anders Brownworth's Web page http://thehamptons.com/anders/netatalk/ has a useful collection of information for Linux users.

The University of Michigan doesn't seem to be doing anything more for **netatalk**, and Adrian Sun has continued development. You'll find his updates at ftp://ftp.cobaltnet.com/pub/users/asun/testing/ He has improved **netatalk** considerably over the last UMich version. Documentation on **netatalk** is, unfortunately, almost non-existent.

There are a couple of very active mail lists for Netatalk: netatalk-admins@umich.edu (mailto:netatalk-admins@umich.edu) and linux-atalk@netspace.org (mailto:linux-atalk@netspace.org).

After you have installed and gotten **netatalk** working you can use the following AppleTalk configuration file to print from a Macintosh to an **LPRng** printer.

```
Your 32 Character Printer Name:\
        :pr=|/your/path/to/lpr -Pprintername:\
        :op=username.for.printing:\
        :pd=/your/path/to/ppd/files/yourprinter.ppd:

Examples:

Student Printers:\
        :pr=|/usr/bin/lpr -Pstudent:\
        :op=root:\
        :pd=/var/spool/lpd/student/HP4000.PPD:
HP 2500c:\
        :pr=|/usr/bin/lpr -Php2500c:\
        :op=root:\
        :pd=/var/spool/lpd/hp2500c/HP2500.PPD:
```

Chapter 4. Print Spooling Tutorial

A print spooler is a program that accepts *print jobs* (which are usually one or more files) from a program or network interface, stores them in a *spool queue*, and then sends them to a printer or another print spooler. Usually there are facilities to submit jobs, check on the current job status, remove jobs from spool queues, and perform administrative functions such as starting or stopping printing.

A print spooler is a client/server application. The client programs are used to submit jobs to the print spooler program which performs the actual printing operations. In order to carry out these operations, the server may need to use other programs to convert print job files into a format acceptable to a printer, or perform various accounting or administrative functions.

4.1. Overview

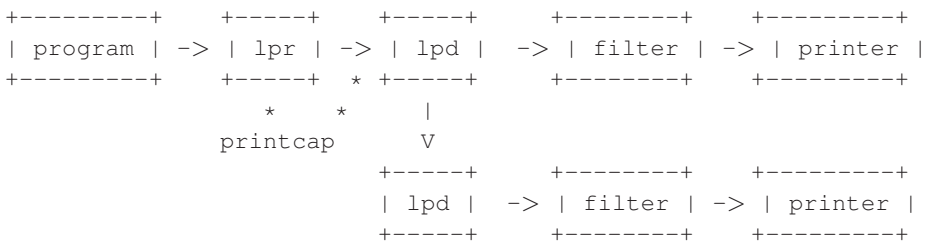

Figure 1

Figure 1 shows the flow of data between the individual components of the **LPRng** print spooling system. A program (or user) will use the **lpr** program to send a file to the **lpd** server over a TCP/IP connection. The **lpd** server will store the file temporarily in a spool queue directory. The information needed by the **lpr** and **lpd** programs to carry out this activity is stored in the `printcap` (usually called the `/etc/printcap`) database file.

The **lpd** server sorts the queue entries and determines the print order. It will select a job to be printed, open a connection to the printer, and then use a *filter* program to convert the file contents into a format suitable for the printer. If the file does not need conversion, then the **lpd** server will send the file directly to the printer.

The **lpd** server can also *forward* jobs to another print server over a network connection, optionally sending them through a filter as well. The destination server can in turn forward the job or send it to a printer.

The protocol or commands used to do this job forward and transfer are specified by RFC1179. This protocol specifies how the **lpr** client program sends a job to the **lpd** server, as well as how the **lpd** server forwards jobs to another server. In addition to job submission, RFC1179 specifies commands to obtain queue status, to remove jobs from the queue, and to start and stop print queues.

4.2. Sample Printcap Entry

As described in the Print Spooling Overview, the information in the `printcap` database is used control printing operations. While there is no RFC specifying its format or content, there is a strong *de facto* standard for its format. For a complete description of the `printcap` database see Printcap Database. For a list of all of the `printcap` and configuration options see
Index To All The Configuration and Printcap Options.

Here is a sample printcap suitable for use by the **LPRng** clients:

```
LPRng use :lp for destination (device and spool queue)
   lp:lp=lp@h4.private
   lpreal:lp=/dev/lp0

Vintage BSD uses :rp:rm for spooler queue and :lp for device
  lp:rp=lp:rm=h4.private
  lpdev:lp=/dev/lp0
```

The `:lp=lp@h4.private` printcap entry tells the *client* programs that jobs for the **lp** printer or print queue are sent to the **lp** print queue queue on host `h4.private`. This can also be specified using the The *legacy* BSD `:rp` and `:rm`. The `:rp=`*queue* option specifies the *print queue* `:rm=`*host* option specifies the host. When both `lp` and `:rp:rm` are present the `:lp` option has precedence.

On the printserver the following is a sample printcap entry suitable for the **lpd** server:

```
lp:
  :lp=/dev/lp0
  :sd=/var/spool/lpd/%P
  :filter=/usr/local/libexec/filters/ifhp
```

The `:sd=`*directory* option (spool queue directory) specifies the directory where print jobs will be placed. The `%P` will be replaced with the name of the spool queue. The `:lp` literal is now the path to the output device the **lpd** server will use to print the job. The `:filter` literal specifies the filter program to use to process the job before sending to the output device.

Here is another example of a printcap entry using the `%P` notation:

```
lp:lp=%P@h4.private
```

This entry will cause all jobs sent to the `lp` spool queue to be sent to the `lp` queue on `server`. `%X` strings in the printcap entries are expanded as shown:

Key	Replaced By
`%P`	printcap entry primary name (printer)
`%Q`	queue requested
`%h`	short host name (host)
`%H`	fully qualified host name (host.dns.whatever)

Key	Replaced By
%R	remote printer (rp value)
%M	remote host (rm value)
%D	date in YYYY-MM-DD format

4.3. Setting Up the Tutorial Configuration

The previous section has given a very high level view of printing operations and shown a sample of some printcap files. In order to do experiment with these LPRng facilities, we will need to be able to modify the printcap information and try various system configurations.

We will use a series of simple printcap entries during this tutorial. We will assume that the **LPRng** system is using the /etc/printcap file. If your system is configured to use another one, then you can make a symbolic link from /etc/printcap or you can simply use your default printcap file.

Save the existing printcap file and then create a new printcap file with the contents as shown below. You will need to have ROOT (superuser) permissions to change the file and perform some of the maintenance operations.

```
h4: {78} # cd /etc
h4: {79} # mv printcap printcap.orig
h4: {80} # vi printcap
#  printcap file contents:
lp:sd=/var/spool/lpd/%P
  :force_localhost
  :lp=/tmp/lp
lp2:sd=/var/spool/lpd/%P
  :force_localhost
  :lp=/tmp/lp2
h4: {81} # #  set permissions so everybody can read file
h4: {82} # chmod 666 printcap
```

We save the original printcap file and create a new one. We give the file world writable permissions so that later we can modify this file without needing to have root permissions. The printcap file has two entries: lp and lp2. Each print queue on the server needs a spool file to hold print jobs, jobs, and the :sd value specifies its location. The %P value is replaced with the name of the printer when it is used. In classical BSD operation each host has an **lpd** print spooler running on the local host (we use localhost in this manual for simplicity). Files were copied to spool directories on the localhost and then then print spooler would send them to the destination, which could be another print spooler. This meant that each localhost machine had to have a print spooler and spool queue directory structure. Management of this becomes very difficult in large organizations. The force_localhost forces this mode of operation and means that the **lpd** server and clients must run on the same host.

We use files for the output devices (:lp=) so that we can see easily view the output (and also to save trees). We will also need to have some simple test files. Create the files using the following commands.

```
h4: {83} # cp /dev/null /tmp/lp
h4: {84} # cp /dev/null /tmp/lp2
h4: {85} # chmod 666  /tmp/lp /tmp/lp2
h4: {86} # echo hi >/tmp/hi
h4: {87} # echo there >/tmp/there
```

We will use a *dummy* `lpd.perms` file that allows all users to do anything. This is useful for testing, but dangerous in a working environment.

```
h4: {88} # #  we modify the lpd.perms to allow an ordinary user to control
h4: {89} # mv lpd.perms lpd.perms.orig
h4: {90} # echo "DEFAULT ACCEPT" >lpd.perms
h4: {91} # chmod 666 lpd.perms
```

Finally we run `checkpc` to make sure that our printcap is correct and to create the necessary spool directories:

```
h4: {92} # checkpc -f -V
Checking printer 'lp'
 Checking directory: '/var/spool/lp'
   directory '/var'
   directory '/var/spool'
   directory '/var/spool/lp'
 Warning -   changing ownership '/var/spool/lp' to 1/1
   checking 'control.lp' file
   checking 'status.lp' file
   checking 'status' file
   cleaning 'status' file, 0K bytes: no truncation
   checking 'log' file
   cleaning 'log' file, 0K bytes: no truncation
   checking 'acct' file
   cleaning 'acct' file, 0K bytes: no truncation
Checking printer 'lp2'
  Checking directory: '/var/spool/lp2'
    directory '/var'
  ....
h4: {93} # lpc reread
h4: {94} # lpd
h4: {95} # lpq
Printer: lp@h4
   Queue: no printable jobs in queue
```

Checkpc performs consistency checks on the printcap file and spool queue entries. The `checkpc -f` (fix) option will change permissions and create directories and can only be executed by ROOT. **Checkpc** has other functions as well - you can view printcap information, see default configuration values, and remove junk files from spool queues with it.

The `lpc reread` command sends a request to the **lpd** server to reread the configuration and printcap information. The **lpd** command is added as insurance in case your **lpd** server is not running. The `exit` command restores ordinary user privileges, and the **lpq** command is used to check that the server is running. Finally, we check to see that the `lpc reread` command is accepted from an ordinary user.

4.4. Restoring Original Configuration

To restore the original configuration, you simply need to restore the original `printcap` and `lpd.perms` file and then restart the **lpd** server. These operations require ROOT permissions.

```
h4: {96} # su
h4: {97} # cd /etc    OR    cd /usr/local/etc
h4: {98} # mv printcap.orig printcap
h4: {99} # mv lpd.perms.orig lpd.perms
h4: {100} # checkpc -f
h4: {101} # lpc reread
h4: {102} # rm -rf /var/spool/lpd/lp /var/spool/lpd/lp2
h4: {103} # rm -f /tmp/lp /tmp/lp2
```

4.5. Printing a File and Checking Status

Try the following commands. The commands appear after the prompt, and sample output that you might see is shown.

```
h4: {104} % lpr -V /tmp/hi
Version LPRng-3.6.14
sending job 'papowell@h4+238' to lp@localhost
connecting to 'localhost', attempt 1
connected to 'localhost'
requesting printer lp@localhost
sending control file 'cfA238h4.private' to lp@localhost
completed sending 'cfA238h4.private' to lp@localhost
sending data file 'dfA238h4.private' to lp@localhost
completed sending 'dfA238h4.private' to lp@localhost
done job 'papowell@h4+238' transfer to lp@localhost
```

The `lpr -V` (Verbose) option causes **lpr** to print status output. As you can see from the above lines, it first tries to connect to the **lpd** server on host `localhost`, then sends a print request (which is accepted), then sends a *control* file containing information about the job and a *data* file or files which are copies of the files to be printed.

If you check the `/tmp/lp` file and you will find that a copy of `/tmp/hi` has been written to it. By default, the **lpd** print spooler acts as a store and forward system, accepting files to be printed, holding them in the print queue, and then forwarding them to the destination system or output device.

You can use the **lpq** command to view the status of the print job.

```
h4: {105} % lpq
Printer: lp@h4
 Queue: no printable jobs in queue
 Status: job 'papowell@h4+238' removed at 09:39:03.256
```

If you want to see more status information, use `lpq -l`, `lpq -ll`, or even `lpq -L`. The `-L` provides alL the status.

```
h4: {106} % lpq -l
Printer: lp@h4
 Queue: no printable jobs in queue
 Status: lp@h4.private: job 'papowell@h4+238' printed at 09:39:03.112
 Status: job 'papowell@h4+238' removed at 09:39:03.256
h4: {107} % lpq -ll
Printer: lp@h4
 Queue: no printable jobs in queue
 Status: finished 'papowell@h4+238', status 'JSUCC' at 09:39:03.108
 Status: subserver pid 8240 exit status 'JSUCC' at 09:39:03.110
 Status: lp@h4.private: job 'papowell@h4+238' printed at 09:39:03.112
 Status: job 'papowell@h4+238' removed at 09:39:03.256
h4: {108} % lpq -L
Printer: lp@h4
 Queue: no printable jobs in queue
 Status: subserver pid 8240 starting at 09:39:03.105
 Status: accounting at start at 09:39:03.105
 Status: opening device '/tmp/lp' at 09:39:03.105
 Status: printing job 'papowell@h4+238' at 09:39:03.106
 Status: no banner at 09:39:03.107
 Status: printing data file 'dfA238h4.private', size 3 at 09:39:03.107
 Status: printing done 'papowell@h4+238' at 09:39:03.107
 Status: accounting at end at 09:39:03.108
 Status: finished 'papowell@h4+238', status 'JSUCC' at 09:39:03.108
 Status: subserver pid 8240 exit status 'JSUCC' at 09:39:03.110
 Status: lp@h4.private: job 'papowell@h4+238' printed at 09:39:03.112
 Status: job 'papowell@h4+238' removed at 09:39:03.256
```

There are different status formats available as well. The `lpq -s` (summary) produces a single line of status per spool queue, while the `lpq -v` (verbose) produces output that is very suitable for processing with programs such as **Perl** or **awk**:

```
h4: {109} % lpq -s
lp@h4  0 jobs
h4: {110} % lpq -v
```

```
Printer: lp@h4
 Printing: no
 Aborted: no
 Spooling: no
 Queue: no printable jobs in queue
 SPOOLCONTROL=
 Status: subserver pid 8240 starting at 09:39:03.105
 Status: accounting at start at 09:39:03.105
 Status: opening device '/tmp/lp' at 09:39:03.105
 Status: printing job 'papowell@h4+238' at 09:39:03.106
 Status: no banner at 09:39:03.107
 Status: printing data file 'dfA238h4.private', size 3 at 09:39:03.107
 Status: printing done 'papowell@h4+238' at 09:39:03.107
 Status: accounting at end at 09:39:03.108
 Status: finished 'papowell@h4+238', status 'JSUCC' at 09:39:03.108
 Status: subserver pid 8240 exit status 'JSUCC' at 09:39:03.110
 Status: lp@h4.private: job 'papowell@h4+238' printed at 09:39:03.112
 Status: job 'papowell@h4+238' removed at 09:39:03.256
```

If you check the `/tmp/lp` file and you will find that a copy of `/tmp/hi` has been written to it. By default, the **lpd** print spooler acts as a store and forward system, accepting files to be printed, holding them in the print queue, and then forwarding them to the destination system or output device.

4.6. Selecting the Print Queue

In the previous section we used the default print queue. How does **LPRng** determine what print queue to use? First, you can explicitly specify the printer using the `lpq -Pprintqueue` option and the `lpq -a` or `lpq -Pall` to select all print queues:

```
h4: {111} % lpq -Plp
Printer: lp@h4
 Queue: no printable jobs in queue
h4: {112} % lpq -Plp2
Printer: lp2@h4
 Queue: no printable jobs in queue
h4: {113} % lpq -a
Printer: lp@h4
 Queue: no printable jobs in queue
Printer: lp2@h4
 Queue: no printable jobs in queue
```

You can combine the `lpq -a` with the `lpq -s` option for a summary listing:

```
h4: {114} % lpq -a
Printer: lp@h4
 Queue: no printable jobs in queue
```

```
Printer: lp2@h4
 Queue: no printable jobs in queue
h4: {115} % lpq -s -a
lp@h4  0 jobs
lp2@h4  0 jobs
```

There is another way to explicitly specify the printqueues listed by `lpq -a`; see the `all` Printcap Entry for details.

Users can set their default printer by using the PRINTER (highest priority), LPDEST (next), NPRINTER (next), and NGPRINTER (lowest priority), environment variables. For example:

```
h4: {116} % setenv PRINTER lp2
h4: {117} % lpq
Printer: lp2@h4
 Queue: no printable jobs in queue
h4: {118} % unsetenv PRINTER
h4: {119} % lpq
Printer: lp@h4
 Queue: no printable jobs in queue
```

If the printer is not specified on the command line or by the environment variables, then the first printer in the printcap database will be used and then the default printer in the configuration database, and finally the compile time default. Printer and Server Information for details.

4.7. Controlling the Print Queue

The **lpc** command is used to examine and control the print server operation. The `lpc status` command displays the administrative status of a print queue. The **lpd** program caches status and job information in order to improve performance. The `lpc flush` command will flush the cached information and cause the server to regenerate it. The `lpc enable` and `lpc disable` commands enable or disable spooling to the print queue, and the `lpc stop` and `lpc start` commands stop and start printing (or transfers) of jobs in the print queue.

Let's look at the status displayed when we use these commands:

```
h4: {120} % lpc status
 Printer  Printing Spooling Jobs Server Subserver Redirect Status/(Debug)
lp@h4      enabled  enabled    0   none    none
h4: {121} % lpc status all
 Printer  Printing Spooling Jobs Server Subserver Redirect Status/(Debug)
lp@h4      enabled  enabled    0    none    none
lp2@h4     enabled  enabled    0    none    none
h4: {122} % lpc
lpc>status
 Printer  Printing Spooling Jobs Server Subserver Redirect Status/(Debug)
lp@h4      enabled  enabled    0    none    none
```

```
lpc>status all
 Printer  Printing Spooling Jobs Server Subserver Redirect Status/(Debug)
lp@h4      enabled  enabled    0    none    none
lp2@h4     enabled  enabled    0    none    none
lpc>quit
```

The **lpc** command can be used in command line or interactive mode as shown above. When used with no parameters it will run in interactive mode, reading one or more commands from its standard input (STDIN). The `lpc status` command shows the administrative status of the select print queue. The `all` queue name selects all print queues for display. As shown in the above example, both print queues have printing and spooling enabled and there are no jobs in the print queue. The *Server* and *Subserver* information shows if there is a process which is printing jobs, and its helper process that does the actual communication with the printer.

It might be puzzling at first why **LPRng** uses two processes for this operation, but the reason is very simple. Many operating system implementations have *memory leaks* that cause the actual process size to grow as it runs. This is especially true if a large number of databases such as the password, Domain Name Server, or other system database is consulted frequently with different queries. Since this is usually done quite a lot by the process which deals with the actual printing, the printing process would soon grow very large and then die when it could no longer obtain more memory. The *Server* process will fork or create a child process *Subserver* process that is responsible for the printing of a single job. When the job printing has been completed, the *Subserver* process will exit and the *Server* process will then create another child until there are no more jobs to be printed. The *Redirect* and *Debug* fields will be discussed in later sections.

Now let's use the basic spool queue control commands:

```
h4: {123} % lpc disable
Printer: lp@h4
lp@h4.private: disabled
h4: {124} % lpq
Printer: lp@h4  (spooling disabled)
 Queue: no printable jobs in queue
h4: {125} % lpc enable
Printer: lp@h4
lp@h4.private: enabled
h4: {126} % lpq
Printer: lp@h4
 Queue: no printable jobs in queue
h4: {127} % lpc stop
Printer: lp@h4
lp@h4.private: stopped
h4: {128} % lpq
Printer: lp@h4  (printing disabled)
 Queue: no printable jobs in queue
h4: {129} % lpc start
Printer: lp@h4
lp@h4.private: started
h4: {130} % lpq
Printer: lp@h4
```

```
 Queue: no printable jobs in queue
```

As we can see, the **lpc** command also reports on the status of the print queue. Let's see what happens when we print to a stopped queue:

```
h4: {131} % lpc stop
Printer: lp@h4
lp@h4.private: stopped
h4: {132} % lpr /tmp/hi
h4: {133} % lpr /tmp/hi /tmp/there
h4: {134} % lpq
Printer: lp@h4  (printing disabled)
 Queue: 2 printable jobs
 Server: no server active
 Rank   Owner/ID               Class Job Files            Size Time
1      papowell@h4+17920         A 17920 /tmp/hi             3 18:14:22
2      papowell@h4+17922         A 17922 /tmp/hi,/tmp/there  9 18:14:30
h4: {135} % lpc status
 Printer  Printing Spooling Jobs  Server Subserver Redirect Status/(Debug)
lp@h4     disabled  enabled    2    none    none
```

The **lpc** status shows that we have two jobs spooled. The *Rank* field shows the order, the `Owner/ID` shows the unique job ID that is assigned to the job and the *Class* field is the job class (this may be changed with the `lpr -C class` option). The *Job* field shows the *job number* assigned to this job in this particular spool queue. While the ID value never changes as a job moves through the **LPRng** system, the *job number* is specific to a particular spool queue and may change if a job is *forwarded* to another spool queue that has a job with the same job number. The *Size* field is the total number of printable bytes in the job, and the *Time* field shows the timestamp associated with the job.

Now let's start the print queue and watch what happens.

```
h4: {136} % lpc start
Printer: lp@h4
lp@h4.private: started
h4: {137} % lpq
Printer: lp@h4
 Queue: 2 printable jobs
 Server: pid 17928 active
 Unspooler: pid 17929 active
 Status: opening device '/tmp/lp' at 18:14:43.921
 Rank   Owner/ID             Class Job Files            Size Time
active papowell@h4+17920       A 17920 /tmp/hi             3 18:14:22
2      papowell@h4+17922       A 17922 /tmp/hi,/tmp/there  9 18:14:30
h4: {138} % lpq -ll
Printer: lp@h4
 Queue: 2 printable jobs
 Server: pid 17928 active
 Unspooler: pid 17929 active
 Status: printing job 'papowell@h4+17920' at 18:14:43.921
```

```
 Status: no banner at 18:14:43.921
 Status: printing data file 'dfA017920h4.private', size 57 at 18:14:43.922
 Rank   Owner/ID             Class Job Files            Size Time
active papowell@h4+17920       A 17920 /tmp/hi             3 18:14:22
2      papowell@h4+17922       A 17922 /tmp/hi,/tmp/there  9 18:14:30
```

The *Rank* value of the first job has been changed to `active` and there is new *Status* information. If we use `lpq -ll` we can see the times that the various print operations are carried out, and details of their success or failure.

We can also use the **lpc** command to see the status of a particular job. We can select jobs by the user name, the ID, or the job number. For example:

```
h4: {139} % lpc stop
Printer: lp@h4
lp@h4.private: stopped
h4: {140} % echo hi |lpr
h4: {141} % echo there | lpr
h4: {142} % echo test |lpr
h4: {143} % lpq
Printer: lp@h4  (printing disabled)
 Queue: 3 printable jobs
 Server: no server active
 Status: job 'papowell@h4+17922' removed at 18:15:13.981
 Rank   Owner/ID            Class Job Files           Size Time
1      papowell@h4+17959      A 17959 (stdin)            3 18:23:24
2      papowell@h4+17962      A 17962 (stdin)            6 18:23:30
3      papowell@h4+17970      A 17970 (stdin)            5 18:23:35
h4: {144} % lpq 17970
Printer: lp@h4  (printing disabled)
 Queue: 3 printable jobs
 Server: no server active
 Status: job 'papowell@h4+17922' removed at 18:15:13.981
 Rank   Owner/ID            Class Job Files           Size Time
3      papowell@h4+17970      A 17970 (stdin)            5 18:23:35
h4: {145} % lpq papowell
Printer: lp@h4  (printing disabled)
 Queue: 3 printable jobs
 Server: no server active
 Status: job 'papowell@h4+17922' removed at 18:15:13.981
 Rank   Owner/ID            Class Job Files           Size Time
1      papowell@h4+17959      A 17959 (stdin)            3 18:23:24
2      papowell@h4+17962      A 17962 (stdin)            6 18:23:30
3      papowell@h4+17970      A 17970 (stdin)            5 18:23:35
h4: {146} % lpq -s 17970
lp@h4  1 jobs
h4: {147} % lpq -s papowell
lp@h4  3 jobs
h4: {148} % lpq -s nobody
lp@h4  0 jobs
```

We use `lpq -Pqueuename` to select a specific print queue and `lpq -a` or `lpq -Pall` to select all queues:

```
h4: {149} % lpc -a stop
Printer: lp@h4
lp@h4.private: stopped
Printer: lp2@h4
lp2@h4.private: stopped
h4: {150} % lpc -Pall start
Printer: lp@h4
lp@h4.private: started
Printer: lp2@h4
lp2@h4.private: started
```

You can use the **lpc** command in *interactive* mode:

```
h4: {151} % lpc
lpc>status
 Printer  Printing Spooling Jobs  Server Subserver Redirect Status/(Debug)
lp@h4      enabled  enabled    3    17990   17993
lpc>status all
 Printer  Printing Spooling Jobs  Server Subserver Redirect Status/(Debug)
lp@h4      enabled  enabled    3    17990   17993
lp2@h4     enabled  enabled    3    none    none
lpc>stop lp
Printer: lp@h4
lp@h4.private: stopped
lpc>start lp
Printer: lp@h4
lp@h4.private: started
lpc>quit
```

The `lpc topq` command can be used to put a job (or jobs) at the head of the spool queue. This command is very useful when some job requires priority service. You can select the job by using the job number or the job ID.

```
h4: {152} % lpc topq lp 17970
Printer: lp@h4
lp: selected 'papowell@h4+17970'
lp@h4.private: started
h4: {153} % lpq
Printer: lp@h4
 Queue: 3 printable jobs
 Server: pid 17999 active
 Rank   Owner/ID          Class Job Files      Size Time
active papowell@h4+17970    A 17970 (stdin)      5 18:23:35
1      papowell@h4+17959    A 17959 (stdin)      3 18:23:24
```

```
2      papowell@h4+17962    A 17962 (stdin)      6 18:23:30
```

4.8. Job Removal

Occasionally we print a file and then change our mind and want to cancel the job. The **lprm** command allows us to do this.

```
h4: {154} % lpq
Printer: lp@h4  (printing disabled)
 Queue: 3 printable jobs
 Server: no server active
 Status: job 'papowell@h4+17922' removed at 18:15:13.981
 Rank   Owner/ID          Class Job Files      Size Time
1      papowell@h4+17959    A 17959 (stdin)       3 18:23:24
2      papowell@h4+17962    A 17962 (stdin)       6 18:23:30
3      papowell@h4+17970    A 17970 (stdin)       5 18:23:35
h4: {155} % lprm
Printer lp@h4:
  checking perms 'papowell@h4+17959'
  dequeued 'papowell@h4+17959'
h4: {156} % lpq
Printer: lp@h4  (printing disabled)
 Queue: 2 printable jobs
 Server: no server active
 Status: job 'papowell@h4+17922' removed at 18:15:13.981
 Rank   Owner/ID          Class Job Files      Size Time
1      papowell@h4+17962    A 17962 (stdin)       6 18:23:30
2      papowell@h4+17970    A 17970 (stdin)       5 18:23:35
h4: {157} % lprm 17970
Printer lp@h4:
  checking perms 'papowell@h4+17970'
  dequeued 'papowell@h4+17970'
h4: {158} % lpq
Printer: lp@h4  (printing disabled)
 Queue: 1 printable job
 Server: no server active
 Status: job 'papowell@h4+17922' removed at 18:15:13.981
 Rank   Owner/ID          Class Job Files     Size Time
1      papowell@h4+17962    A 17962 (stdin)      6 18:23:30
```

By default, the **lprm** command removes the first job in the queue that the user has permission to remove. Also, as shown in the example, you can remove a job by specifying the job ID or the job number. If you specify a user name, you remove `all` of the user's jobs. This can be dangerous:

```
h4: {159} % lpq
Printer: lp@h4  (printing disabled)
```

```
 Queue: 3 printable jobs
 Server: no server active
 Status: job 'papowell@h4+17922' removed at 18:15:13.981
 Rank   Owner/ID          Class Job Files     Size Time
1      papowell@h4+17962    A 17962 (stdin)      6 18:23:30
2      papowell@h4+18499    A 18499 /tmp/hi      3 18:56:00
3      papowell@h4+18501    A 18501 /tmp/there   6 18:56:02
h4: {160} % lprm papowell
Printer lp@h4:
  checking perms 'papowell@h4+17962'
  dequeued 'papowell@h4+17962'
  checking perms 'papowell@h4+18499'
  dequeued 'papowell@h4+18499'
  checking perms 'papowell@h4+18501'
  dequeued 'papowell@h4+18501'
h4: {161} % lpq
Printer: lp@h4  (printing disabled)
 Queue: no printable jobs in queue
 Status: job 'papowell@h4+17922' removed at 18:15:13.981
```

The special user `all` matches all jobs in a print queue. Clearly you should be careful not to specify `lprm all` by accident. Even more dangerous is the following command:

```
h4: {162} % lprm -a all
```

As you might surmise, this removes `all` print jobs in `all` queues, which is an excellent way to purge print queues of all jobs.

4.9. Print Job Filters

A printer usually understands one or more *Print Job Languages*. Files sent to this printer must be in one of these languages and have the appropriate *job format*. The most common Print Job Languages are PostScript and PCL. Text files are PCL with no special PCL control sequences.

In order for a printer to reliably print a job it needs to be reset to a known configuration and then at the end of job having it flush all of the output to the printing device. This is done by sending it *start of job* and *end of job* commands. These commands differ from printer to printer and depend on the print job language as well. Some *vintage* line printers also have a set of proprietary *escape sequences* that are used to set up margins, form size, and other printing characteristics. Usually a *setup string* with these escape sequences must be sent to the printer before a file can be printed.

When sending a job to the printer the print spooler will first process the job using a *print job filter* or *filter* program. This program reads the job file and then produces output in the format required for the printer.

When a print job is created the files in the print job are assigned a *format*. This format was meant as a guide to the print spooler and was to be used to select the filter program for the files in the job. The format was a lower case letter; the `f` is the default format and indicates normal processing and the `l`

format indicates a literal or binary file. Job files that are flagged as having literal or binary format are usually passed directly to the printer or have at the most a minimal amount of processing. See Print Job Formats for more information about formats and their use with filters.

There are two ways to specify filters: the default `:filter=...` option and the more specific `Xf=...` option. The `X` is a lower case letter corresponding to a format. Here is a sample printcap entry with a filter specification:

```
lp:sd=/var/spool/lpd/%P
  :filter=/usr/local/lib/filters/ifhp
  :rf=/usr/local/lib/filters/rfilter
```

All jobs with formats other than `r` will be processed using the `ifhp` program while the jobs with the `r` format will be processed using the `rfilter` program.

We will set up a very simple filter and use it to demonstrate how filtering is done by the **lpd** print spooler. First, set up the `/tmp/testf` file as shown below.

```
#!/bin/sh
# /tmp/testf - test filter for LPRng
PATH=/bin:/usr/bin; export PATH
echo TESTF $0 "$@" >&2
echo TESTF $0 "$@"
echo ENV
set
echo LEADER
/bin/cat
echo TRAILER
exit 0
```

Let us carefully examine the script line by line. The first couple of lines are *boilerplate*. You should *always* set the PATH value in a filter script or use full pathnames for executable programs. This is a good practice as it ensures that only the specified directories will be searched for commands.

The next lines echo the command line arguments to file descriptor 2 (STDERR) and to STDOUT. We will soon see how this information is displayed by the **LPRng** software. We then use the `set` command to list the shell variables to STDOUT, print LEADER to STDOUT, copy STDIN to STDOUT, and print TRAILER to STDOUT. We exit with a zero result code.

We can test our script, with the results shown below:

```
h4: {163} % chmod 755 /tmp/testf
h4: {164} % echo hi |/tmp/testf -a1
TESTF /tmp/testf -a1
TESTF /tmp/testf -a1
ENV
USER=papowell
HOSTNAME=h4
...
PATH=/bin:/usr/bin
LEADER
```

```
hi
TRAILER
```

Let's now use this filter. Edit the `lp` printcap entry so it has contents indicated below, use `checkpc -f` to check the printcap, and then use `lpc reread` to restart the **lpd** server.

```
lp:sd=/var/spool/lpd/%P
  :force_localhost
  :lp=/tmp/lp
  :filter=/tmp/testf
```

Execute the following commands to print the `/tmp/hi` file and observe the results:

```
h4: {165} % cp /dev/null /tmp/lp
h4: {166} % lpr /tmp/hi
h4: {167} % lpq -llll
Printer: lp@h4
 Queue: no printable jobs in queue
 Status: lp@h4.private: job 'papowell@h4+26593' printed at 21:37:21.312
 Status: job 'papowell@h4+26593' removed at 21:37:21.323
 Status: subserver pid 26683 starting at 21:39:21.908
 Status: accounting at start at 21:39:21.908
 Status: opening device '/tmp/lp' at 21:39:21.909
 Status: printing job 'papowell@h4+26681' at 21:39:21.909
 Status: no banner at 21:39:21.909
 Status: printing data file 'dfA026681h4.private', size 3, \
    IF filter 'testf' at 21:39:21.909
 Status: IF filter msg - 'TESTF /tmp/testf -Apapowell@h4+26681 \
     -CA -D2000-04-11-21:39:21.877 -Ff -Hh4.private -J/tmp/hi \
      -Lpapowell -Plp -Qlp -aacct -b3 -d/var/tmp/LPD/lp \
      -edfA026681h4.private -f/tmp/hi -hh4.private -j026681 \
      -kcfA026681h4.private -l66 -npapowell -sstatus \
      -t2000-04-11-21:39:21.000 -w80 -x0 -y0 acct' \
       at 21:39:21.914
 Status: IF filter finished at 21:39:22.070
 Status: printing done 'papowell@h4+26681' at 21:39:22.070
 Status: accounting at end at 21:39:22.070
 Status: finished 'papowell@h4+26681', status 'JSUCC' at 21:39:22.070
 Status: subserver pid 26683 exit status 'JSUCC' at 21:39:22.072
 Status: lp@h4.private: job 'papowell@h4+26681' printed at 21:39:22.072
 Status: job 'papowell@h4+26681' removed at 21:39:22.085
h4: {168} % more /tmp/lp
TESTF /tmp/testf -Apapowell@h4+26681 -CA -D2000-04-11-21:39:21.877 \
 -Ff -Hh4.private -J/tmp/hi -Lpapowell -Plp -Qlp -aacct -b3 \
 -d/var/tmp/LPD/lp -edfA026681h4.private -f/tmp/hi -hh4.private \
 -j026681 -kcfA026681h4.private -l66 -npapowell -sstatus \
 -t2000-04-11-21:39:21.000 -w80 -x0 -y0 acct
ENV
USER=papowell
```

```
LD_LIBRARY_PATH=/lib:/usr/lib:/usr/5lib:/usr/ucblib
HOME=/home/papowell
PRINTCAP_ENTRY=lp
 :force_localhost
 :filter=/tmp/testf
 :lp=/var/tmp/lp
 :sd=/var/tmp/LPD/lp

PS1=$
OPTIND=1
PS2=>
SPOOL_DIR=/var/tmp/LPD/lp
LOGNAME=papowell

CONTROL=Hh4.private
 Ppapowell
 J/tmp/hi
 CA
 Lpapowell
 Apapowell@h4+15850
 D2000-04-26-18:13:55.505
 Qlp
 N/tmp/hi
 fdfA015850h4.private
 UdfA015850h4.private

PATH=/bin:/usr/bin
SHELL=/bin/sh
LOGDIR=/home/papowell
IFS=
PRINTER=lp
LEADER
test Test
TRAILER
```

The `cp` command clears out the `/tmp/lp` file we are using as a dummy output device. The `lpr` command prints the `/tmp/hi` file and the `lpq -llll` command shows the status information. The status information now contains the line that the `testf` script wrote to STDERR. The **lpd** server captures filter STDERR messages and puts it them in the spool queue status file.

As we see from the lpq status, **lpd** passes a large number of command line options to our filter. These options and their meanings are discussed in detail in
Filter Command Line Options and Environment Variables. We will discuss these in more detail in the next section.

If we look at the `/tmp/lp` file, we see the command line options and values of the shell variables. For a full discussion of the environment variables passed to a filter see
Filter Command Line Options and Environment Variables. The more interesting environment variables include PRINTCAP_ENTRY - the printcap entry for this printer, CONTROL - the control file, and HF - the hold file.

4.9.1. Control Files and Filter Options

When you transfer a print job the **lpd** print spooler will send the job as two or more files: *control* file that contains information about the job and *data* files that contain the information to be printed. Here is sample control file:

```
Hh4.private
Ppapowell
J/tmp/hi
CA
Lpapowell
Apapowell@h4+15850
D2000-04-26-18:13:55.505
Qlp
N/tmp/hi
fdfA015850h4.private
UdfA015850h4.private
```

Lines starting with upper case letters contain job information such as the user who submitted the job. Lines starting with lower case letters indicate the data file to be printed and the corresponding format. For full details about the exact format of the control file see Job Files.

Table 4-1 shows the correspondence between lines in the control file and **lpr** command line options. The `N` values are the names of the files that are printed. The `U` indicates a data file is in a job and is present to meet RCF1179 and *vintage* print spooler requirements.

Table 4-1. Filter Options

Control File	Filter Option	Purpose or Value
	`-P`*Printer*	Print queue name - printcap information
H	`-H`*Host*	Host Name
`P`	`-n`*User*	User Login Name of job originator
`J`	`-J`*Job name*	lpr -J option or file name
`C`	`-C`*Class*	Print class (lpr -C option)
`L`	`-L`*Banner*	Banner page request
`A`	`-A`*Jobid*	Job Id
`D`	`-D`*Date*	Date or time information
`Q`	`-Q`*Queue*	Original Print queue job was sent to
`N`	`-N`*Filename*	Filename
`f,l,p,...`	`-F`*f*	Datafile format
`U`		Datafile (historical)

When a print filter processes these jobs the values in the control file are passed on the command line as

options starting with upper case letters:

```
/tmp/testf -Apapowell@h4+26681 -CA \
   -D2000-04-11-21:39:21.877 -Ff -Hh4.private ....
```

Sometimes we want to pass only a small subset of these command line options to a filter or provide them in a specific order in order to be compatible with *legacy* print filters. **LPRng** provides several different ways to do this and we will explore how to control command line options.

If the filter entry starts with `-$`, this suppresses the automatic addition of command line options; we can then add our own options to the command line. Modify the printcap entry to have the following form:

```
lp:sd=/var/spool/lpd/%P
  :force_localhost
  :lp=/tmp/lp
  :filter= -$ /tmp/testf '$P' $0P -X$-P ${lp} G\072 or \:
```

Lets print our `/tmp/hi` test file and then look at the **lpq** status:

```
h4: {169} % cp /dev/null /tmp/lp
h4: {170} % lpr /tmp/hi
h4: {171} % lpq -llll
Printer: lp@h4
 ....
 Status: IF filter msg - 'TESTF /tmp/testf -Plp -P lp -Xlp \
    -Ylp /tmp/lp G: or :' at 01:20:21.560
```

The `-$` suppresses the adding the default literals to the filter command line. You can pass specific options using `$X`; if the option has a non-null value then it will be expanded in the following format:

```
Option    Value Expansion
$X        -X<value>
$'X       -X'<value>'
$0X       -X '<value>'
$-X       <value>
$'-X      '<value>'
${X}      control file X option value
$'{X}     control file X option value (in quotes)
${name}   printcap option value if value nonzero length
$'{name}  printcap option value if value nonzero length in quotes
\nnn      single printable character
$*        all options in control file expanded using $X
```

Command line options can be grouped and passed as a single argument by enclosing them in single or double quotes. You should be aware that **LPRng** has an *extremely* primitive way of handling quotes. When the `/bin/sh -c` parameter is not used, the the command line is broken on spaces and each unit is passed as an individual argument. If the first character after a space is a quote (single or double), the next

quote is found, and then entire element is then used as a single parameter. Substitution of `$X` parameters is then done. As a special case, when you have a `$0X`, this causes a split and all of the string previous and including the `-X` flag is passed as a single option and all of the option value and following are passed as another option. If the result of the expansion is a zero length parameter then it is removed from the parameter list. When the `/bin/sh -c` is used the command line is not broken, and all non-empty option values are enclosed in single quotes.

The `${name}` option is used to pass a printcap option value. For example, you can pass the value of the printcap option `form` as shown below. You can experiment with this by using the `/tmp/testf` filter and printcap shown below.

```
printcap:
lp:sd=/var/spool/lpd/%P
  :force_localhost
  :lp=/tmp/lp
  :filter=/tmp/testf -F ${form}
  :form=payroll

h4: {172} % cp /dev/null /tmp/lp
h4: {173} % lpr /tmp/hi
h4: {174} % lpq -llll
Printer: lp@h4
 ...
 Status: IF filter msg - 'TESTF /tmp/testf -F payroll' at 09:55:31.276
 ...
```

If we have a *legacy* print filter that was originally written for the BSD print spooler, then we may find that it requires a small number of command line options in a very specific order. We can use the `:bkf` (BSD Kompatible Filter or BacKwards compatible Filter) flag to pass suitable options. Modify the printcap entry to have the following form:

```
lp:sd=/var/spool/lpd/%P
  :force_localhost
  :lp=/tmp/lp
  :filter=/tmp/testf
  :bk
```

Lets print our `/tmp/hi` test file and then look at the **lpq** status:

```
h4: {175} % cp /dev/null /tmp/lp
h4: {176} % lpr /tmp/hi
h4: {177} % lpq -llll
Printer: lp@h4
 ....
 Status: IF filter msg - 'TESTF /tmp/testf -Plp -w80 -l66 \
    -x0 -y0 -Ff -Lpapowell -J/tmp/hi -CA -n papowell \
    -h h4.private acct' at 08:07:46.583
```

Finally, there are times when we would like the print filter to be a simple shell command or to chain several programs together in a simple pipeline. While this is possible using a print filter, you can also do this in the filter specification. If your filter specification starts with a parenthesis (`(`) or contains the IO redirection for pipeto (`|`), input redirection (<), or output redirection (>) then the **lpd** server will use the `:shell` configuration option value (default `/bin/sh`) and execute it using:

```
${shell} -c "( ${if} )"
```

If this is done, then no command line options are added to the command. However, expansion of `$X` parameters are still done. Modify the printcap entry to have the following form:

```
lp:sd=/var/spool/lpd/%P
  :force_localhost
  :lp=/tmp/lp
  :filter=(echo "PREAMBLE"; /tmp/testf; echo "APPENDIX")
```

Lets print our `/tmp/hi` test file and then look at the **lpq** status:

```
h4: {178} % cp /dev/null /tmp/lp
h4: {179} % lpr /tmp/hi
h4: {180} % lpq -llll
Printer: lp@h4
 ....
 Status: printing data file 'dfA018881h4.private', size 3, \
   IF filter 'echo' at 09:22:11.476
 Status: IF filter msg - 'TESTF /tmp/testf' at 09:22:11.510
 Status: IF filter finished at 09:22:11.514
```

If we examine the `/tmp/lp` file we find:

```
PREAMBLE
TESTF /tmp/testf
ENV
USER=papowell
LD_LIBRARY_PATH=/lib:/usr/lib:/usr/5lib:/usr/ucblib
...
PRINTER=lp
LEADER
hi
TRAILER
APPENDIX
```

As we expected, no options were passed on the command line. If the printcap is modified to have the following contents, then you will see:

```
lp:sd=/var/spool/lpd/%P
  :force_localhost
  :lp=/tmp/lp
  :filter=(echo "PREAMBLE"; /tmp/testf $*; echo "APPENDIX")

h4: {181} % lpr /tmp/hi
h4: {182} % lpq -llll
Printer: lp@h4
 ....
 Status: IF filter msg - 'TESTF /tmp/testf -Apapowell@h4+18941 \
   -CA -D2000-04-29-09:27:30.700 -Ff -Hh4.private -J/tmp/hi \
   -Lpapowell -Plp -Qlp -aacct -b3 -d/var/tmp/LPD/lp \
   -edfA018941h4.private -f/tmp/hi -hh4.private -j018941 \
   -kcfA018941h4.private -l66 -npapowell -sstatus \
   -t2000-04-29-09:27:30.864 -w80 -x0 -y0 acct' at 09:27:30.879
```

Using the shell invocation is especially useful when you may have a parameter that has an empty string value, and need to pass this as a command line parameter. Modify the `/tmp/testf` filter, the printcap, and execute the following commands:

```
printcap:
  lp:sd=/var/spool/lpd/%P
    :force_localhost
    :lp=/tmp/lp
    :filter=( /tmp/testf -F '${form}' )
    :form=

#!/bin/sh
# /tmp/testf - test filter for LPRng
PATH=/bin:/usr/bin; export PATH
echo TESTF $0 "$@" >&2
echo TESTF $0 "$@"
while test $# -gt 0 ; do
  echo "PARM '$1'";
  shift;
done
echo LEADER
/bin/cat
echo TRAILER
exit 0

h4: {183} % cp /dev/null /tmp/lp
h4: {184} % lpr /tmp/hi
h4: {185} % lpq -llll
Printer: lp@h4
 ...
 Status: IF filter msg - 'TESTF /tmp/testf -F' at 09:59:27.365

h4: {186} % more /tmp/lp
TESTF /tmp/testf -F
PARM '-F'
```

```
PARM ”
LEADER
hi
TRAILER
```

As you can see, there are *empty* parameters passed to the filter. This is due to the combination of the `$’{form}` and using the `:filter=(...)` form.

4.9.2. Filter Environment Variables

In this section we will look further at the environment variables passed to the filter. We printed the shell variable values for the filter at the start of the file:

```
h4: {187} % cat /tmp/lp
/tmp/testf -Plp -P lp -Xlp -Ylp /tmp/lp G:’ at 01:20:21.560
ENV
CONTROL=Hh4.private
 Ppapowell
 J/tmp/hi
 CA
 Lpapowell
 Apapowell@h4+105
 D2000-04-12-15:27:26.662
 Qlp
 N/tmp/hi
 fdfA105h4.private
 UdfA105h4.private
HF=H=h4.private
 P=papowell
 J=/tmp/hi
 C=A
 L=papowell
 A=papowell@h4+105
 D=2000-04-12-15:27:26.662
 Q=lp
 transfername=cfA105h4.private
 datafiles=N=/tmp/hi,transfername=fdfA105h4.private,
HOME=/home/daemon
LD_LIBRARY_PATH=/lib:/usr/lib:/usr/5lib:/usr/ucblib
LOGDIR=/home/daemon
LOGNAME=daemon
OPTIND=1
PATH=/bin:/usr/bin:/usr/local/bin
PRINTCAP_ENTRY=lp
 :force_localhost
 :filter=/tmp/testf
 :lp=/tmp/lp
 :sd=/tmp/LPD/lp
PRINTER=lp
```

```
PS1=$
PS2=>
SHELL=/bin/sh
SPOOL_DIR=/tmp/LPD/lp
USER=daemon

LEADER
hi
TRAILER
```

The HOME, USER, SHELL, PS1, and PS2 variables are usually set by the shell, and are reflect the information for the UID of the user running the shell.

The PATH and `LP_LIBRARY_PATH` are set by the **lpd** server to values specified in the printcap or configuration information. It is recommended that users set these to site specific values if the defaults are not suitable for their sites.

The **lpd** server sets the `PRINTER`, `PRINTCAP_ENTRY`, `CONTROL` and `HF` environment variables to the printer name, printcap entry, control file, and hold file for the print job. This information is very useful to filters that must make decisions based on values passed to the print server in the control file and which use parameters in the printcap entry to control their actions.

4.9.3. Using Command Line and Printcap Options In Filters

One of the problems commonly encountered problem in writing a filter is getting the command line values. The UNIX POSIX Standard provides a C Language `getopt` function that can be used for command line options, and some, but not all shell implementations have a corresponding shell `getopt` function. Also, many times it would be useful to get the values of the printcap options. These could be used to specify options or operations that are not easily done by passing command lines.

- Observe that all the command line options are single letters. If we set the shell variables to the corresponding option value, then we could access them by using `$x`, where `x` is the option letter. There is an exception to this rule, which is the `-c` command line literal, which for various historical and compatibility reasons does not take a value. But if it is present, we might as well assign it the value `1`.
- Observe that by convention all printcap options have lowercase names of two or more letters, and that all environment variables have all upper case letters. If we set shell variables with the corresponding printcap entry values, then we can access them using `$literal`. If we need to create a local shell variable for use, we can use `mIxEd` case and not have a conflict.

The `decode_args_with_sh` script which is in the UTILS directory of the **LPRng** distribution follows these conventions and sets the appropriate shell variables. We have also include a bit of code that will extract the control file control line values and put them into variables as well.

Save the current `/tmp/testf` filter file in `/tmp/testf.old` and replace it with the following:

```
#!/bin/sh
```

```
# this is an example of how to use /bin/sh and LPRng
# to get the command line and printcap option values
# and set shell variables from them
#  Note that we use a couple of variables
#PATH=/bin:/usr/bin
Args=""
vAr=""
vAlue=""
vAls=""
iI=""
Tf=""
Debug=1
if -n $Debug ; then
    set >/tmp/before
fi
Args="$@"
if -n $Debug ; then
    echo "$@" >>/tmp/before
fi
while expr "$1" : '-.*' >/dev/null ; do
  vAr=`expr "$1" : '-\(.\).*'`;
  vAlue=`expr "$1" : '-.\(.*\)`;
  case "$vAr" in
    - ) break;;
    c ) c=1;;
    [a-zA-Z] )
      if test "X$vAlue" = "X" ; then shift; vAlue=$1; fi;
      eval $vAr='$vAlue';
      #setvar $vAr "$vAlue"
      ;;
  esac;
  shift;
done

# set shell variables to the printcap options
#  flag   ->  flag=1
#  flag@  ->  flag=0
#  option=value ->  option='value'
#
setpcvals () {
    while test "$#" -gt 0 ; do
      iI=$1
      if expr "$iI" : " *\:" >/dev/null ; then
        vAr=`expr "$iI" : " *\:\([^=][^=]*\)=.*"`;
        vAlue=`expr "$iI" : " *\:[^=][^=]*=\(.*\)"`;
        if test "X$vAr" = "X" ; then
          vAr=`expr "$iI" : " *:\(.*\)@"`;
          vAlue=0;
        fi
        if test "X$vAr" = "X" ; then
          vAr=`expr "$iI" : " *:\(.*\)"`;
          vAlue=1;
        fi
```

```
        if test "X$vAr" != "X" ; then
          eval $vAr='$vAlue';
          #setvar $vAr "$vAlue"
        fi
      else
        vAr=`expr "$iI" : " *\([^|][^|]*\).*"`;
        if test "X$vAr" != "X" ; then
          eval Printer="$vAr"
        fi
      fi;
      shift
    done
}

# set shell variables to the printcap options
#  flag   ->  flag=1
#  flag@  ->  flag=0
#  option=value ->  option='value'
#
setcontrolvals () {
    while test "$#" -gt 0 ; do
      iI=$1
      vAr=`expr "$iI" : " *\([A-Z]\).*"`;
      vAlue=`expr "$iI" : " *[A-Z]\(.*\)"`;
      if test "X$vAr" != "X" ; then
        eval $vAr='$vAlue';
        #setvar $vAr "$vAlue";
      fi;
      shift
    done
}

Tf=$IFS
IFS="
"
setpcvals $PRINTCAP_ENTRY
setcontrolvals $CONTROL
IFS=$Tf

#
# restore argument list
set -- $Args
Args=""
vAr=""
vAlue=""
vAls=""
iI=""
Tf=""

if test -n "$Debug" ; then
    set >/tmp/after
    echo "$@" >>/tmp/after
    diff /tmp/before /tmp/after
```

```
fi
/bin/cat
exit 0
```

Lets print our `/tmp/hi` test file and then look at the results in `/tmp/lp`:

```
h4: {188} % cp /dev/null /tmp/lp
h4: {189} % lpr /tmp/hi
h4: {190} % more /tmp/lp
0a1
> e=dfA021771h4.private
2a4,6
> l=66
> s=status
> L=papowell
10a15,17
> j=021771
> C=A
> J=/tmp/hi
12a20
> a=acct
...
33a58
> Printer=lp
...
hi
```

As we see from the output, shell variables have the values of our command line and printcap options. It is left as an exercise for the reader to add the necessary `export` statements to cause these values to be exported to subshells. It is *not* recommended that a wholesale export of the shell variables be done, but only selected ones.

The paranoid and security minded reader will see some possible security problem with this script. The `eval $vAr='$vAlue'` command sets the value of the shell variable `$vAr` to the value `$vAlue`. The `$vAr` variable is always taken from either a single letter or is the name of an option in the printcap file. Clearly the printcap file must not be modifiable by users, and should have the same security considerations as any other system configuration file. The values of the `$vAlue` are taken directly from the control file, whose contents are under the control of the originator of the print job request.

For this reason **LPRng** takes the rather brutal step of *sanitizing* the control file. Only alphanumerics or a character in the list `@/:()=,+-%_` are used in the control file; all others replaced by the underscore (_) character. In addition, all filters are run as the **lpd** user specified in the `lpd.conf` configuration file.

The following is an example of how to extract the same information in Perl:

```
#!/usr/bin/perl
eval 'exec /usr/bin/perl -S $0 ${1+"$@"}'
    if $running_under_some_shell;
            # this emulates #! processing on NIH machines.
            # (remove #! line above if indigestible)
```

```
use Getopt::Std;
my(%args,%options);
# get the arguments
getopt(
    "a:b:cd:e:f:g:h:i:j:l:m:n:o:p:q:r:s:t:u:v:w:x:y:z:" .
    "A:B:C:D:E:F:G:H:I:J:L:M:N:O:P:Q:R:S:T:U:V:W:X:Y:Z:",
    \%args );

# set :key=value  -> $option{$key}=$value
# set :key@       -> $option{$key}="0"
# set :key        -> $option{$key}="1"
 map {
    if( m/^\s*:([^=]+)=(.*)/ ){
     $options{$1}=$2;
    } elsif( m/^\s*:([^=]+)\@$/ ){
     $options{$1}="0";
    } elsif( m/^\s*:([^=]+)/ ){
     $options{$1}="1";
    } elsif( m/^\s*([^|]+)/ ){
     $options{"Printer"}=$1;
    }
 } split( "\n", $ENV{'PRINTCAP_ENTRY'});

# get the control file entries
 map {
    if( m/^\s*([A-Z])(.*)/ ){
     $options{$1}=$2;
    } elsif( m/^\s*([a-z])/ ){
     $options{'Format'}=$1;
    }
 } split( "\n", $ENV{'CONTROL'});
```

The Perl `Getopt::Std` routine parses the command line options and puts their values in the `%args` hash variable where they can be accessed using `$args{'x'}`. Similarly, the `map` and `split` functions process the PRINTCAP_ENTRY and CONTROL environment variable and set `%options` with the printcap entry options and the values from the control file. The `map` function could be replaced by a `foreach` loop, but this is Perl: *There is more than one way to do it* and no tutorial would be complete without at least one mind stretching example that has the reader reaching for the reference manual.

4.9.4. Filter Exit Codes

The **lpd** server uses the exit code of the filter to determine if the filter was successful or unsuccessful. The Filter Exit Codes section discusses these values in detail, but here are the most important:

0 - JSUCC

A JSUCC exit code indicates that the filter was successful in doing its work.

1 - JFAIL

A JFAIL exit code indicates that the filter was unsuccessful in doing its work, possibly due to a transient condition such as out of paper, printer jam, etc., but an additional attempt might be successful. Usually the **lpd** server will try at most send_try attempts before giving up.

2 - JABORT

A JABORT exit code indicates that the filter was unsuccessful in doing its work and has detected a condition that would make it impossible to print the job. In addition, the printer may require administrative attention, and the print queue operation may need to be suspended until the problem is rectified.

3 - JREMOVE

The JREMOVE exit code will cause the job to be removed from the print queue.

6 - JHOLD

The JHOLD exit code will cause the job to be temporarily prevented from printing until release by the `lpc release` command.

Other Values

Usually any other value, including exit due to a signal, is treated as a JABORT exit, and the same action is taken.

It should be obvious that the filter exit code is very important, and that care needs to be taken to return the correct value.

4.9.5. Job Formats and Filter Selection

In the previous sections we discussed how a print filter was executed and how it could be used. Now we will look at how the **lpd** spooler chooses a print filter program. Let us re-examine our example print job control file:

```
Hh4.private
Ppapowell
J/tmp/hi
CA
Lpapowell
Apapowell@h4+105
D2000-04-12-15:27:26.662
Qlp
N/tmp/hi
fdfA105h4.private
UdfA105h4.private
```

Each data file for a print job has a name with the format `df`*X*`nnn`h4`.private`. The `df` is used to indicate that the file is a *data file*, and the remainder is a unique name for the file in the job. The `X` part of the name must be an upper or lower case letter, setting a limit of 52 different files in a single print job.

The `fdfA105h4.private` line in the control file specifies that we are to print the job using the filter for the `f` format; the file printing information consists of the format assigned to the file and the name of the file. In the legacy BSD print spoolers, this format was used to select the print filter to be used. **LPRng** has expanded this by providing a *default* filter specification.

Table 4-2. Job Formats and Filter Selection

lpr command line option	Control File Line	Printcap Option For Filter	Filter Comma Line
(default)	fdfAnnn	:if=/path	/path -Ff ...
-b or -l	ldfAnnn	:if=/path	/path ... -Ff -c
-p	pdfAnnn	:if=/path	pr | format f filte
-c, -d, -n, -r, -t, -v, -FX	XdfAnnn	:Xf=/path ...	/path -FX
any format	XdfAnnn	:filter=/path ...	/path -FX

Table 4-2 shows the rather baroque relationship between the format options specified by the **lpr** command and the way that **lpd** uses them. The reason for this complexity lies in the various implementations and variations that occurred during the development and deployment of the original BSD print spooling software.

Here is the complete, arcane, and baroque set of rules that are used to select and print filters. The default format used by **lpr** is `f`; unless some other format is specified this is used. The `lpr -b` and `lpr -l` (`b`inary and `l`iteral literals) are a request to **lpd** to do as little processing as possible of this file before printing it. **Lpd** use the `:if` filter for formats `f` and `l`; the `l` literal causes the the `-c` filter command line flag to be used as well. The `lpr -c`, `-d`, `-n`, `-r`, `-t`, and `-v` options cause the corresponding format to be used, and for **lpd** to use the filter specified by the printcap option `:Xf`, where `X` is the specified format.

The `lpr -p` (`p`retty-print literal) selects `p` format, and **lpd** is supposed to use the program specified by the `:pr` printcap option to format the file and then process the output of this program according to format `f`. Unpredictable results may occur using this facility.

the `lpr -FX` allows you to explicitly specify format `X` where `X` is a lower case letter, and **lpd** will use the filter specified by printcap option `:Xf`, where `X` is the specified format. If there is no `:Xf` printcap literal value then the printcap `:filter` literal value will be used as the filter, and if this is undefined then the file will be passed without processing through to the printing device.

If a filter is not specified for the format then the default filter specified by `:filter=/path` filter is used, and if there is no default, then the output is sent directly to the output device.

If the `:fx=`*formats* is present in a printcap entry, it specifies the formats that are allowed. For example, `:fx=lfv` would allow only formats `l`, `f`, and `v` to be used on a particular spool queue.

Some `Xf` options have pre-assigned meanings and cannot be used for filter selection.

Printcap Option	Purpose
Printcap Option	Purpose
:af=/path	Accounting File

Printcap Option	Purpose
:ff=formfeed	Form Feed String
:if=/path	filter (l,b,p,f formats)
:filter=/path	Default filter
:lf=/path	Log file
:of=/path	OF filter
:sf	suppress form feed between job files

The `:of` filter is a special case and is used for banner printing and accounting purposes. See OF Filter for details.

4.10. Job File Format Conversion with Filters

One of the major problems that face new users to UNIX printing is when they have a printer that has a proprietary print job format such as the HP DeskJet series of printers. The solution to this problem is quite simple: generate your output in PostScript, and then use the GhostScript program to convert the GhostScript output to a format compatible with your printer.

```
lp:filter=/usr/local/lib/filters/myfilter:...

/tmp/myfilter:

#!/bin/sh
/usr/local/bin/gs -dSAFER -dNOPAUSE -q -sDEVICE=djet500 \
    -sOutputFile=- - && exit 0
exit 2
```

This simple *tutorial* example suffers from some serious problems. If you accidentally send a non-PostScript file to the printer GhostScript will detect this and exit with an error message but only after trying to interpret the input file as PostScript. If the input file was a text file, this can result in literally thousands of error messages and hundreds of pages of useless output.

In order to make a more robust filter we need to meet the following minimum requirements:

1. The file type should be determined, and only files that are PostScript should be passed to GhostScript.
2. We may have some conversion routines that can convert files into PostScript files and then we can send them to GhostScript for raster conversion.
3. If we cannot convert a file, then we should simply terminate the printing and cause the spooler to remove the job.

The **ifhp** Print Filter program is a companion to the **LPRng** software and does this type of operation. If you are using Linux, then you may find the *RedHat Print Filters* (http://www.debian.org) installed and in use on your system. The *magicfilter* developed by H. Peter Anvin http://www.debian.org is distributed with Debian Linux. The **apsfilter** by Andreas Klemm http://www.freebsd.org/~andreas/index.html is also widely used, although now most of its functionality is directly available in **LPRng**. Finally, the **a2ps** (Ascii to PostScript) converter by Akim Demaille (http://demaille@inf.enst.fr) and Miguel Santana (mailto:santana@st.com) is available from www-inf.enst.fr/~demaille/a2ps (http://www-inf.enst.fr/~demaille/a2ps). This package provides a very nice set of facilities for massaging, mangling, bending, twisting, and being downright nasty with text or other files.

4.10.1. Simple Filter with File Format Detection

Since this is a tutorial, we will demonstrate a simple way to make your own *multi-format* print filter, and provide insight into how more complex filters work.

The file utility developed by Ian F. Darwin uses a database of file signatures to determine what the contents of a file are. For example:

```
h4: {191} % cd /tmp
h4: {192} % echo hi >hi
h4: {193} % gzip -c hi >hi.gz
h4: {194} % echo "%!PS-Adobe-3.0" >test.ps
h4: {195} % gzip -c test.ps >test.ps.gz
h4: {196} % file hi hi.gz test.ps test.ps.gz
hi:         ASCII text
hi.gz:      gzip compressed data, deflated
test.ps:    PostScript document text conforming at level 3.0
test.ps.gz: gzip compressed data, deflated
h4: {197} % file - <test.ps
standard input: PostScript document text conforming at level 3.0
```

If we are given a file, we can now use **file** to recognize the file type and if the file type is suitable for our printer we can send it to the printer, otherwise we can reject it. The following is a simple yet very powerful shell script that does this.

```
#!/bin/sh
# set up converters
gs="/usr/local/bin/gs -dSAFER -dNOPAUSE -q -sDEVICE=djet500 \
    -sOutputFile=/dev/fd/3 - 3>&1 1>&2"
a2ps="/usr/local/bin/a2ps -q -B -1 -M Letter --borders=no -o-"
decompress=""
# get the file type
type=`file - | tr A-Z a-z | sed -e 's/  */_/g'`;
echo TYPE $type >&2
case "$type" in
  *gzip_compressed* ) decompress="gunzip -c |" compressed="compressed" ;;
esac

# we need to rewind the file
```

```
perl -e "seek STDIN, 0, 0;"

if test "X$decompress" != "X" ; then
    type=`$decompress head | file - | tr A-Z a-z | sed -e 's/  */_/g'`;
    echo COMPRESSED TYPE $type >&2
    # we need to rewind the file
    perl -e "seek STDIN, 0, 0;"
fi
case "$type" in
  *postscript* ) process="$gs" ;;
  *text* )       process="$a2ps | $gs" ;;
  * )
    echo "Cannot print type $compressed '$type'" >&2
    # exit with JREMOVE status
    exit 3
    ;;
esac
# in real life, replace 'echo' with 'exec'
echo "$decompress $process"
# exit with JABORT if this fails
exit 2
```

Copy this to the `/tmp/majik` file, and give it `0755` (executable) permissions. Here is an example of the output of the script:

```
h4: {198} % /tmp/majik <test.ps.gz
TYPE standard_input:_gzip_compressed_data,_deflated...
COMPRESSED TYPE standard_input:_postscript_document_level_3.0
gunzip -c | /usr/local/bin/gs -dSAFER -dNOPAUSE -q -sDEVICE=djet500 \
   -sOutputFile=/dev/fd/3 - 3>&1 1>&2
h4: {199} % /tmp/majik </tmp/hi
TYPE standard_input:_ascii_text
 /usr/local/bin/a2ps -q -B -1 -M Letter --borders=no -o- \
  | /usr/local/bin/gs -dSAFER -dNOPAUSE -q -sDEVICE=djet500 \
   -sOutputFile=/dev/fd/3 - 3>&1 1>&2
```

The first part of the script sets up a standard set of commands that we will use in the various conversions. A full blown package for conversion would use a database or setup file to get these values. We then use the **file** utility to determine the input file type. The output of the **file** utility is translated to lower case and multiple blanks and tabs are removed.

We use a simple shell `case` statement to determine if we have a compressed file and get a decompression program to use. We reapply the **file** utility to the decompressed file (if it was compressed) and get the file type.

Finally we use another `case` statement to get the output converter and then we run the command. For tutorial purposes, we use an `echo` rather than an `exec` so we can see the actual command, rather than the output.

Just for completeness, here is **majikperl**:

```
#!/usr/bin/perl
eval 'exec /usr/bin/perl -S $0 ${1+"$@"}'
    if $running_under_some_shell;
            # this emulates #! processing on NIH machines.
            # (remove #! line above if indigestible)
my($gs) = "/usr/local/bin/gs -dSAFER -dNOPAUSE -q -sDEVICE=djet500 \
    -sOutputFile=/dev/fd/3 - 3>&1 1>&2";
my($a2ps)="/usr/local/bin/a2ps -q -B -1 -M Letter --borders=no -o-";

my($decompress,$compressed,$process,$type);
$decompress=$compressed=$process=$type="";

# get the file type
$type = ` file - `;
$type =~ tr /A-Z/a-z/;
$type =~ s/\s+/_/g;
print STDERR "TYPE $type\n";
($decompress,$compressed) = ("gunzip -c |", "gzipped")
  if( $type =~ /gzip_compressed/ );
print STDERR "decompress $decompress\n";
unless( seek STDIN, 0, 0 ){
    print "seek STDIN failed - $!\n"; exit 2; }
if( $decompress ne "" ){
    $type = ` $decompress file - `;
    $type =~ tr /A-Z/a-z/;
    $type =~ s/\s+/_/g;
    print STDERR "COMPRESSED TYPE $type\n";
    unless( seek STDIN, 0, 0 ){
        print "seek STDIN failed - $!\n"; exit 2; }
}
$_ = $type;
if( /postscript/ ){
    $process="$gs";
} elsif( /_text_/ ){
    $process="$a2ps | $gs" ;;
} else {
    print STDERR "Cannot print $compressed '$type'" >&2;
    # JREMOVE
    exit 3;
}
exec "$decompress $process";
print "exec failed - $!\n";
exit 2;
```

4.10.2. The ifhp Filter

The ifhp Print Filter is the companion print filter supplied with **LPRng** and is normally installed together with the **LPRng** software. **Ifhp** supports a wide range of PostScript, PCL, text, and raster printers, and can be configured to support almost any type of printer with a stream based interface. It provides

diagnostic and error information as well as accounting information. It recognizes a wide range of file types by using the **file** utility and the pattern matching technique demonstrated in the previous section, and can do selective conversions from one format to others.

The PostScript and PCL printer job languages are supported by most printer manufacturers. However, in order to have a job printed correctly the following steps must be taken.

1. The printer must be put into a known *initial* state by sending it the appropriate reset strings or performing a correct set of IO operations.
2. If accounting is being done, then the printer accounting information must be obtained and recorded. See Accounting for more information about **LPRng** support for accounting.
3. The file to be printed must be checked to see if it is compatible with the printer, and if not, a format conversion program invoked to convert it to the required format.
4. If the user selects a set of printer specific options such as landscape mode, duplex printing, multiple copies, or special paper, the appropriate commands must be sent to the printer to select these options.
5. The file must be transferred to the printer and the printer is monitored for any error conditions.
6. Any required end of job commands are sent to the printer, and the printer monitored for error conditions while the job finishes printing.
7. If accounting is being done, the printer accounting information such as page count and time used must be obtained and recorded. See Accounting for more information about **LPRng** support for accounting.

The **ifhp** filter uses the `ifhp.conf` configuration file to determine the actions and commands appropriate for various models of printers. See the **ifhp** documentation for details about the format and contents of this file. This file contains entries for a large number of PostScript, PJL, and other printers. The default printer used by **ifhp** is the HP LaserJet 4M Plus which supports PostScript, PCL, and PJL. The commands and formats used by this printer is compatible with a large number of other HP printers.

We will demonstrate how to add the **ifhp** filter to your printcap entry. Find the path to the **ifhp** filter using the **find** command as we did in the previous exercise. Modify the printcap as shown below and use `lpc lpd` to restart **lpd**.

```
lp:sd=/var/spool/lpd/%P
  :force_localhost
  :lp=/tmp/lp
  :ifhp=model=default
  # modify the path to ifhp appropriately
  :filter=/usr/local/libexec/filters/ifhp
```

Now print the `/tmp/hi` and then display `/tmp/lp` using a text editor such as **vi** or **emacs** that shows control characters:

```
h4: {200} % cp /dev/null /tmp/lp
h4: {201} % lpr /tmp/hi
h4: {202} % vi /tmp/lp
```

```
^[%-12345X@PJL
@PJL JOB NAME = "PID 405" DISPLAY = "papowell"
@PJL RDYMSG DISPLAY = "papowell"
@PJL USTATUSOFF
@PJL USTATUS JOB = ON
@PJL USTATUS DEVICE = ON
@PJL USTATUS PAGE = ON
@PJL USTATUS TIMED = 10
@PJL ENTER LANGUAGE = PCL
^]E^]&^]&k2G^]&s0C^]&l0O^]9^](s0P^](s10.00H^](s4099Thi
^]E^]%-12345X@PJL
@PJL RDYMSG DISPLAY = "papowell"
@PJL EOJ NAME = "PID 405"
@PJL USTATUSOFF
@PJL USTATUS JOB = ON
@PJL USTATUS DEVICE = ON
@PJL USTATUS PAGE = ON
@PJL USTATUS TIMED = 10
@PJL RDYMSG DISPLAY = ""
^[%-12345X
```

The output now contains all of the control sequences and setup codes needed to print a text file on the default printer. The `:ifhp=model=default` printcap entry is used by **ifhp** to get the information it needs to perform its operation. The following options are commonly provided in the `:ifhp=` option to configure the **ifhp** filter.

Table 4-3. :ifhp= Options

Option	Purpose
`model=`*name*	Use *name* entry in `ifhp.conf`
`status` or `status@`	Printer does or does not provide status information
`sync`, `sync@`, `sync=`*(ps|pjl)*	Printer does or does not indicate ready to operate at start of job, or use PostScript or PJL code sequence to determine if printer is ready.
`pagecount`, `pagecount@`, `pagecount=`*(ps|pjl)*	Printer does or does not have pagecount support, or use PostScript or PJL code sequence to determine pagecount.
`waitend`, `waitend@`, `waitend=`*(ps|pjl)*	Wait or do not wait for end of job, or send PostScript or PJL code sequence to have printer report end of job.

The `model=`*name* entry is used to specify the configuration entry in the `ifhp.conf` file to be used by **ifhp**. This entry usually has all of the specific information needed by the **ifhp** filter.

The `status` option is the most common option usually provided in a printcap entry. This option is needed when the communication with the printer is *write-only* and no status information will be returned. If a printer normally supports returning status information then the `ifhp.conf` configuration entry will indicate this and the **ifhp** filter will try to get status. When no status is returned it will either

terminate operation after a timeout or sit in an endless loop waiting for status. By specifying `status@` you will suppress getting status. This also has the effect of doing `sync@`, `pagecount@`, and `waitend@`

The `sync` option is used to cause **ifhp** to wait for an *end of job* indication from the printer before starting the next job. This is usually done in order to make sure that all jobs have been flushed from a printer before starting another job. If you specify `sync@` then you may get slightly faster startup but at the expense of losing the ends of previous print jobs.

The `pagecount` option is used to cause **ifhp** to get the value of a hardware pagecounter from the printer. If your printer supports such an item then the `ifhp.conf` configuration option usually indicates this. However, it takes a small amount of time to get the pagecounter information from the printer and you may not need it. Use `sync@` if you do not want page counts.

Finally, `waitend` option is used to cause **ifhp** to wait for an *end of job* indication from the printer before exiting. If you specify `waitend@` then the filter will exit immediately after sending the job, but you will possibly lose any error information or status reports from the printer.

For a complete list of all of the **ifhp** options please see the IFHP documentation.

4.10.3. The Jaggies - LF to CR-LF Conversion With lpf

When printing to vintage hard copy devices or to printers that support a *text* mode, many UNIX users discover that their output suffers from a case of the jaggies.

```
Input file:

  This is
  a nice day

Output:

  This is
         a nice day
```

UNIX systems terminate lines with a single NL (new line) character. This causes the printer to move down one line on the printing page but does not change its horizontal position and print the next character at the left margin. This is done by using the CR (carriage return) character. You need to convert the single NL to a `CR-LF` combination and the **lpf** filter supplied with **LPRng** does this.

First, locate the **lpf** filter. You can find it by using the command:

```
h9: {160} % find /usr/ -type f -name lpf -print
/usr/libexec/lpr/lpf
```

We will first see what the output is like without **lpf**, and then see what it does. Modify the **lp** printcap entry as shown below and then use `lpc restart` to restart the **lpd** server.

```
lp:sd=/var/spool/lpd/%P
```

```
  :force_localhost
  :lp=/tmp/lp
```

Print a file and view the output using the following commands. If you do not have the **od** (octal dump) program, try using **hexdump** or some other appropriate program that displays the numerical contents of the file.

```
h4: {203} % cp /dev/null /tmp/lp
h4: {204} % lpr /tmp/hi
h4: {205} % od -bc /tmp/lp
0000000  150 151 012
           h   i  \n
0000003
```

Now we will use the **lpf** filter. Modify the printcap as shown below and use `lpc reread` to cause **lpd** to reread the configuration information.

```
lp:sd=/var/spool/lpd/%P
  :force_localhost
  :lp=/tmp/lp
  # modify the path to lpf appropriately
  :filter=/usr/local/libexec/filters/lpf
```

Now reprint the file:

```
h4: {206} % cp /dev/null /tmp/lp
h4: {207} % lpr /tmp/hi
h4: {208} % od -bc /tmp/lp
od -bc /tmp/lp
0000000  150 151 015 012
           h   i  \r  \n
0000004
```

As you see, **lpf** changes the `LF` to a `CR-LF` sequence.

4.10.4. Store and Forward Spool Queues

Up to now we have assumed that associated with each spool queue is a hardware printing device. When a job is sent to the spool queue the **lpd** server will take actions to filter it and then send it to the printing device.

However, we can also have *store and forward* spool queues. These queue act to simply buffer jobs and then forward them to another spooler. The following printcap entry shows how you can specify a store and forward queue.

```
# store and forward using classical BSD :rm:rp
lp:rp=pr:rm=host
  :sd=/var/spool/lpd/%P
  :server
# store and forward using LPRng lp=pr@host
lp:lp=pr@host
  :sd=/var/spool/lpd/%P
  :server
```

The legacy `:rp` (remote printer) and `:rm` (remote host) format can be used to specify the print queue and destination host for jobs sent to this queue. The **LPRng** `:lp=pr@host` format serves the same function, and has precedence over the `:rm:rp` form.

Edit the printcap file so it has contents indicated below, use `checkpc -f` to check the printcap, and then use `lpc reread` to restart the **lpd** server.

```
lp:force_localhost
lp:server
  :sd=/var/spool/lpd/%P
  :lp=lp2@localhost
lp2:force_localhost
lp2:server
  :sd=/var/spool/lpd/%P
  :lp=/tmp/lp2
```

Execute the following commands to print the `/tmp/hi` file and observe the results:

```
h4: {209} % lpr /tmp/hi
h4: {210} % lpq -lll
Printer: lp@h4 (dest lp2@localhost)
 Queue: no printable jobs in queue
 Status: sending control file 'cfA029h4.private' \
    to lp2@localhost at 09:39:57.719
 Status: completed sending 'cfA029h4.private' \
    to lp2@localhost at 09:39:57.724
 Status: sending data file 'dfA029h4.private' \
    to lp2@localhost at 09:39:57.727
 Status: completed sending 'dfA029h4.private' \
    to lp2@localhost at 09:39:57.925
 Status: done job 'papowell@h4+29' transfer \
    to lp2@localhost at 09:39:57.926
 Status: subserver pid 29031 exit status 'JSUCC' at 09:39:57.953
 Status: lp@h4.private: job 'papowell@h4+29' printed at 09:39:57.961
 Status: job 'papowell@h4+29' removed at 09:39:57.993
Printer: lp2@h4
 Queue: no printable jobs in queue
 Status: no banner at 09:39:58.054
 Status: printing data file 'dfA029h4.private', size 3 at 09:39:58.054
 Status: printing done 'papowell@h4+29' at 09:39:58.054
 Status: accounting at end at 09:39:58.054
 Status: finished 'papowell@h4+29', status 'JSUCC' at 09:39:58.054
```

```
 Status: subserver pid 29033 exit status 'JSUCC' at 09:39:58.056
 Status: lp2@h4.private: job 'papowell@h4+29' printed at 09:39:58.056
 Status: job 'papowell@h4+29' removed at 09:39:58.069
```

As we see from the status, our job was sent to the `lp` spool queue first. It was store there and then the **lpd** server transferred it to the `lp2` spool queue, where it was printed to the file `/tmp/lp2`.

4.10.5. Filtering Job Files In Transit

One of the major problems with store and forward operation is that the destination spool queue may not actually be a spool queue - it can be a printer. Many network printers provide an RFC1179 compatible network interface and act, for job forwarding purposes, like a host running a limited capability BSD print spooler.

By adding a filter to the printcap information we can modify the format of a job file so that it is compatible with the destination printer.

Edit the printcap and `/tmp/testf` files so they have the contents indicated below, give `/tmp/testf` executable permissions, use `checkpc -f` to check the printcap, and then use `lpc reread` to restart the **lpd** server.

```
# set /tmp/testf to contain the following
#   and chmod 755 /tmp/testf
#!/bin/sh
echo TESTF $0 $@
/bin/cat
exit 0

# printcap
lp:force_localhost
lp:server
  :sd=/var/spool/lpd/%P
  :lp=lp2@localhost
  :filter=/tmp/testf
  :bq_format=ffl
lp2:force_localhost
lp2:server
  :sd=/var/spool/lpd/%P
  :lp=/tmp/lp2
```

Execute the following commands to print the `/tmp/hi` file and observe the results:

```
h4: {211} % lpr /tmp/hi
h4: {212} % lpq -llll
h4: {213} % lpq -llll
Printer: lp@h4 (dest lp2@localhost)
 Queue: no printable jobs in queue
 Status: no banner at 09:55:53.681
 Status: printing data file 'dfA086h4.private', size 3, \
```

```
    IF filter 'testf' at 09:55:53.683
 Status: IF filter finished at 09:55:53.713
 Status: printing done 'papowell@h4+86' at 09:55:53.714
 Status: sending job 'papowell@h4+86' to lp2@localhost at 09:55:53.734
 Status: connecting to 'localhost', attempt 1 at 09:55:53.735
 Status: connected to 'localhost' at 09:55:53.739
 Status: requesting printer lp2@localhost at 09:55:53.740
 Status: sending control file 'cfA086h4.private'
      to lp2@localhost at 09:55:53.752
 Status: completed sending 'cfA086h4.private'
      to lp2@localhost at 09:55:53.757
 Status: sending data file 'dfA086h4.private'
      to lp2@localhost at 09:55:53.758
 Status: completed sending 'dfA086h4.private'
      to lp2@localhost at 09:55:53.939
 Status: done job 'papowell@h4+86' transfer
      to lp2@localhost at 09:55:53.940
 Status: subserver pid 29088 exit status 'JSUCC' at 09:55:53.980
 Status: lp@h4.private: job 'papowell@h4+86' printed at 09:55:53.983
 Status: job 'papowell@h4+86' removed at 09:55:53.998
Printer: lp2@h4
 Queue: no printable jobs in queue
 Status: subserver pid 29092 starting at 09:55:54.005
 Status: accounting at start at 09:55:54.005
 Status: opening device '/tmp/lp2' at 09:55:54.005
 Status: printing job 'papowell@h4+86' at 09:55:54.005
 Status: no banner at 09:55:54.006
 Status: printing data file 'dfA086h4.private', size 298 at 09:55:54.006
 Status: printing done 'papowell@h4+86' at 09:55:54.006
 Status: accounting at end at 09:55:54.006
 Status: finished 'papowell@h4+86', status 'JSUCC' at 09:55:54.006
 Status: subserver pid 29092 exit status 'JSUCC' at 09:55:54.008
 Status: lp2@h4.private: job 'papowell@h4+86' printed at 09:55:54.008
 Status: job 'papowell@h4+86' removed at 09:55:54.020
```

We have displayed a bit more status information so that we can see what the actions the `lp` queue carries out. It first *processes* the job data file using the **testf** filter and puts the results in a temporary file. Then it sends the contents of the temporary file to the `lp2` queue. The `lp2` queue receives the converted job file and then prints it to the `/tmp/lp2` file in turn.

By default, each file in a job is processed by a print file and the processed output is then sent to the destintion as individual job files, each with the format specified by the value of the `bq_format` (default `f`) option. The `bq_format` option has the format `iOiO...d`; each `i` is the original format and the corresponding `O` is the output format. If there is an odd number of characters then the last unmatched character is used as the default format, otherwise no translation is done. For example, `flrfl` will cause the `f` format to be mapped to `l`, `r` to `f`, and any others to `l`.

4.11. Printcap Basics

In the previous sections we have used simple printcap entries to define how to set up filters and pass parameters to them. We will now examine the printcap database in more detail.

The **LPRng** server and client software gets their configuration information from:

- Compile time settings which set the default values for the configuration information.
- A `lpd.conf` file that contains values that override the compile time defaults. This information can effect the behavior of the **lpd** server and clients.
- Printcap entries which have configuration information for individual print queues. The information in the printcap entries for the queue override the `lpd.conf` and compile time defaults. The system printcap file is read first, followed by the user printcap file.
- Command line and environment variable values. These can be used to override or select particular configuration information or to select one of a set of options for use.

Each print queue or printer has a name which is used to look up the printcap information for the printer. The `/etc/printcap` file is the default location for the printcap information, although it can also be obtained from database servers, or generated by programs. See the
Using Programs To Get Printcap Information section for details.

We will use a more complex printcap file to explore how **LPRng** gets the printcap information. Put the following lines in the `/etc/printcap` file:

```
# client entry
lp:tc=.client
lp2:tc=.client
.client:
  :lp=%P@localhost
  :force_localhost

lp:server
  :cm=The Main Print Queue
  :lp=/tmp/lp
  :tc=.common

lp2:server
  :cm=The Second Print Queue
  :lp=/tmp/lp2
  :tc=.common

.common:
  :sd=/var/spool/lpd/%P
  :mx=0
```

The `lpc client` command is very useful to see how **LPRng** uses this printcap information:

```
h4: {214} % lpc client
Config
```

```
h4: {215} % lpc client all
Config
 :lpd_port=2000
 :printcap_path=/var/tmp/LPD/printcap

Names
 :.client=.client
 :.common=.common
 :lp=lp
 :lp2=lp2
 :main=lp

All
 :lp
 :lp2

Printcap Information
lp|main
 :force_localhost
 :lp=lp@localhost
lp2
 :force_localhost
 :lp=lp2@localhost
```

The `lpc client all` command shows all of the configuration and printcap information, and is the handiest one for system debugging and diagnostics. The *Name* information is the names of the printcap entries that have been found in the database and is listed in sorted order. The *All* are entries that correspond to actual queues or printers and are listed in the order that they appear in the printcap or according to an order specified by the system administrator. (See the `all` Printcap Entry for details.)

4.11.1. Printcap Processing Format

Queue or printer names must start with an alphanumeric character, and contain only alphanumerics, hyphens (-) and underscores (_). To avoid known and nasty problems with sending and receiving print jobs from case sensitive and case insensitive systems, **LPRng** brutally lowercases all printcap entry names and printer names.

The printcap file is processed by reading it line by line and composing the individual printcap entries. Each entry has an name and one or more aliases. The entries in the printcap assign values to options. These can have the format:

```
option=string value \n with escapes
flag         # equivalent to flag=1
flag@        # equivalent to flag=0
option#value # equivalent to option=value
```

An option will have the last value that occurs in the printcap entry.

Our example shows the use of the `tc` (*t*ermcap in*c*lude) facility. The `:tc` value is a list of printcap entries that should be prefixed to the *start* of the printcap entry in which it appears. This allows options to be set in the printcap entry which will override the values in the `:tc` included entry. For convenience, the options are displayed in sorted order.

The **LPRng** clients and **lpd** server may require a different set of options for the same spool queue. The clients require options whose values tell the clients how the contact the **lpd** server and transfer a print job or query to it. The **lpd** server needs options that tell it how to either print a job or forward it to another **lpd** server. The `:client` or `:server` option marks a printcap entry as for client or **lpd** server use only; unmarked entries are used by both server and client. The `lpc client` command shows the printcap information that the **LPRng** clients would use. For example, here is what the **lpd** server would use:

```
h4: {216} % lpc server all
Config
 :lpd_port=2000
 :printcap_path=/var/tmp/LPD/printcap

Names
 :.client=.client
 :.common=.common
 :lp=lp
 :lp2=lp2
 :main=lp

All
 :lp
 :lp2

Printcap Information
lp|main
 :cm=The Main Print Queue
 :force_localhost
 :lp=/tmp/lp
 :mx=0
 :sd=/var/spool/lpd/%P
 :server
lp2
 :cm=The Second Print Queue
 :force_localhost
 :lp=/tmp/lp2
 :mx=0
 :sd=/var/spool/lpd/%P
 :server
```

When we select the `server` printcap information, we see that the `:sd` option has been added, and the `:lp` replaced with new values.

We can use the `:oh` (`o`n this `h`ost) option to mark printcap entries for use by a selected set of hosts. For example:

```
lp:oh=10.0.0.0/255.255.255.0,*.private,!10.0.0.10
```

```
  :lp=%P@10.0.0.10
```

The `:oh` option takes a list of IP addresses and masks or glob patterns, and applies these to the IP addresses or list of Fully Qualified Domain Names for the current host. If there is a for at least one IP address or pattern in the list match then the entry is used. An exclamation mark (`!`) inverts the sense of the match, and the entry is used if the match fails.

Finally, we can use the `wildcard` facility to cause a default printcap entry to be used:

```
lp|*:cm=Wildcard Alias - %P=lp, %Q=wanted
  :lp=%P@10.0.0.10
*:cm=Wildcard Name- %P=wanted, %Q=wanted
  :lp=%P@10.0.0.10
```

The **LPRng** software first searches the printcap information for an exact match. If none is found then it searches for the first wildcard entry that matches the printer name. If the wildcard is used as an alias, then the printcap entry is simply selected for use, with the printer name and queue name selected as shown above. We can also use partial matching as well:

```
lp|lp_*
  :lp=%P@10.0.0.10
qt|qt_*
  :lp=%P@10.0.0.12
```

In the example above the first entry matches `lp` and all printer names starting with `lp_`, while the second entry matches `qt` and all printer names starting with `qt_`. This can be useful when setting up a family of spool queues as discussed in later sections.

4.11.2. Printcap Information From Programs and Databases

There many administrators store system information on a database server and having programs or utilities get their configuration information from this server. The use of the database allows easier system administration and also centralizes the administration. Rather than build in the various types of database access, the LPRng software allows the use of programs to obtain printcap information. This not only allows any type of database to be used, but also removes any legal or license restrictions on the redistribution of the actual software.

We will use very simple example to show how you can use a program to get printcap information. First, you must configure the **LPRng** software to use a program to get the filter information. This is done by setting a value in the `lpc.conf` file (usually `/etc/lpd.conf` or `/usr/local/etc/lpd.conf`). Copy your `lpd.conf` file to `lpd.conf.bak` and then add the following line to the end of the file:

```
printcap_path=|/tmp/getpc
```

```
h4: {217} % cd /etc
```

```
h4: {218} % cp lpd.conf lpd.conf.bak
h4: {219} % echo 'printcap_path=|/tmp/getpc' >>lpd.conf
```

Next, edit the `/tmp/getpc` file and set its values as shown below.

```
set /tmp/getpc:

  #!/bin/sh
  # /tmp/getpc
  echo PROG $0 "$@" >>/tmp/trace
  cat >>/tmp/trace
  cat <<EOF
  lp:lp=test@host
  EOF
  exit 0

h4: {220} % chmod 755 /tmp/getpc
h4: {221} % echo testing | /tmp/getpc -aoption
lp:lp=test@host
h4: {222} % cat /tmp/trace
PROG /tmp/getpc -aoption
testing
```

After you have tested the **getpc** script, use the `lpc client all` command:

```
h4: {223} % lpc client all
Config
 :lockfile=/var/tmp/LPD/lpd
 :lpd_port=2000
 :printcap_path=|/tmp/getpc

Names
 :lp=lp

All
 :lp

Printcap Information
lp
 :lp=test@host
h4: {224} % cat /tmp/trace
h4: {225} % cat /tmp/trace
PROG /tmp/getpc -Pall -aacct -l66 -sstatus \
  -t2000-05-05-08:40:51.000 -w80 -x0 -y0 acct
all
PROG /tmp/getpc -Pall -aacct -l66 -sstatus \
  -t2000-05-05-08:40:51.000 -w80 -x0 -y0 acct
*
```

As seen from the `/tmp/trace` file, the `getpc` program is invoked with the standard filter parameters. The `-P` command line literal is set to the name of the printcap entry and the name of the entry is written to the filter's STDIN. If the entry is not found, then the wildcard printcap entry will be requested. The `-P` literal is *not* set to `*`, as this has the possibility of opening a security loophole when a shell script parses the filter's command line options.

You restore the original `lpd.conf` file to restore the system to normal operation.

```
h4: {226} % cd /etc
h4: {227} % cp lpd.conf.bak lpd.conf
```

When using the program method to return information, special consideration should be given to the `all` request. If there is not an explicit `all` entry, then the program should take appropriate steps to enumerate the values in the database, or report that there is a missing `all` entry to the appropriate administrative authority.

4.11.3. User Printcap Information

In addition to the system printcap, each user can define a private printcap file that will be read after the system printcap. Users can define **LPRng** client entries and can augment the system printcap information.

By default, `${HOME}/.printcap` is the the user printcap file. Here is a simple example of a user printcap file.

```
# remote printer - default
lp:lp=raw@localhost
  :ifhp=model=laserjet4
  :filter=/usr/local/libexec/filters/ifhp
# direct connection to printer over TCP/IP connection
lp:lp=10.0.0.5%9100
  :direct
  :ifhp=model=phaser
  :filter=/usr/local/libexec/filters/ifhp
```

The two examples show how a simple printer definition can be created. The first example shows how to create a simple way to send a file directly to a remote print queue after passing it through a filter. This is usually called *client side* filtering.

The second example is more interesting. Here we do the same thing, but we open a connection to the remote port on a host and send the print job. We do not spool the print job but send it directly. This is called *lightweight lpr* printing.

While the user printcap file is read after the system printcap file, the order of printcap entries is modified so that any entries that appeared in the user printcap file will appear before entries in the system printcap file. This allows users to modify the order in which printer entries are displayed.

4.12. Banner Printing and the OF filter

Banner or header pages can be printed at the beginning, end, or both at the beginning or end. The following flags control how and where banners are printed. These flags are listed in order of precedence.

`:sh`

Suppress all banner printing or header pages. This prevents any banners from being generated by the **lpd** print spooler and if present in a client printcap entry, will cause **lpr** not to put any banner printing information in the control file. Even if this flag is not present, then **LPRng** will not print a banner unless a banner printing program is specified with the `:bp, :bs`, `:be`, or `:sb` option.

`:ab`

Print a banner or header page, even if the user has not requested one. The `:sh` option has precedence over the `:ab` option.

`:hl`

The banner (header) is at the end (last page) of the job.

`:bs=/...` and `:be=/...` and

The `:bs` and `:be` options specify that a banner page is to be generated at the start and end of the job respectively, using indicated filter program. If the `:hl` flag has been set, only the `:be` will be used.

`:bp=...`

If there is no `:bs` or `:be` value when printing a banner at the start or end of the job respectively, then use the indicated filter program to generate a banner.

`:sb` and `:bl=....`

If there is no program specified to generate the banner and the `:sb` flag is set, send the `:bl` (banner line) string to the printer.

`:of=filter`

A filter used to process banners and other non-job file information.

`:suspend_of_filter`

Controls whether the `:of` filter is suspended or has its input terminated.

The **pclbanner**, **psbanner**, and **lpbanner** programs are part of the **LPRng** distribution and are usually installed in the same location as the **LPRng** supported filters. They produce a PCL, PostScript, or text banner respectively.

The OF filter (`:of=/path`) is used to process banner pages and to do any necessary setup to initialize the printer to handle banner pages. This filter has the following unusual behavior:

- It must be explicitly specified in the printcap file. It is not run by default.

If specified, it is started at the beginning of job printing and stays present throughout the entire job printing session.

- When printing individual files, the `:of` filter is sent a special *suspend yourself* two character string, `\031\001`. This will cause the `:of` filter to send itself a SIGSUSP (suspend) signal.
- The `:of` filter is restarted when any information not part of a print job file, such as the initialization string (`:ld` option), termination string (`:ld` option), or form feeds at start (`:fo`), end (`:fq`), and between job files (`:ff_separator`), and when banners are generated by the `:bp`, `:bs`, `:be`, or `:sb` option and need to be sent to the printer.

This rather baroque behavior is mostly historical in origin, and is very much embedded in the existing documentation and methodologies of the BSD print spooling system. Originally, when a printer port was opened, a special device initialization string was sent by the printer port device driver; this usually resulted in an extra page of paper being ejected by the printer. By opening the device once and then keeping it open, the print spooler would avoid the wasted paper. The reason for suspending the `:of` filter was simply to save the overhead of creating an extra processes.

Unfortunately, the `:of` filter suspension behavior is now a problem rather than a benefit. For example, for many devices to finish printing a page correctly the filter must be closed in order for it to flush buffers and for the low level drivers to properly finish. In order to provide this functionality, the `suspend_of_filter@` flag can be used. This will cause the **lpd** server to close the `:of` filters input, rather than sending it the suspend string, and to restart a new `:of` filter process when necessary.

4.13. Printing from lpr Directly To A Device

While the most reliable way to print is to send jobs to a print spooler, sometimes it is desirable to print directly to a printer. This method is supported by the special `:direct` printcap flag or the **lpr** `-Y` command line flag. The following shows the effects of this flag:

```
lpr -Y -Phost%port file1 file2 ...
Eqivalent to:
  ( for i in file1 file2 ... ; do
    $filter $i
  done ) | tcpip_connection( host%port)

lpr -Y -P/dev/lp file1 file2 ...
Eqivalent to:
  ( for i in file1 file2 ... ; do
    $filter $i
  done ) >>/dev/lp

lpr -Y -P '|/program' file1 file2 ...
Eqivalent to:
  ( for i in file1 file2 ... ; do
    $filter $i
  done ) | /program
```

The above examples show how we can use the command line options to send files directly to a printer. You can also create a printcap that will do the same:

```
lp:direct:lp=/tmp/a:remote_support=R
Command:
  lpr -Plp file1 file2 ...
Equivalent to:
lpr -P/tmp/a -Y file1 file2 ...

Example:
h4: {228} % lp -P/tmp/a /tmp/hi
h4: {229} % cat /tmp/a /tmp/hi
hi
h4: {230} % lp -Plp /tmp/hi
h4: {231} % cat /tmp/a /tmp/hi
hi
hi
```

The **lpr** `-X filter` option allows us to specify a user filter on the command line. We will use a simple example to show how this capability could be used in practice. Create the `/tmp/pass` file with the following contents, and give it executable permissions as shown below:

```
#!/bin/sh
# /tmp/pass file
echo LEADER
cat
echo TRAILER
exit 0
```

Execute the following commands to print the `/tmp/hi` file and observe the results:

```
h4: {232} % cp /dev/null /tmp/a
h4: {233} % lpr -P/tmp/a -X /tmp/pass /tmp/hi
h4: {234} % cat /tmp/a
LEADER
hi
TRAILER
```

As we see from the example, our filter has processed the input file and added the `LEADER` and `TRAILER` strings. In practice, the actual processing of the input job would be far more elaborate, and may do such things as incorporate files or other material available only on the local system. We can also use a printcap entry:

```
lp:direct:lp=/tmp/a:filter=/tmp/pass

h4: {235} % cp /dev/null /tmp/a
h4: {236} % lpr -Plp /tmp/hi
h4: {237} % cat /tmp/a
```

```
LEADER
hi
TRAILER
```

4.14. Moving Jobs From Queue to Queue and Redirecting Queues

The `lpc move` command is used to move jobs in one queue to another queue on an individual basis, while the `lpc redirect` command redirects all incoming jobs to a new queue. Edit the printcap file so it has contents indicated below, use `checkpc -f` to check the printcap, and then use `lpc reread` to restart the **lpd** server.

```
lp:force_localhost
lp:server
  :sd=/var/spool/lpd/%P
  :lp=lp2@localhost
lp2:force_localhost
lp2:server
  :sd=/var/spool/lpd/%P
  :lp=/tmp/lp2
```

Execute the following commands to print the `/tmp/hi` file and observe the results:

```
h4: {238} % lpc stop lp lp2
Printer: lp@h4
lp@h4.private: stopped
Printer: lp2@h4
lp2@h4.private: stopped
h4: {239} % lpr /tmp/hi
h4: {240} % lpq -a
Printer: lp@h4  (printing disabled)
 Queue: 1 printable job
 Server: no server active
 Rank   Owner/ID           Class Job Files      Size Time
1      papowell@h4+659       A   659 /tmp/hi       3 08:04:03
Printer: lp2@h4  (printing disabled)
 Queue: no printable jobs in queue
h4: {241} % lpc move lp papowell lp2
Printer: lp@h4
lp: selected 'papowell@h4+659'
lp@h4.private: move done
h4: {242} % lpq -a
Printer: lp@h4  (printing disabled)
 Queue: no printable jobs in queue
 Status: job 'papowell@h4+659' removed at 08:19:24.962
Printer: lp2@h4  (printing disabled)
 Queue: 1 printable job
 Server: no server active
```

```
 Rank   Owner/ID           Class Job Files       Size Time
1      papowell@h4+659       A   659 /tmp/hi        3 08:19:24
```

We first stop the queues so that the jobs will remain in them. We then use the `lpc move fromqueue id toqueue` command to select a job in the *fromqueue* and move it to the *toqueue*. A list of job numbers, job IDs, or glob patterns to match job IDs can be used to select the job.

The `lpc redirect fromqueue toqueue` will cause all incoming jobs to be redirected to the specified queue. You can execute the following commands and observe the results.

```
h4: {243} % lpc redirect lp lp2
Printer: lp@h4
forwarding to 'lp2'
lp@h4.private: redirected
h4: {244} % lpq -a
Printer: lp@h4  (printing disabled) (redirect lp2)
 Queue: no printable jobs in queue
Printer: lp2@h4  (printing disabled)
 Queue: no printable jobs in queue
h4: {245} % lpr /tmp/hi
h4: {246} % lpq -a
Printer: lp@h4  (printing disabled) (redirect lp2)
 Queue: no printable jobs in queue
 Status: job 'papowell@h4+935' removed at 09:08:21.410
Printer: lp2@h4  (printing disabled)
 Queue: 1 printable job
 Server: no server active
 Rank   Owner/ID           Class Job Files        Size Time
1      papowell@h4+935       A   935 /tmp/hi         3 09:08:21
```

To turn redirection off, use `lpc redirect queue off` as shown in the example below:

```
h4: {247} % lpc redirect lp off
Printer: lp@h4
forwarding off
h4: {248} % lpq
Printer: lp@h4  (printing disabled)
 Queue: no printable jobs in queue
 Status: job 'papowell@h4+935' removed at 09:08:21.410
```

4.15. Print Job Classes, User Requested Job Priority, and Form Support

The **LPRng** software allows users to assign a class name to print jobs using the `lpr -Cname` option. This causes the **lpr** command to put the line `Cname` in the control file. By default, the (upper cased) first letter of the class name is used to assign a user requested priority to the job, with `A` being the default lowest priority and `Z` being the highest.

The `ignore_requested_user_priority` printcap option can be used to ignore the user requested priority and jobs will be printed in the normal first-in first-out order.

LPRng also makes use of the class information to do form support and restrict printing to a specific set of classes. By default the job class information is ignored, but the `lpc class` command can be used to specify one or more classes (actually glob patterns) to be printed. This facility can be used to do support printing of jobs that require a specific form setup. Here is a simple example of how to use this facility.

Edit the printcap file so it has contents indicated below, use `checkpc -f` to check the printcap, and then use `lpc reread` to restart the **lpd** server.

```
lp:force_localhost
lp:server
  :sd=/var/spool/lpd/%P
  :lp=lp2@localhost
lp2:force_localhost
lp2:server
  :sd=/var/spool/lpd/%P
  :lp=/tmp/lp2
```

Execute the following commands to print the `/tmp/hi` file and observe the results:

```
h4: {249} % lpc class lp red
Printer: lp@h4
classes printed 'red'
lp@h4.private: class updated
h4: {250} % lpq
Printer: lp@h4  (classes red)
 Queue: no printable jobs in queue
h4: {251} % lpr /tmp/hi
h4: {252} % lpq
Printer: lp@h4  (classes red)
 Queue: no printable jobs in queue
 Holding: 1 held jobs in queue
 Server: no server active
 Rank   Owner/ID           Class Job Files        Size Time
holdclass papowell@h4+82     A    82 /tmp/hi         3 09:29:52
h4: {253} % lpr -Cred /tmp/hi
h4: {254} % lpq
Printer: lp@h4  (classes red)
 Queue: no printable jobs in queue
 Holding: 1 held jobs in queue
 Server: no server active
 Status: job 'papowell@h4+89' removed at 09:30:13.569
```

```
 Rank   Owner/ID           Class Job Files        Size Time
holdclass papowell@h4+82     A    82 /tmp/hi         3 09:29:52
```

As seen in the example, we set the queue class to `red`, and then sent a (default) class `A` job to the printer. It was not printed, and is listed with `holdclass` status. We sent another job which was immediately printed.

We can change the print queue class at any time, and then new class will then control what jobs are printed. To disable the class selection, use the `lpc class queue off` command.

```
h4: {255} % lpc class lp off
Printer: lp@h4
all classes printed
lp@h4.private: class updated
```

4.16. Holding and Releasing Jobs

The **LPRng** software has a wide range of facilities to hold or temporarily prevent jobs from printing. Jobs can be individually held or all jobs submitted to a queue can be held until released by an operator. Some administrators use the `holdall` facility and a **cron** script to cause jobs to be printed at specific times. The `lpc holdall` command causes all jobs submitted to a queue to be held until released with the `lpc release` command. The `lpc noholdall` command disables the `holdall` operation.

Edit the printcap file so it has contents indicated below, use `checkpc -f` to check the printcap, and then use `lpc reread` to restart the **lpd** server.

```
lp:force_localhost
lp:server
  :sd=/var/spool/lpd/%P
  :lp=lp2@localhost
lp2:force_localhost
lp2:server
  :sd=/var/spool/lpd/%P
  :lp=/tmp/lp2
```

Execute the following commands to print the `/tmp/hi` file and observe the results:

```
h4: {256} % lpc holdall lp
Printer: lp@h4
lp@h4.private: holdall on
h4: {257} % lpq
Printer: lp@h4  (holdall)
 Queue: no printable jobs in queue
h4: {258} % lpr /tmp/hi
h4: {259} % lpq
```

```
Printer: lp@h4  (holdall)
 Queue: no printable jobs in queue
 Holding: 1 held jobs in queue
 Server: no server active
 Rank   Owner/ID           Class Job Files        Size Time
hold   papowell@h4+213       A   213 /tmp/hi         3 09:45:05
h4: {260} % lpc release lp 213
Printer: lp@h4
lp: selected 'papowell@h4+213'
lp@h4.private: started
h4: {261} % lpq
Printer: lp@h4  (holdall)
 Queue: no printable jobs in queue
 Status: job 'papowell@h4+213' removed at 09:45:22.570
```

The `lpc holdall` command causes all jobs to be held. We spool a job, and then use the `lpc release` command to release the selected job. We disable the `holdall` operation using the `lpc noholdall` command.

```
h4: {262} % lpc noholdall lp
Printer: lp@h4
lp@h4.private: holdall off
```

You can also use the `lpc hold` command to select individual jobs in a spool queue to be held. This command is useful if there is a set of jobs which require special handling or printing at a later date. The following example shows how this command is used. We use the `lpc stop` and `lpc start` commands to simulate the normal delays in print spooling operations.

```
h4: {263} % lpc stop lp
Printer: lp@h4
lp@h4.private: stopped
h4: {264} % lpq
Printer: lp@h4  (printing disabled)
 Queue: no printable jobs in queue
 Status: job 'papowell@h4+495' removed at 10:10:50.629
h4: {265} % lpr /tmp/hi
h4: {266} % lpr /tmp/hi
h4: {267} % lpq
Printer: lp@h4  (printing disabled)
 Queue: 2 printable jobs
 Server: no server active
 Rank   Owner/ID           Class Job Files        Size Time
1      papowell@h4+459       A   459 /tmp/hi         3 10:40:32
2      papowell@h4+461       A   461 /tmp/hi         3 10:40:34
h4: {268} % lpc hold lp 459
Printer: lp@h4
lp: selected 'papowell@h4+459'
lp@h4.private: updated
h4: {269} % lpq
```

```
Printer: lp@h4  (printing disabled)
 Queue: 1 printable job
 Holding: 1 held jobs in queue
 Server: no server active
 Rank    Owner/ID           Class Job Files        Size Time
1      papowell@h4+461       A   461 /tmp/hi         3 10:40:34
hold   papowell@h4+459       A   459 /tmp/hi         3 10:40:32
```

In the next example we show how to use the `lpc hold` command to select and hold an individual job. Then we start the queue and see what happens:

```
h4: {270} % lpc start
Printer: lp@h4
lp@h4.private: started
h4: {271} % lpq
Printer: lp@h4
 Queue: no printable jobs in queue
 Holding: 1 held jobs in queue
 Server: no server active
 Status: job 'papowell@h4+461' removed at 10:41:24.873
 Rank    Owner/ID           Class Job Files        Size Time
hold   papowell@h4+459       A   459 /tmp/hi         3 10:40:32
h4: {272} % lpc release lp 459
Printer: lp@h4
lp: selected 'papowell@h4+459'
lp@h4.private: started
h4: {273} % lpq
Printer: lp@h4
 Queue: no printable jobs in queue
 Status: job 'papowell@h4+459' removed at 10:41:39.457
```

As we see, the held job is not printed until we release it, and then is processed normally.

The printcap `:ah` (autohold) option has the same effect as the `lpc holdall` command but its actions cannot be disabled by the `lpc noholdall` command.

4.17. Load Balance Queues and Printer Pools

A Load Balance Queue provides a way to use multiple printers for a single print queue. All jobs are normally sent to the main or load balance queue which then dispatches the jobs to *server* queues or printers that do the actual printing as they become available. You can also send jobs to the individual server printers if they have special processing or setups required for a particular job. Because all of the server printers are shared by the load balance queue, they are said to be in a *printer pool*.

Edit the printcap file so it have the contents indicated below, create the `/tmp/lp2` and `/tmp/lp3` files with `0777` permissions, use `checkpc -f` to check the printcap, and then use `lpc reread` to restart the **lpd** server.

```
# printcap
lp:force_localhost
lp:server
  :sd=/var/spool/lpd/%P
  :sv=lp2,lp3
lp2:force_localhost
lp2:server
  :ss=lp
  :sd=/var/spool/lpd/%P
  :lp=/tmp/lp2
lp3:force_localhost
lp3:server
  :ss=lp
  :sd=/var/spool/lpd/%P
  :lp=/tmp/lp2
```

The `:sv=...` option flags the queue as a load balance queue and lists the queues that are used for load balancing. The `:ss=...` option flags the queue as a server for a load balance queue and specifies the name of the load balance queue. When a job is sent to the load balance queue the **lpd** server checks to see which server queues are available and then the first one to become available.

Execute the following commands to print the `/tmp/hi` file and observe the results:

```
h4: {274} % lpq
Printer: lp@h4  (subservers lp2, lp3)
 Queue: no printable jobs in queue
 Status: job 'papowell@h4+42' removed at 07:29:57.924
Server Printer: lp2@h4  (serving lp)
 Queue: no printable jobs in queue
Server Printer: lp3@h4  (serving lp)
 Queue: no printable jobs in queue
h4: {275} % lpr /tmp/hi
h4: {276} % lpq
Printer: lp@h4  (subservers lp2, lp3)
 Queue: 1 printable job
 Server: pid 4063 active
 Status: waiting for subserver to finish at 07:31:08.074
 Rank   Owner/ID           Class Job Files        Size Time
1      papowell@h4+62        A    62 /tmp/hi         3 07:31:07
Server Printer: lp2@h4  (serving lp)
 Queue: no printable jobs in queue
Server Printer: lp3@h4  (serving lp)
 Queue: no printable jobs in queue
h4: {277} % lpq
Printer: lp@h4  (subservers lp2, lp3)
 Queue: no printable jobs in queue
 Status: no more jobs to process in load balance queue at 07:31:12.317
Server Printer: lp2@h4  (serving lp)
 Queue: no printable jobs in queue
Server Printer: lp3@h4  (serving lp)
 Queue: no printable jobs in queue
```

```
 Status: job 'papowell@h4+62' removed at 07:31:10.311
```

The first **lpq** command shows how the status is displayed for a load balance queue - the queue and its server queues are shown as well. Next, we use **lpr** to print a job (job id `papowell@h4+62`). We then use a couple of **lpq** commands to see how the job is first sent to the `lp` queue, which then forwards it to the `lp3` queue, which then processes it and removes it. (For purposes of demonstration we have artificially slowed down the operation of the load balance queue so that the jobs will remain in the queue for sufficient time for us to display their status.) We can send another job to the load balance queue:

```
h4: {278} % lpr /tmp/hi
h4: {279} % lpq
Printer: lp@h4  (subservers lp2, lp3)
 Queue: no printable jobs in queue
 Status: no more jobs to process in load balance queue at 07:37:17.953
Server Printer: lp2@h4  (serving lp)
 Queue: no printable jobs in queue
 Status: job 'papowell@h4+89' removed at 07:37:15.936
Server Printer: lp3@h4  (serving lp)
 Queue: no printable jobs in queue
 Status: job 'papowell@h4+81' removed at 07:36:40.116
```

This time we see that the job was put in `lp2`. The normal load balance queue operation is to use the server queues in round robin order.

While this simple configuration is suitable for a large number of configurations, there are situations where server queue must be chosen *dynamically*. For example, if the server queues are actually transferring jobs to remote clients then as soon as the job is sent to the remote client the queue appears empty and available for use. To correctly check to see if the queue is available, the status of the remote queue or destination of the server queue must be checked.

To handle this situation, a `:chooser` program or filter can be used. When the load balance queue is trying to decide where to send a job, it first checks the server queues to see if they are enabled for printing. If a `:chooser` program is specified in the load balance queue printcap entry, then it is started with the normal filter options and environment variables, supplemented as discussed below. The `:chooser` program will read a list of candidate queues from its STDIN, write the chosen one to its STDOUT, and then exit. The **lpd** server checks the `:chooser` exit code - if it is zero (successful) then the chosen queue is used otherwise the exit code is used for the result value of processing the job. This allows the chooser process to not only control the destination of the job but also to hold, remove, or abort the job handling process. If the `:chooser` does not specify a queue, then the job is skipped and another job is chosen.

One side effect of the using a chooser program is that while there are jobs that can be processed in the queue the **lpd** server needs to periodically check to see if a server queue has become available. If it did this continually then a very high load would be put on the system. Instead, the `chooser_interval` option specifies a maximum time in seconds (default 10 seconds) between the times that the **lpd** server checks to see if there is an available server.

Normally, the `chooser` is applied to the first job in the queue. If the job cannot be printed then **lpd** will wait for the `chooser_interval` time. However, the chooser can also be used to direct jobs by their

characteristics, or other criteria. This means that the entire spool spool queue has to be scanned for work. If the `:chooser_scan_queue` flag is set to 1, then all of the jobs are tested to see if they can be sent to an appropriate destination.

Edit the printcap file so it have the contents indicated below, create the `/tmp/lp2` and `/tmp/lp3` files with `0777` permissions. Then create the `/tmp/chooser.script` with the contents indicated below, and give it `0755` (executable) permissions. Make sure that the path to the **head** program used in `chooser.script` is correct. Use `checkpc -f` to check the printcap, and then use `lpc reread` to restart the **lpd** server.

```
# printcap
lp:force_localhost
lp:server
  :sd=/var/spool/lpd/%P
  :sv=lp2,lp3
  :chooser=/tmp/chooser.script
lp2:force_localhost
lp2:server
  :ss=lp
  :sd=/var/spool/lpd/%P
  :lp=/tmp/lp2
lp3:force_localhost
lp3:server
  :ss=lp
  :sd=/var/spool/lpd/%P
  :lp=/tmp/lp2

# /tmp/chooser.script

#!/bin/sh
echo CHOOSER $0 $@ >>/tmp/chooser
set >>/tmp/chooser
/usr/bin/head -1
exit 0
```

Now run the following commands:

```
h4: {280} % lpr /tmp/hi
h4: {281} % lpq -lll
Printer: lp@h4  (subservers lp2, lp3)
 Queue: no printable jobs in queue
 Status: CHOOSER selected 'lp3' at 14:02:50.605
 Status: transferring 'papowell@h4+178'
      to subserver 'lp3' at 14:02:50.614
 Status: transfer 'papowell@h4+178'
      to subserver 'lp3' finished at 14:02:50.624
 Status: job 'papowell@h4+178' removed at 14:02:50.632
 Status: starting subserver 'lp3' at 14:02:50.632
 Status: waiting for server queue process to exit at 14:02:50.651
 Status: subserver pid 10182 exit status 'JSUCC' at 14:02:52.872
 Status: no more jobs to process in load balance queue at 14:02:52.879
```

```
Server Printer: lp2@h4  (serving lp)
 Queue: no printable jobs in queue
Server Printer: lp3@h4  (serving lp)
 Queue: no printable jobs in queue
 Status: waiting for subserver to exit at 14:02:50.748
 Status: subserver pid 10183 starting at 14:02:50.820
 Status: accounting at start at 14:02:50.821
 Status: opening device '/tmp/lp3' at 14:02:50.833
 Status: printing job 'papowell@h4+178' at 14:02:50.834
 Status: processing 'dfA178h4.private', size 3, format 'f', \
      IF filter 'none - passthrough' at 14:02:50.838
 Status: printing finished at 14:02:50.839
 Status: accounting at end at 14:02:50.839
 Status: finished 'papowell@h4+178', status 'JSUCC' at 14:02:50.841
 Status: subserver pid 10183 exit status 'JSUCC' at 14:02:50.843
 Status: lp3@h4.private: job 'papowell@h4+178' printed at 14:02:50.856
 Status: job 'papowell@h4+178' removed at 14:02:50.871
```

As you see from the example above, the CHOOSER selected `lp3` for use. Let us look at the `/tmp/chooser` file and see how the `chooser.script` program was run:

```
CHOOSER -Apapowell@h4+113 -CA -D2000-06-01-14:02:13.313 -Hh4.private \
   -J/tmp/hi -Lpapowell -Plp -Qlp -aacct -b3 -d/var/tmp/LPD/lp \
   -hh4.private -j113 -kcfA113h4.private -l66 -npapowell -sstatus \
   -t2000-06-01-14:02:13.379 -w80 -x0 -y0 acct
USER=papowell
LD_LIBRARY_PATH=/lib:/usr/lib:/usr/5lib:/usr/ucblib
HOME=/home/papowell
PRINTCAP_ENTRY=lp
 :chooser=/var/tmp/LPD/chooser
 :lp=/tmp/lp
 :sd=/var/tmp/LPD/lp
 :server
 :sv=lp2,lp3

lp2=change=0x0
 done_time=0x1
 held=0x0
 move=0x0
 printable=0x0
 printer=lp2
 printing_aborted=0x0
 printing_disabled=0x0
 queue_control_file=control.lp2
 server=0
 spooldir=/var/tmp/LPD/lp2
lp3=change=0x0
 done_time=0x2
 held=0x0
 move=0x0
 printable=0x0
```

```
 printer=lp3
 printing_aborted=0x0
 printing_disabled=0x0
 queue_control_file=control.lp3
 server=0
 spooldir=/var/tmp/LPD/lp3
PS1=$
OPTIND=1
PS2=>
SPOOL_DIR=/var/tmp/LPD/lp
LOGNAME=papowell
CONTROL=Hh4.private
Ppapowell
J/tmp/hi
CA
Lpapowell
Apapowell@h4+113
D2000-06-01-14:02:13.313
Qlp
N/tmp/hi
fdfA113h4.private
UdfA113h4.private
```

As you can see, the program is invoked with the same options as a normal filter. In addition, the printcap information for each server queue is passed in an environment variable with the name of the server queue. This means that if there is information needed by the chooser program to test for the availability of hardware, etc., this can be placed in the printcap information.

One of the limitations of using the `:chooser` program is that you may have a high overhead associated with running the program. The **LPRng** software provides support for linking in a user provided routine that does the same thing as the `:chooser` program. This routine has the following API or interface:

```
Printcap Option: chooser_routine
   chooser_routine@  - default - do not use chooser routine
   chooser_routine   - use chooser routine

Configuration:
   configure --with-chooser_routine=name --with-user_objs=objectfile.o
      defines the CHOOSER_ROUTINE compilation option to name
      includes the objectfile.o in the library.

extern int CHOOSER_ROUTINE( struct line_list *servers,
    struct line_list *available, int *use_subserver );
  servers:    all subserver queues for this load balance queue
  available:  subserver queues to choose from
  use_subserver: chosen subserver queue
  RETURNS:
     0 - use the 'use_subserver' value as index into servers
         list for server to use
     != 0 - set job status to value returned.
```

See the LPRng/src/common/lpd_jobs.c and LPRng/src/common/user_objs.c files for details of the servers, available, and user_subserver parameters. The user_objs.c file provides a simple template that can be used as a starting point for a more complex routine. You should modify the code in the user_objs.c file and then use the configure options shown above to cause the user_objs.c file to be compiled and linked into the **LPRng** executables.

4.18. Routing Jobs To Print Queues

A *routing queue* is similar in concept to a load balance queue in that it transfers a job to a (different) print queue, but the job destination is chosen at the time the job is submitted to the queue rather than at the time the job is removed from the queue. A routing queue can modify the job control file, multiple copies of the same job can be sent to the same or different printers, and the job can be held, rejected, or processed immediately.

Edit the printcap file so it have the contents indicated below, create the /tmp/lp2 and /tmp/lp3 files with 0777 permissions. Create the /tmp/router.script with the contents indicated below, and give it 0755 (executable) permissions. Use checkpc -f to check the printcap, and then use lpc reread to restart the **lpd** server.

```
# printcap
lp:force_localhost
lp:server
  :lp=/dev/null
  :sd=/var/spool/lpd/%P
  :router=/tmp/router.script
lp2:force_localhost
lp2:server
  :sd=/var/spool/lpd/%P
  :lp=/tmp/lp2
lp3:force_localhost
lp3:server
  :sd=/var/spool/lpd/%P
  :lp=/tmp/lp2

# /tmp/router.script

#!/bin/sh
/bin/cat <<EOF
dest lp2
copies 2
Cred
end
dest lp3
end
EOF
exit 0
```

The `router.script` will write the routing information to its STDOUT. For our example, we want the destination `lp2` to get two copies of the job and we want to change the class to `red`. Now run the following commands:

```
h4: {282} % lpc stop all
Printer: lp@h4
lp@h4.private: stopped
Printer: lp2@h4
lp2@h4.private: stopped
Printer: lp3@h4
lp3@h4.private: stopped
h4: {283} % lpq
Printer: lp@h4 (dest lp@localhost) (printing disabled) (dest lp2, lp3)
 Queue: no printable jobs in queue
Printer: lp2@h4  (printing disabled)
 Queue: no printable jobs in queue
Printer: lp3@h4  (printing disabled)
 Queue: no printable jobs in queue
h4: {284} % lpr /tmp/hi
h4: {285} % lpq
Printer: lp@h4 (dest lp@localhost) (printing disabled) (dest lp2, lp3)
 Queue: 1 printable job
 Server: no server active
 Rank   Owner/ID           Class Job Files        Size Time
1      papowell@h4+235       A   235 /tmp/hi         3 16:14:22
 -          papowell@h4+235.1       ->lp2 <cpy 0/2>
 -          papowell@h4+235.2       ->lp3
Printer: lp2@h4  (printing disabled)
 Queue: no printable jobs in queue
Printer: lp3@h4  (printing disabled)
 Queue: no printable jobs in queue
```

The status reported for the spooled job indicates that the job is routed to `lp2`, and that two copies will be sent. The `:destinations` option in the printcap entry causes **lpq** to display the contents of the specified queues. Now execute the following commands:

```
h4: {286} % lpc start
Printer: lp@h4
lp@h4.private: started
h4: {287} % lpq
Printer: lp@h4 (dest lp@localhost) (destinations lp2, lp3)
 Queue: no printable jobs in queue
 Status: job 'papowell@h4+235' removed at 16:14:37.491
Printer: lp2@h4  (printing disabled)
 Queue: 2 printable jobs
 Server: no server active
 Rank   Owner/ID           Class Job Files        Size Time
1      papowell@h4+235.1C1   A   235 /tmp/hi         3 16:14:36
2      papowell@h4+235.1C2   A   236 /tmp/hi         3 16:14:37
Printer: lp3@h4  (printing disabled)
```

```
 Queue: 1 printable job
 Server: no server active
 Rank   Owner/ID           Class Job Files        Size Time
1      papowell@h4+235.2     A   237 /tmp/hi         3 16:14:37
h4: {288} % more /var/spool/lpd/lp2/cfA235*
Hh4.private
Ppapowell
J/tmp/hi
Cred
Lpapowell
Apapowell@h4+235.1C1
D2000-06-01-16:03:25.237
Qlp
N/tmp/hi
fdfA235h4.private
UdfA235h4.private
```

As you can see, two copies of the job has been transferred to `lp2` and one to `lp3`, each with a different job number, If we examine the control file for the jobs in the `lp2` spool queue, we will find that the `C` or class information has been changed to `red`.

For details about all of the capabilities of the routing filter, see Dynamic Routing. Here is a summary of the information that the routing filter can put into the routing file.

dest queue

: Route this job to `queue`. The `queue@host` form will transfer the job to the queue on the named host.

copies N

: Send N copies of this job to the destination.

priority C

: Set the job priority letter to `C`, where C is a single upper case letter.

Cvalue

: Set the control file line starting with `C` to `Cvalue`.

The exit status of the routing filter controls how the job will be processed. If the exit code is JSUCC (0), then the job will be processed normally, JHOLD will hold the job until released, JREMOVE will remove the job, and so forth.

4.19. Job Options and the Z Control File Entry

Many printers have special capabilities such as printing in landscape mode, duplex printing, binding, or stapling. These capabilities are usually invoked or enabled by the print spooler sending special printer

control commands to the printer based on values it finds in the control file. The **LPRng** print spooler uses the `Z` line in the control file to specify these options, while other print spoolers such as the Sun Microsystems Solaris **lp** system pass them on the `S` line.

Job formatting options are specified using the `lpr -Z` option. The **lpr** program concatenates the `-Z` options and puts them in the control file as a single `Z` line. For example:

```
h4: {289} % lpc stop
Printer: lp@h4
lp@h4.private: stopped
h4: {290} % lpr -Zthis -Zthat /tmp/hi
h4: {291} % cat /var/spool/lp/cf*
Hh4.private
Ppapowell
J/tmp/hi
CA
Lpapowell
Zthis,that
Apapowell@h4+115
D2000-05-05-10:05:41.351
Qlp
N/tmp/hi
fdfA115h4.private
UdfA115h4.private
```

As we see, the `Z` options have been put into the control file on the `Z` line. The `Z` option values are passed to filters on the command line as the `-Z` command line option. These values are used by the **ifhp** filter to determine what control commands to send to the printer and how to format the print job output. Because each printer is different and supports a different set of capabilities it is impossible to have a set of job options supported across all printers. The following are supported by the **ifhp** configuration where possible. Many of these options rely on the printer supporting PostScript or having the appropriate PCL commands to do the indicated operation.

- -Zlandscape -Zportrait - select landscape or portrait orientation.
- -Zduplex -Zsimplex - select duplex (both sides of a page) or simplex (single side of a page) printing.
- -Zletter -Zlegal -Zledger -Za4 -Za5 -Zenvelope -Ztransparency - select a paper size
- -Zinupper -Zinmiddle -Zinlower - select input media from the appropriate input tray
- -Zmanual - select input from the manual feed

4.19.1. Setting Job Options Using the Printcap

An alternative to this method of using **lpr** and the `-Z` option is to define a set of spool queues which will put the necessary options into the job control file. This can be done by the **lpr** program when the job is generated, or by the **lpd** spooler when the job is processed. The The options specified by the `:prefix_z`, `:append_z`, and `:delete_z` are prefixed, appended, or deleted from the current set of `Z`

control file options by the **lpr** program when the job is submitted and they are specified in the printcap for the queue, or by the **lpd** spooler when the job is submitted to the queue. We can use this capapbility to configure print queues to a desired set of `Z` options into the control file. For example:

```
landscape:lp=%P@server
landscape:server:tc=.common
  :lp=raw@server:append_z=landscape:delete_z=portrait
raw:server:tc=.common:lp=....
  :filter=/usr/local/libexec/filters/ifhp
.common:sd=/var/spool/lpd/%P
```

When a job is sent to the `landscape` queue, the control file `Z` line will have the `portrait` option removed and the `landscape` option appended. The `:delete_z` values are glob patterns and options that match are removed from the option list. Options are assumed to be separated by commas or semicolons in the option list.

4.19.2. Converting SystemV Options to LPRng Options

On some SystemV **lp** print spoolers, the `lp -o option`, puts the option information into the control file `S` line, and on other systems on the puts the option information into the control file `O` line. To convert these options to **LPRng** `Z` options use the `:prefix_option_to_option=from,from... to` facility to prefix the *from* control file lines to the *to* control file line. For example:

```
# System V to LPRng - S and O to Z options
convert:server:tc=.common
  :lp=raw@server:prefix_option_to_option=S,O Z
# LPRng to System V O options
convert:server:tc=.common
    :lp=raw@server:prefix_option_to_option=Z O
```

4.19.3. Selecting a Single Option - Muliple Queues

Here is an example of how you can set up queues that will append the appropriate `Z` option to select landscape mode, do duplex printing, or select legal or ledger size paper:

```
landscape:lp=%P@server
landscape:server:tc=.common
    :lp=raw@server:append_z=landscape
duplex:lp=%P@server
duplex:server:tc=.common
    :lp=raw@server:append_z=duplex
ledger:lp=%P@server
ledger:server:tc=.common
    :lp=raw@server:append_z=ledger
```

```
legal:lp=%P@server
legal:server:tc=.common
    :lp=raw@server:append_z=legal
raw:server:tc=.common:lp=....
  :filter=/usr/local/libexec/filters/ifhp
.common:sd=/var/spool/lpd/%P
```

The problem with this method is that for each option we need to define a queue whose only purpose is to append the appropriate option and then forward this to the main print queue.

4.19.4. Selecting Multiple Options - Single Queue

In the previous section, we showed how to set up a queue that would append a single option to the control file `Z` line. If we want to have combinations of option options specified by the printer name then we will have to create a large number of queues each with a different set of options and each appending a different set of values. The problem becomes compounded when we have many printers, each of which requires these options.

The solution to this problem originated with the **apsfilter** program written by Andreas Klemm and Thomas Bueschgens. They made the observeration that if we know the name of the print queue then we can use this name to select options for the printer. The **LPRng** provides this functionality by using wildcard queues and *editing* or *filtering* the control file when the job is submitted to the spool queue.

The `incoming_control_filter=/path` filter processes the incoming job control or job ticket file. It can be used to values in the job ticket of incoming jobs. It reads the control file on its STDIN and writes the new or modified values on STDOUT. A 0 exit code value causes normal processing of the job, `JHOLD` will hold the job, and any other value will cause the job to be discarded. The `incoming_control_filter` filter can modify priority or other job options, including using the `move=` field to cause a job to be redirect to another spool queue or printer. Only changes to the jobs options need to be generated by the `incoming_control_filter=/path` filter.

The input and output have the format:

```
INPUT:
 X<option>       - option from control file
 X=<option>      - alternative option format
 key=<option>    - spooler option
 X==<option>     - option starting with = sign

OUTPUT:
 X                    - delete option or value
 X=                   - delete option or value
 X<option>       - set option value
 X=<option>      - set option value
 key=<option>    - set value of 'key' to option
 key=                 - delete option or value
```

In addition to modifying job options, the contents of the jobs data files can be modified or the data files removed. Any data files with a 0 length will be removed from the job. If all of the data files have a 0 length then the job will be discarded. Modification of job options may have unforseen effects on

The following shows how we can set up a single queue that will allow various combinations of options to be selected by the format of the queue name:

```
# for clients
pr|pr_*:lp=%Q@server
# for server
pr|pr_*:server
  :tc=.common:lp=....
  :incoming_control_filter=/usr/local/libexec/filters/update_z
  :filter=/usr/local/libexec/filters/ifhp
.common:sd=/var/spool/lpd/%P
```

The `pr` and `pr_*` aliases will match printer `pr` all print queue names starting with `pr_`. We can then use various suffixes to select job options. The following filter program uses the `_landscape`, `_legal`, and `_ledger` suffixes to set the corresponding option in the `Z` file. This program and other are available in the **LPRng** distribution in the `UTILS` directory. You should note that additional options can be specified as desired.

```
#
#!/usr/bin/perl
# update_z script:
# Determine the options according to the format of the queue name
#  Inspired by the psfilter code of Andreas Klemm
#  and Thomas Bueschgens
# First, get command line arguments
#
use Getopt::Std;
my(%args,$Q,$Zopts,@file);
getopts(
"A:B:C:D:E:F:G:H:I:J:K:L:M:N:O:P:Q:R:S:T:U:V:W:X:Y:Z:" .
"a:b:cd:e:f:g:h:i:j:k:l:m:n:o:p:q:r:s:t:u:v:w:x:y:z:",
\%args );
# read stdin
@file = <STDIN>;
$Zopts = "";
# first use command line Queue name
$Q = $args{"Q"};
if( not $Q and (($Q) = grep(/^Q/,@file)) ){
   # next use control file Queue name
    chomp $Q if $Q;
}
# now we split up the name and use as parameters for Z options
while( $Q =~ /_([^_]+)/g ){
    # you can add them or test and then add them
    if( $1 eq "landscape"
        or $1 eq "legal"
        or $1 eq "ledger" ){
```

```
        $Zopts .= ",$1"
    }
}
if( $Zopts ){
    # remove leading comma
    $Zopts = substr( $Zopts, 1 );
    #replace or prefix Z options
    if( not (grep { s/$/,$Zopts/ if /^Z/; } @file) ){
        print "Z" . $Zopts . "\n";
    }
}
print @file if( @file );
exit 0

Example Input:
...
Z=over
Q=lp_landscape_ledger
...

Example output:

Z=over,landscape,ledger
Q=lp_landscape_ledger
```

The Perl script first uses the `getopts` function to parse the command line options. If there is not a command line `-Q` option then the control file `Q` line is used after stripping the trailing newline. The queue name is then split up into parts separated by underscores (_) and those used as option names. As shown in the example, the literal values are placed in the control file. You can also use the following code segment to translate short forms of options into longer ones:

```
while( $Q =~ /_([^_]+)/g ){
    # you can add them or test and then add them
    Zopts .= ",landscape" if( $1 eq "ld" );
    Zopts .= ",ledger" if( $1 eq "11" );
    Zopts .= ",legal" if( $1 eq "15" );
    Zopts .= ",a4" if( $1 eq "a4" );
}
```

4.20. Interfacing to Non-LPRng Spoolers

Given the large number of defective RFC1179 implementations that are currently in use, there will come a time when the administrator will discover that their printer with its built-in network interface, the non-UNIX based print spooler on a mainframe, or even a new print spooler on a new OS distribution will not accept jobs from the **LPRng** system. Usually this is caused by the presence, absence, or order of

lines in the control file being sent to the remote system. To deal with this particular problem, the `:bk`, `:control_file_line_order`, `:nline_after_file`, and `:control_filter` options are used.

The `:bk` (BSD Kompatibility) option causes the lpd server to remove all but an extremely small subset of lines from the control file, and to put the lines in the most commonly used order. In addition it will make the control and data files names extremely short and simple. This almost always solves compatibility problems when sending jobs to older *vintage* print spoolers or UNIX systems.

If this does not solve the problem, then you can specify the allowed control file lines and their order using the `control_file_line_order=...` option. For example, `control_file_line_order=CJPMD` would allow only control file lines starting with `C`, `J`, `P`, `M`, and `D`, and this order in the control file. Note that this does not provide *missing* line values, it only controls line values that are present in the control file. You should also use the `:bk` option as well.

You might run into some *really* unusual implementations where the control file `N` (file name) information must come *after* the control file name. This is forced by the `:nline_after_file` option.

If these horrible kludges do not solve your compatibility problems then we turn to the very large hammer and edit the control file. The very last step before transfering the control file to the remote server is to filter it using the `:control_filter=/path` program specified in the printcap. The `:control_filter` reads the control file from STDIN and writes the modified control file to STDOUT. No consistency checking or validity checks are done on the new control file and the result is transferred directly to the remote system. If the `:control_filter` exits with a 0 status, then the normal processing continues. Any other status will cause the transfer operation to be aborted and an error reported.

4.21. Debugging, Tracing, and Log Files

The **LPRng** software was designed and written to provide as high a level of diagnostic information as possible. This was largely in part due to the problems with portability, coding errors, and other human frailties. Approximately 80% of the **LPRng** source code concerns itself with checking return values from system functions and producing error messages, debugging and tracing information, and various facilities used for regression testing and diagnosis.

The approach used by **LPRng** is to produce *trace* output for the **LPRng** clients or *log* files for the **lpd** server that show the various events or flow of information through the **LPRng** system. There are several classes or types of actions that can be traced, and various levels of trace information generated. The interface used to control these actions are the command line `-D literals` flags and the `lpc debug` command.

First, we will look at how you can use the debugging facilities for the clients. Enter the following commands:

```
h4: {292} % lpr -D=
debug usage: -D [ num | key=num | key=str | flag | flag@ | flag+N ]*
  keys recognized: network[+N,@], database[+N,@], lpr[+N,@],
   lpc[+N,@], lprm[+N,@], lpq[+N,@], test=num, job=num, log[+N,@]
h4: {293} % lpr -V /tmp/hi
LPRng-3.7.2, Copyright 1988-2000 Patrick Powell, <papowell@lprng.com>
sending job 'papowell@h4+981' to lp@localhost
connecting to 'localhost', attempt 1
connected to 'localhost'
```

```
requesting printer lp@localhost
sending control file 'cfA981h4.private' to lp@localhost
completed sending 'cfA981h4.private' to lp@localhost
sending data file 'dfA981h4.private' to lp@localhost
completed sending 'dfA981h4.private' to lp@localhost
done job 'papowell@h4+981' transfer to lp@localhost
h4: {294} % lpr -D1 /tmp/hi
09:38:08.707 h4 [13991] lpr  Get_printer: original printer '<NULL>'
09:38:08.708 h4 [13991] lpr  Get_all_printcap_entries: starting
09:38:08.708 h4 [13991] lpr  Select_pc_info: looking for 'all', depth 0
09:38:08.708 h4 [13991] lpr  Select_pc_info: returning '<NULL>'
09:38:08.708 h4 [13991] lpr  Select_pc_info: looking for '*', depth 0
09:38:08.708 h4 [13991] lpr  Select_pc_info: returning '<NULL>'
09:38:08.708 h4 [13991] lpr  Dump_line_list: Get_all_printcap_entries
...
```

The `lpr -D=` causes the **lpr** (and other **LPRng** programs) to show what debugging flags are available. The `lpr -V` flag causes **lpr** to run in verbose mode and show its activities. Finally, we use `lpr -D1` to enable the simplest level of debugging. This will produce a trace of the various activities that **lpr** carries out. Try `lpr -D2`, `lpr -D3`, and so forth to see the increasing amount of detail that you get.

The *network* and *database* debug flags turn on debugging for the network facilities and the database (`lpd.conf`, `printcap`, and `lpd.perms`) lookups. Lets see what `lpr -Dnetwork` shows us:

```
Ch4: {295} % lpr -Dnetwork /tmp/hi
lp: getconnection: START host localhost, timeout 10, connection_type 1
lp: getconnection: fqdn found localhost.private, h_addr_list count 1
lp: Link_dest_port_num: port 2000 = 2000
lp: getconnection: sock 3, src ip 127.0.0.1, port 65209
lp: getconnection: dest ip 127.0.0.1, port 2000
lp: getconnection: connection to 'localhost' sock 3, errmsg 'No Error'
lp: Link_send: host 'localhost' socket 3, timeout 6000
lp: Link_send: str '^Blp
', count 4, ack 0x80447a0
lp: Link_send: final status NO ERROR
lp: Link_send: host 'localhost' socket 3, timeout 6000
lp: Link_send: str '^B135 cfA276h4.private
', count 22, ack 0x8044370
lp: Link_send: final status NO ERROR
lp: Link_send: host 'localhost' socket 3, timeout 6000
lp: Link_send: str 'Hh4.private
Ppapowell
J/tmp/hi
CA
Lpapowell
Apapowell@h4+276
D2000-06-02-09:44:52.369
Qlp
N/tmp/hi
fdfA276h4.private
UdfA276h4.private
```

```
', count 136, ack 0x8044370
lp: Link_send: final status NO ERROR
lp: Link_send: host 'localhost' socket 3, timeout 6000
lp: Link_send: str '^C3 dfA276h4.private
', count 20, ack 0x8044310
lp: Link_send: final status NO ERROR
lp: Link_send: host 'localhost' socket 3, timeout 6000
lp: Link_send: str '', count 1, ack 0x8044310
lp: Link_send: final status NO ERROR
```

As we see, we get a detailed exposition of the network connection and transfer steps. If you need or want more detail, try using `lpr -Dnetwork+2` or `lpr -Dnetwork+3`. You may want to try `lpr -Ddatabase` and observe the actions of the **lpr** program as it extracts information from the `lpd.conf` and `printcap` files. If you need or want more detail, try using `lpr -Ddatabase+2` or `lpr -Ddatabase+3`.

If you need to trace the activities of the **lpd** server, it becomes a little more complex. The lpd server has a single *listening* process that forks and creates individual processes to handle incoming requests. Debug or diagnose the main process actions by using `lpd -D...`. You may also want to use `lpd -F` to keep the server in the foreground so you can kill it off easily. Needless to say, you should also redirect the STDERR and STDOUT so that it goes to a file so that you can examine the voluminous records at your leisure. The following shows a typical main **lpd** process debugging session using the C Shell.

```
h4: {296} % lpd -F -D1 >&/tmp/logfile &
[2] 14299
h4: {297} % tail -f /tmp/logfile
2000-06-02-09:53:39.716 h4 [1200] Waiting Read_server_status: \
                          select status 1
2000-06-02-09:53:39.716 h4 [1200] Waiting Read_server_status: \
                          read status 1
2000-06-02-09:53:39.716 h4 [1200] Waiting Dump_line_list: \
    Read_server_status - input - 0x8047980, count 0, max 0, list 0x0
2000-06-02-09:53:39.716 h4 [1200] Waiting Read_server_status: \
                          select status 0
2000-06-02-09:53:39.716 h4 [1200] Waiting lpd: LOOP START
2000-06-02-09:53:39.716 h4 [1200] Waiting Get_max_servers: \
                          getrlimit returns 64
2000-06-02-09:53:39.716 h4 [1200] Waiting Get_max_servers: \
                          returning 32
2000-06-02-09:53:39.716 h4 [1200] Waiting lpd: \
                          max_servers 32, active 0
2000-06-02-09:53:39.716 h4 [1200] Waiting lpd: \
                          starting select timeout 'yes'

^C
h4: {298} % jobs
[1]  - Running                 lpd -F -D1 >& /tmp/logfile
h4: {299} % kill %1
```

We start the debugging session by running the **lpd** server in foreground mode. This causes it to send its output to STDOUT and STDERR. We redirect both of these to a file and put the **lpd** server in the background. Then we use `tail -f` to read from the log file. Finally, we kill off the **lpd** server.

This method is extremely difficult to use, as all of the output produced by the server and its subprocesses is sent to a single output file. If we want to debug the actions concerning a single queue, then we can use the queue *log file* and `lpc debug` command instead. The following options control debugging of an individual print queue.

lf=log

The log file for the queue. The queue server process will open this file and place debugging information into this file.

max_log_file_size=nnn

The maximum size of the log file in K bytes. When the queue server process first opens this file it will check to see if the file is larger than the maximum size. If it is, then it will truncate it. A zero (0) value suppress truncation.

min_log_file_size=nnn

When the log file is truncated only the the last nnn K bytes are retained.

db=options

These are debugging options for the spool queue. These options are permanent and cannot be changed by using the `lpc debug` facility.

The `lpc debug` command is used to set the debugging options in force for the spool queue. This is done by writing the debug options into the spool queue control file. Let us see how we can use this facility to trace the actions of printing a file.

Edit the printcap file so it have the contents indicated below, create the `/tmp/lp` and `/tmp/lp2` files with `0777` permissions. Use `checkpc -f` to check the printcap, and then use `lpc reread` to restart the **lpd** server.

```
# printcap
lp:force_localhost
lp:server
  :lp=/dev/null
  :sd=/var/spool/lpd/%P
  :lf=log
lp2:force_localhost
lp2:server
  :sd=/var/spool/lpd/%P
  :lp=/tmp/lp2
  :lf=log
```

Now execute the following commands:

```
h4: {300} % lpq
Printer: lp@h4
 Queue: no printable jobs in queue
h4: {301} % lpc debug lp 1
Printer: lp@h4
debugging override set to '1'
lp@h4.private: updated
h4: {302} % lpc status
 Printer Printing Spooling Jobs Server Subserver Redirect Status/(Debug)
lp@h4    enabled  enabled    0    none    none          (1)
h4: {303} % lpr /tmp/hi
h4: {304} % more /var/spool/lpd/lp2/log
2000-06-02-10:10:50.589 h4 [1201] (Server) lp: \
   Update_spool_info: printer 'lp'
2000-06-02-10:10:50.590 h4 [1201] (Server) lp: \
   Do_queue_jobs: printable 1, held 0, move 0
2000-06-02-10:10:50.590 h4 [1201] (Server) lp: \
   Do_queue_jobs: after Scan_queue next fd 5
2000-06-02-10:10:50.590 h4 [1201] (Server) lp: \
   Do_queue_jobs: MAIN LOOP
2000-06-02-10:10:50.590 h4 [1201] (Server) lp: \
   Do_queue_jobs: Susr1 before scan 0
2000-06-02-10:10:50.591 h4 [1201] (Server) lp: \
   Do_queue_jobs: chooser '<NULL>', chooser_routine 0
```

The `lpc debug` command sets the debug level to 1. We can use the `lpc status` command to see what debug flags or actions are currently specified for the spool queue. We then send a job to the spool queue and examine the log file contents.

Each line in the log file has a timestamp, the name of the host, the process id that produced it, and a heading that tells the action or activity that the process is performing, and the name of the print queue that is being processed and a trace message. By convention, the trace message lists the name of the routine that processed it and then the actual information. Some messages may extend over several lines, but each line has the standard header at the start of the line.

The default debug or trace actions were designed to trace problems with printing, as these are the most common. However, you can also use the **lpr**, **lpc**, **lprm**, or **lpq** option to cause the **lpd** server to trace the actions during the execution of an **lpr**, **lpc**, **lprm**, or **lpq** request.

The *log* option is used to test various logging facilities and is usually not used for general purpose debugging.

Chapter 5. LPRng Clients - lpr, lprm, lpq, lpc, lpstat

The **LPRng** software is a true set of client/server applications. The **LPRng** clients, **lpr**, **lpq**, **lprm**, and **lpc** connect to a **lpd** server using a TCP/IP connection. This means that you must have TCP/IP networking enabled on your workstation to use **LPRng**.

However, you do not need to have an external network connection to the Internet. For most single system users, the **lpd** server is running on the same workstation as the client program, and the clients will simply talk to the `localhost`.

5.1. Printer and Server Information

Options used:

- PRINTER, LPDEST, NPRINTER NGPRINTER *Environment variables*
- `force_localhost` FLAG *force clients to send requests to localhost*
- `require_explicit_q` FLAG *require queue to be specified*

When an **LPRng** client such as **lpr**, **lpq**, **lprm**, or **lpc** needs to communicate with a print server, the only information they normally need is:

1. The remote printer (`:rp`) value to be used in requests to the **lpd** print server. This is sometimes referred to as the *printer* or *print queue* name.
2. The remote server (`:rm`) which is the The IP address or hostname of the *print server*.
3. The original queue name specified by the user which may be used as part of the job information sent to the print server.

LPRng has several ways to specify the *printer queue* and *server* information.

5.2. Command Line -Pprinter@host

The `-P printer@host` literal explicitly provides a printer and server value. This is equivalent to the printcap entry shown below:

```
lpr -Plaser@10.0.0.1
  equivalent to printcap entry:
    laser:lp=laser@10.0.0.1
lpq -Plp@myserver
  equivalent to printcap entry:
```

```
lp:lp=lp@myserver
```

5.3. Command Line -Pprinter

Use the printcap entry with the name or alias `printer` and use the information in that printcap entry. Example:

```
lpr -Plp
  look up printcap entry for 'lp'
```

5.4. PRINTER, LPDEST, NPRINTER, and NGPRINTER Environment Variables

If no command line option is specified, the **LPRng** clients will use the first defined PRINTER, LPDEST, NPRINTER, or NGPRINTER environment variable value and will use the value as though specified as a `-P$PRINTER` command line literal. If the value has the form `-Pprinter@host` the print queue will be `printer` on server `host` and not consult the printcap database. If the value has the form `printer` then the printcap will be searched for a `printer` printcap entry. For example:

```
export PRINTER=laser@10.0.0.1; lpr
  equivalent to printcap entry:
    laser:lp=laser@10.0.0.1
export PRINTER=pr; lpr
  look up printcap entry for 'lp'
```

5.5. Wildcard Printcap Entry

If you specify a printcap name or alias as * (wildcard), then if the **LPRng** system cannot find a printcap with the exact name then the first matching wildcard will be used. For example:

```
pr|*:rm=server
  lpr -Ppr2 will match, and result in a printcap entry
    pr|pr2:rm=server

*|pr:rm=server
  lpr -Ppr2 will match, and result in a printcap entry
    pr2|pr:rm=server
```

5.6. First Printcap Entry

If you do not specify a printer on the command line or in the environment variables, and the `require_explicit_q` value is not set then **LPRng** will search the printcap and use the first valid printcap entry as the printer. It will use the first one found as the default.

5.7. Default Printer and Server Host

Options used:

- `default_remotehost`=*default rm (remote host)*
- `default_printer`=*default rp (remote printer)*

If the preceding steps do not yield a printer name and the `require_explicit_q` flag is not set then the `default_printer` and `default_remote_host` values from the `lpd.conf` configuration file will be used. If there is no configuration file, then the compile time defaults will be used.

Using the fallback values is usually not a desirable event and may indicate that you have a misconfigured host, so the compile time defaults are usually set to `missingprinter@localhost` to provide an annoying message for users.

5.8. Force Connection to Localhost

Options used:

- `force_localhost` FLAG *force localhost to be remote host*

The legacy BSD print spooler required an **lpd** print server to be running on each host. During the initial stages of development and deployment, the default **LPRng** configuration and deployment was to always allow *lightweight* operation, that is, clients would always connect to the remote host specified in the printcap.

While this default was appropriate for experienced system administrators, novice administrators or those who had already configured print spooling systems and simply wanted to upgrade found themselves confused by this change. This problem resulted in over 700 postings to the **LPRng** mailing list in a five year period.

This problem was solved by providing a `force_localhost` option in the configuration, and setting the default value to `1` or TRUE. When this option is TRUE, then all **LPRng** clients will connect to the server

on the localhost, *unless* they use the `lpr -Pserver@host` command line form. If lightweight operation is wanted, the administrator can either compile the **LPRng** software with the appropriate value or can explicitly set the `force_localhost@` flag.

```
# default:
lp:lp=lp@10.0.0.1:...
  lpr -Plp ->
    lp:lp=lp@localhost:...
  lpr -Plp@10.0.0.1 ->
    lp:lp=lp@10.0.0.1  (no other options)

lp:lp=lp@10.0.0.1:force_localhost@:...
  lpr -Plp ->
    lp:lp=lp@10.0.0.1:...

To disable at compile time:
  configure --disable-force_localhost
```

5.9. User Identification

Options used:

- `allow_user_setting=`*privileged users*

When an client program sends a command to the **lpd** server it may need to provide the name of the user who is originating the request for service. This name is obtained by looking up the UID of the user running the client in the appropriate user information database; if the information is not found the UID is used instead. Also, the client machine hostname may also be needed. This is usually determined by using a DNS lookup and trying to determine if there is a canonical or Fully Qualified Domain Name for the host and using this.

The `lpr -U name@host` (and for **lpq**, **lprm**, and **lpc**) option allows privileged users to cause the client software to use the `name` value as the originator and `host` as the machine name. This allows privileged users to *impersonate* other users. This is most useful for programs such as Samba and PCNFS, which need to act as proxies for users.

By default, ROOT (UID 0) is the only user that can masquerade as another user. The `allow_user_setting=name,name...` configuration option can be used to specify a list of names or UIDs that can also perform masquerading. For example, if the Samba server was running as user `samba`, then `allow_user_setting=samba` would allow it to specify the name of print job originator as a remote user, and the remote user would not need a login account on the system.

Chapter 6. lpr - Job Spooler Program

The **lpr** client program is used to submit a job to a print spooler. It does this by collecting information about the job, putting it in a control file, and then sending the control file and files to be printed to the print server. The **lpr** command line options are used to control or specify the values placed in the control file and how the job is to be transferred to the remote host. In addition, there are printcap or configuration level options that provide a further degree of administrative control over additional facilities. You can get the currently supported command line options by using the following command:

```
h4: {305} % lpr -=
lpr: Illegal option '='
 usage summary: lpr [-Pprinter[@host]] [-A] [-B] [-Cclass] [-Fformat]
   [-Jinfo] [-(K|#)copies] [-Q] [-Raccountname]  [-Ttitle]
   [-Uuser[@host]] [-V] [-Zoptions] [-b] [-m mailaddr] [-h]
   [-i indent] [-l] [-w num ] [-r] [-Ddebugopt ] [ filenames ...  ]
 -A          - use authentication specified by AUTH environment variable
 -B          - filter job before sending
 -C class    - job class
 -D debugopt - debugging flags
 -F format   - job format
   -b,-l        - binary or literal format
    c,d,f,g,l,m,p,t,v are also format options
 -J info     - banner and job information
 -K copies, -# copies   - number of copies
 -P printer[@host] - printer on host
 -Q          - put 'queuename' in control file
 -Raccntname - accounting information
 -T title    - title for 'pr' (-p) formatting
 -U username - override user name (restricted)
 -V          - Verbose information during spooling
 -Z options  - options to pass to filter
 -h          - no header or banner page
 -i indent   - indentation
 -m mailaddr - mail final status to mailaddr
 -r          - remove files after spooling
 -w width    - width to use
 PRINTER, NPRINTER environment variables set default printer.
```

If you are interested in the exact details of the job transfer, control file, and data file formats, please see RFC 1179 - Line Printer Daemon Protocol for the exact details.

6.1. Job Format Options

Options used:

- `default_format`=*default print job format (f)*

- `fx=`*supported formats for printing*

The legacy or vintage BSD print spooling system assigned each job a format. This format was used by the **lpd** server to select an appropriate filter program that would process the job and format it correctly for the printer. By convention, lower case letters were used to specify job formats.

The **LPRng lpr** client supports the legacy or vintage BSD formats, and also provides the `-Fx` literal to allow addition formats to be specified. If a format is specified the `default_format` value (usually `f`) is used. The `fx=...` option value is a list of the (lower case) characters corresponding to the formats allowed for a spool queue. For example, `fx=flv` would allow only jobs with format `f`, `l`, or `v` to be spooled to a queue. By default, all job formats are allowed.

A couple of job formats that require special treatment: the `b` (binary) and its alias the `l` (literal) format, and the `p` (pretty print) format. The `-b` or `-l` command line literal will select job format `l` (literal) for the job. By default, jobs marked with `l` format are supposed to have a minimum amount of handling and passed directly to a printer. The `p` (pretty print) format is just the opposite - these jobs are supposed to be pretty printed according to the various facilities available to the **lpd** print spooler.

6.2. Job Pretty Printing, Banners, Priority, and Accounting

Options used:

- `ab` FLAG *always print banner*
- `sh` FLAG *suppress header (banner)*

The legacy or vintage BSD pretty printing facility support allowed users to specify the title (`-T title`), indentation (`-i indent`), width (`-i indent`) of output for pretty printing. These probably will not have any effect, but their values are placed in the control file and sent to the **lpd** server.

The job name (`-J name`) and no banner page (`-h`) literals also cause information to be placed in the control file. The `-J` value is used to specify a job name on a banner page, and is placed in the control file. The `-h` (no header) has the strange effect of *not* putting *banner name* information in the control file. Apparently, if you do not have a name for your banner then no name will be sent.

If the `:sh` option is explicitly specified in a printcap entry for the **lpr** client, then it has the effect of the `-h` literal. However, the `:ab` (Always Print Banner) option can be used on the print server to always force a banner to be printed even if the user does not specify a banner.

Finally, the `-R` literal allows additional accounting information to be placed in the control file.

6.3. Job Class and Priority

Options used:

- `break_classname_priority_link` FLAG *classname does not set priority*
- `default_priority`=*default priority*
- `ignore_requested_user_priority` FLAG *ignore requested user priority*

The `-Cclass` information is used placed in the control file; the default class and priority is set by the `default_priority` option (default `A`). The first letter of the class information is uppercased and then used as the job priority.

The `break_classname_priority_link` option causes **lpr** to place class information in the control file but not to use it to set the job priority. This is usually used in institutions where users should not specify their priority. Finally, the `ignore_requested_user_priority` is actually used by **lpd** to ignore the priority requested by users.

6.4. Job Copies and Job Size

Options used:

- `mc`=*maximum number of copies*
- `mx`=*maximum job size (Kbytes)*

The `:mc` (maximum number of copies) and `:mx` (maximum job size) options are used by both the **lpr** and **lpd** programs. The `lpr -Kn` or `lpr -#n` option sets the numbers of copies wanted to `n`; if this is larger than the `:mc` value then the **lpr** client will not print the job and the **lpd** server will discard the job with an error. The job size is the product of the number of copies and the sum of the individual file sizes. This is, in effect, the total number of bytes to be transferred to the printer. If the job size is larger than the `:mx` limit, then the **lpr** client will not print the job and the **lpd** server will discard the job with an error.

6.5. Job Completion Notification Requested

Options used:

- `allow_user_logging` FLAG *users can send status*
- `lpr_bsd` FLAG *lpr -m acts as legacy BSD flag*

The `lpr -m mailaddr` option will put the specified mailaddr value into the control file. How this is processed is left to the particular server implementation. See the Abnormal Termination section for details.

LPRng extends the `lpr -m mailaddr` to allow mail addresses of the form `host[%port][/(TCP|UPD)]`, and when the job is being transferred or printed the **lpd** server to open a connection to the specified TCP/IP address and send job progress information. This is a security loophole and the `allow_user_logging` flag must be present to allow this operation.

The `lpr_bsd` will cause the `lpr -m` option not to take an argument, as in the BSD lpr, and will place the users name and host information in the control file as the destination for notification.

6.6. Remove Files After Spooling

The `lpr -r` option is extremely dangerous and has proven to be fatal to a large number of system administrators, so it should be used with caution. Unfortunately, it is used in a large number of shell scripts and by default in a vast number of programs so it is present in the **LPRng** software.

Its action is quite simple. After sending the print job to the spooler, it will try to unlink the original files. It will do the unlinking operation as the original user, but if the original user had permissions to remove, say, `/etc/password`, then the file will be removed.

6.7. The -Z Passthrough to Filter Options

Options used:

- `append_z`=*Append values to Z options list*
- `prefix_z`=*Prefix values to Z options list*
- `remove_z`=*Remove values from Z options list*
- `prefix_s_to_z` FLAG *Prefix control file S to Z information*
- `prefix_s_to_z` FLAG *Prefix control file Z to S information*

By convention all of the values specified with the `lpr -Z` option are placed in the job control file and passed to the **lpd** server, which turn passes them to the print filters. This allows users to pass options that are not part of the RFC1179 repetoire to filters.

In addition to this simple method of providing options, we can have the **lpr** and **lpd** systems add options to the control file information. The `prefix_z` and `append_z` options allow us to put Z options at the start or end of the user provided list; the `remove_z` removes specified options from the list.

The `prefix_s_to_z` and `prefix_z_to_s` options are usually used by the **lpd** server, and provides a simple way to have the SystemV `lp -o option` values put into the `Z` options value. This is very useful when existing SysV **lp** print spooling systems need to be interfaced to **LPRng** and the options need to be

handled in a meaningful manner. The `prefix_z_to_s` is similarly used when you want to forward jobs to a SysV spooling system and have the options passed correctly.

If present in the printer configuration, the various requested actions are are done by the **lpr** client, the **lpd** server on job reception, and the **lpd** server again on job processing. This redundancy ensures that jobs which are sent to this queue, arrive at this queue, or are processed by this queue have the options set appropriately.

6.8. Record Queue Name in Control File

Options used:

- `qq` FLAG *Insert queue name into control file*
- `force_queuename=`*Queuename to be used*

The `:qq` use queuename option tells **LPRng** to record the queue name that a job was originally queued to in the control file as the `Q` information. The `force_queuename=...` entry forces the queue name to be the specified value. For example, the following printcap entry and **lpr** commands will result in a filter being passed the arguments shown below.

```
#printcap
  pr1_landscape|pr1_portrait|pr_raw
   :filter=/.../filter
   :lp=/dev/pr
   :qq

lpr -Ppr1_landscape      lpr -Ppr1_portrait
  control file -
    Qpr1_landscape          Qpr1_portrait

filter command line:
  ... -Qpr1_landscape     ... -Qpr1_portrait
```

The `force_queuename` can be used for force a specific value into the control file `Q` information.

```
john|tom|frank
  :filter=/.../filter
  :lp=/dev/pr
  :force_queuename=office

lpr -Pjohn
  control file -
   Qoffice

filter command line:
```

```
... -Qoffice
```

6.9. Check For Nonprintable File

Options used:

- `check_for_nonprintable` FLAG ***lpr** checks for non-printable file*
- `ml`=*minimum number of printable characters*

By default **lpr** does not check files for for non-printable characters (i.e., escape characters) at the start of the print file. You can set the `check_for_nonprintable` flag to cause it to do so. The `:ml` value specifies the number of characters that are to be checked. Clearly, if it is 0, none will be checked.

6.10. Job Filtering By LPR

Options used:

- `lpr_bounce` FLAG *lpr does filtering*

Some users would like the advantages of the filtering and processing capabilities of a lpd daemon without running a **lpd** daemon on their system. By having the **lpr** program process the job by passing it through the various filters and then send the output of the filters as the print job you can get the desired effect.

```
# Simple example of an lpr_bounce entry
bounce
  :lpr_bounce
  :lp=lp@remote
  :filter=/usr/local/bin/lpf
```

The `lpr_bounce` flag, if present in the printcap entry, will force **lpr** to process the job using the specified filters and send the outputs of the filters to the remote printer for further processing.

6.11. Restrict Queue Use to Group Members

Options used:

- `rg`=*Restricted group list*

The `:rg` value specifies a list of groups. If this value is present use of a printer or operation is restricted to only users in a particular group. This check is done by both the **lpr** client and the **lpd** server if the option is present.

6.12. Fixing Bad Control Files and Metacharacters

Options used:

- `safe_chars`=*additional safe characters for control file*

RFC1179 defines a simple protocol and standard for print jobs to be interchanged between print spooling systems. Unfortunately, there were some major mistakes in not specifying the exact form that text would take when placed in the control file.

By default, **LPRng** will brutally convert a non-conforming RFC1179 control file into one that is acceptable to most, if not all, existing RFC1179 implementations. In order to prevent problems with **LPRng** ruthlessly purges all characters but upper and lower case letters, spaces, tabs, and `-_.@/:()=,+-%` from the control file, replacing suspicious characters with underscore (_). In addition, **LPRng** will ruthlessly regenerate control file entries and data file names so that they are compliant with all known RFC1179 implementations.

For some installations, the default set of safe characters may be overly restrictive. For example, *vintage* software may generate files with `#` characters in the `J` line of the control file. The replacement of this character by underscore (_) may cause other applications which use the control file information to stop working. The `:safe_chars` option allows the user to specify an additional set of safe characters in the `lpd.conf` configuration file(s).

For example, `:safe_chars=#"` would allow the `#` and `"` characters to appear in the control file.

6.13. Minimum Spool Queue Free Space

Options used:

- `minfree`=*Size in Kbytes*

If this value is non-zero, then the **lpd** receiving server checks to see that there is the specified number of Kbytes of file space available before accepting a job. If there is not enough space then the job is rejected.

6.14. FQDN Host Information

Options used:

- `force_fqdn_hostname` FLAG *Update control file with FQDN name*
- `force_ipaddr_hostname` FLAG *Update control file with IP address of remote host*
- `use_shorthost` FLAG *short control and data file names*

Some **lpr** clients do not put a FQDN host name in their control file. The `force_fqdn_hostname` flag will cause **lpd** to put a FQDN host name in the control file. This option will assume that the domain is where the connection originated from.

Similarly, the `force_ipaddr_hostname` flag will cause **lpd** to put a FQDN host name in the control file.

Some systems cannot handle FQDN hostnames in control and data file formats. The `use_shorthost` option will send control and data files with very short names to the remote print server.

Chapter 7. lpq - Status Monitoring Program

The **lpq** program is the main method used to monitor queues. You can obtain the available options by using:

```
h4: {306} % lpq -=
lpq: Illegal option '='
usage: lpq [-aAclV] [-Ddebuglevel] [-Pprinter] [-tsleeptime]
  -a           - all printers
  -c           - clear screen before update
  -l           - increase (lengthen) detailed status information
                 additional l flags add more detail.
  -L           - maximum detailed status information
  -n linecount - linecount lines of detailed status information
  -Ddebuglevel - debug level
  -Pprinter    - specify printer
  -s           - short (summary) format
  -tsleeptime  - sleeptime between updates
  -V           - print version information
```

7.1. lpq Queue Selection (lpq -Pprinter, lpq -a)

If no queue is specified, then the default queue is used, with `-Pprinter@host` causing a direct connection to the named host. The `-a` literal selects all queues.

7.2. lpq Job Selection

Jobs are selected by providing a job number, or `glob` pattern that is matched to the user name and and job identifier. If multiple patterns are provided each is applied in turn against the jobs in a queue until all have been tried. If any one of the patterns match then the job status is displayed.

If no selection information is provided or the `all` pattern is provided then all jobs in the queue are displayed.

7.3. lpq Short Format (lpq -s)

This is one line per spool queue:

```
% lpq -sa
t1@astart110  (printing disabled) 1 job
t2@astart110  (routed/bounce to t1@h10.private) 0 jobs
t3@astart110  (forwarding to t3a@h10.private)
t3a@astart110  (forwarding to t2@h10.private)
```

```
t4@astart110  (subservers t5, t6)  0 jobs
t5@astart110  (serving t4) 0 jobs
t6@astart110  (serving t4) 0 jobs
```

Note that the name of the printer/host is first, followed by optional status information, followed by the number of jobs. Only printcap entries with spool queues have a jobs word in the last position. The `-a` literal forces status for all queues or the queues in the `all` printcap entry to be returned.

The `stalled_time` (default 120 seconds) printcap option can be used to set a time after which active jobs will be reported as stalled.

7.4. lpq Long Format (lpq, lpq -l, lpq -L)

This is the default status display. It is a nicely formatted, extremely verbose format that is suitable for humble human interpretation. For example:

```
% lpq -a
Printer: t1@astart110  'Test Printer 1' (printing disabled)
 Queue: 1 printable job
 Server: no server active
 Status: finished operations at 09:44:00
 Rank   Owner/ID            Class Job  Files         Size Time
1       papowell@lprng110+202228663 A 10663 /tmp/hi     3 20:22:29
Printer: t2@astart110  'Test Printer 2' (routed/bounce to t1@h10.private)
 Queue: no printable jobs in queue
 Status: finished operations at 16:30:08
Printer: t3@astart110  (forwarding to t3a@h10.private)
Printer: t3a@astart110  (forwarding to t2@h10.private)
Printer: t4@astart110  (subservers t5, t6)
 Queue: no printable jobs in queue
 Status: finished operations at 09:44:06
Server Printer: t5@astart110  (serving t4)
 Queue: no printable jobs in queue
 Status: finished operations at 09:44:06
Server Printer: t6@astart110  (serving t4)
 Queue: no printable jobs in queue
 Status: finished operations at 09:10:00
```

The `lpq -l` (longer information) option causes more of the status information to be printed. You can use increasing numbers of `lpq -l` options (`lpq -ll` also works) to get more status. Use `lpq -L` for the maximum amount of status information.

7.5. lpq Verbose Format (lpq -v)

This uses an extension to the RFC1179 protocol, and is supported only by **LPRng**. The amount of information displayed is the brutal, and in effect does a total database dump of the LPD. This has been developed in order to provide diagnostic and status information for databases that need to keep track of job progress through a spool queue.

```
% lpq -v
Printer: t1@astart110
 Comment: Test Printer 1
 Printing: no
 Spooling: yes
 Queue: 1 printable job
 Server: no server active
 Status: accounting at end 'papowell@lprng110+094352860' at 09:44:00
 Status: printing 'papowell@lprng110+094352860', \
            closing device at 09:44:00
 Status: printing 'papowell@lprng110+094352860', finished  at 09:44:00
 Status: subserver status 'JSUCC' for 'papowell@lprng110+094352860' \
            on attempt 1 at 09:44:00
 Status: finished operations at 09:44:00
 Job: papowell@lprng110+202228663 status= 1
 Job: papowell@lprng110+202228663 CONTROL=
 - Hh10.private
 - Ppapowell
 - J/tmp/hi
 - CA
 - Lpapowell
 - Apapowell@lprng110+202228663
 - Qt1
 - fdfA010663h10.private
 - N/tmp/hi
 - UdfA010663h10.private
 Job: papowell@lprng110+202228663 HOLDFILE=
 - active 0
 - done 0
 - hold 0
 - move 0
....
```

7.6. Job Taking Too Long - Stalled

Options used:

- `stalled_time=`*seconds after which to report a stalled active job*

The `stalled_time` option is actually used by the **lpd** server to report that a job has been active more than the indicated time with no change in state. This is useful for spotting things such a printers that are out of paper, and so forth.

7.7. Configuring Format and Displayed Information

The following sections describe options that are used by the **lpd** server to control how it will return status information to a **lpq** request.

7.7.1. Display Class Information

Options used:

- `class_in_status` FLAG *show class name in status*

Setting the `class_in_status` option causes the class name rather than priority to be displayed in the status information.

7.7.2. Reverse Short and Long lpq Formats

Options used:

- `reverse_lpq_format=` FLAG *reverse **lpq** status format for specified remote systems*

Various Solaris and other System V implementations support an RFC1179 interface to remote printers. Unfortunately, there is a problem in that when they send a status request, the status format is reversed. That is, when LONG status format is wanted, they send SHORT, and vice versa.

The `reverse_lpq_format=` specifies a list of printers or IP addresses for which the **lpd** server will return LONG status when SHORT is requested, and vice versa. For example:

```
reverse_lpq_format=*.eng.com,130.192.0.0/16
```

will cause hosts whose Fully Qualified Domain Name (FQDN) ends in `eng.com` or from subnet `130.192.0.0` to have reversed status returned.

7.7.3. Status Line Length and Line Count

Options used:

- `return_short_status=`*return short **lpq** status for specified remote systems*
- `short_status_length=`*short **lpq** status length in lines*

In order to be compatible with non-**LPRng** client programs, some administrators would like **lpd** to return a short or brief status to normal status queries.

The `return_short_status=` specifies a list of printers or IP addresses for which the **lpd** server will return an abbreviated status when LONG status is requested. For example:

```
return_short_status=*.eng.com,130.192.0.0/16
short_status_length#3
```

will cause hosts whose Fully Qualified Domain Name (FQDN) ends in `eng.com` or from subnet `130.192.0.0` to get only 3 lines of detailed status returned.

7.7.4. lpq Status Format Determined by Requesting Host Address

Options used:

- `force_lpq_status=`*force **lpq** status format for specified remote systems*

In order to be compatible with non-**LPRng** client programs which are totally unpredictable, this allows the administrator to specify the format for **lpq** status when requests arrive.

The `force_lpq_status=` specifies a list of formats and printers or IP addresses for which the **lpd** server will return status in the specified format. The entry has the format `KEY=list;KEY=list...` where KEY is `s` for short and `l` for long format, and list is a list of hosts or IP addresses. For example:

```
force_lpq_status=s=pc*.eng.com,130.192.12.0/24,l=sun*.eng.com
```

will cause hosts whose Fully Qualified Domain Name (FQDN) matches `pc*eng.com` or from subnet `130.192.12.0` to get short status returned and hosts which match `sun*.eng.com` get long status.

Chapter 8. lprm - Job Removal Program

The **lprm** program is used to remove jobs from a print queue. You can obtain the available options by using:

```
h4: {307} % lprm -=
lprm: Illegal option '='
 usage: lprm [-A] [-a | -Pprinter] [-Ddebuglevel] (jobid|user|'all')*
  -a           - all printers
  -A           - use authentication
  -Pprinter    - printer (default PRINTER environment variable)
  -Uuser       - impersonate this user (root or privileged user only)
  -Ddebuglevel - debug level
  -V           - show version information
  user           removes user jobs
  all            removes all jobs
  jobid          removes job number jobid
 Example:
    'lprm -Plp 30' removes job 30 on printer lp
    'lprm -a'      removes all your jobs on all printers
    'lprm -a all'  removes all jobs on all printers
  Note: lprm removes only jobs for which you have removal permission
```

8.1. lprm Queue Selection (lprm -Pprinter, lprm -a)

If no queue is specified, then the default queue is used, with `-Pprinter@host` causing a direct connection to the named host. The `-a` literal selects all queues.

8.2. lprm Job Selection

Jobs are selected by providing a job number, or `glob` pattern that is matched to the user name and and job identifier. If multiple patterns are provided each is applied in turn against the jobs in a queue until all have been tried. If any one of the patterns match then the job status is displayed.

If no selection information is provided then the user name performing the request is used as the selection pattern, and only the first job that matches is removed.

The `all` pattern will match all jobs in the queue.

The user must have permission to control the queue or to remove a selected job in order for job removal to succeed.

Chapter 9. lpc - Administration Program

The **lpc** command is the main way that the **lpd** server is controlled. Here is the help information displayed by the command:

```
h4: {308} % lpc -=
lpc: Illegal option '='
usage: lpc [-Ddebuglevel][-Pprinter][-Shost][-Uusername][-V] [command]
 with no command, reads from stdin
  -Ddebuglevel - debug level
  -Pprinter    - printer or printer@host
  -Shost       - connect to lpd server on host
  -Uuser       - identify command as coming from user
  -V           - increase information verbosity
 commands:
 active  (printer[@host])        - check for active server
 abort   (printer[@host] | all)  - stop server
 class   printer[@host] (class | off) - show/set class printing
 disable (printer[@host] | all)  - disable queueing
 debug   (printer[@host] | all) debugparms - set debug level for printer
 down    (printer[@host] | all)  - disable printing and queueing
 enable  (printer[@host] | all)  - enable queueing
 hold    (printer[@host] | all) (name[@host] | job | all)* - hold job
 holdall (printer[@host] | all)  - hold all jobs on
 kill    (printer[@host] | all)  - stop and restart server
 lpd     (printer[@host]) - get LPD PID
 lpq     (printer[@host] | all) (name[@host] | job | all)*   - run lpq
 lprm    (printer[@host] | all) (name[@host]|host|job| all)* - run lprm
 msg printer message text- set status message
 move printer (user|jobid)* target - move jobs to new queue
 noholdall (printer[@host] | all)- hold all jobs off
 printcap(printer[@host] | all)  - report printcap values
 quit                            - exit LPC
 redirect(printer[@host] | all) (printer@host | off )*     - redirect jobs
 redo    (printer[@host] | all) (name[@host] | job | all)* - redo jobs
 release (printer[@host] | all) (name[@host] | job | all)* - release jobs
 reread  (printer[@host])        - LPD reread database information
 start   (printer[@host] | all)  - start printing
 status  (printer[@host] | all)  - status of printers
 stop    (printer[@host] | all)  - stop  printing
 topq    (printer[@host] | all) (name[@host] | job | all)*  - reorder job
 up      (printer[@host] | all) - enable printing and queueing
   diagnostic:
      defaultq              - show default queue for LPD server
      defaults              - show default configuration values
      client  (printer | all) - client config and printcap information
      server (printer | all) - server config and printcap
```

Most of the **lpc** command line options are common to all **LPRng** Clients, with the exception of the `-S server` literal. This option allows the **lpd** host to be explicitly specified.

The **lpc** commands can be classified as *informational*, *queue management*, *problem management*, *job scheduling*, and *diagnostic*.

9.1. Informational Commands - status, flush, active, reread

The `lpc status` command displays the current status of various activities of interest to the system administrator. This information includes the process ID of the server and other processes.

During normal operation, when requested for job status information the **lpd** server will create this information and then save it in a status cache. When successive requests for the same information arrive, the cache is checked to see if the information is already in the cache and there have been no status changes. If this is the case, the cached status information is used. The `lpc flush` command will flush (delete) this cache information and cause the **lpd** server to regenerate it from the original job files.

The `lpc active` command connects to the print server and gets the Process ID of the **lpd** process. This is useful to determine if the **lpd** server is running on the print server.

The `reread` command connects to the **lpd** print server and requests that the server reread the `printcap`, `lpd.conf`, and `lpd.perms` database files.

9.2. Queue Management - enable, disable, up, down

The `enable` and `disable` commands enable and disable *queuing* or sending jobs to the print queue. The `up` command combines the `enable` and `start` commands while the `down` command combines the `disable` and `stop` commands.

9.3. Printing Management - start, stop, up, down

These commands are used to start and stop printing. The `up` combines `start` and `enable`, while `down` combines `stop` and `disable`.

9.4. Problem Management - abort, redo, kill

These commands are usually used when there is a problem printing a job. The `abort` command is used to kill off all printing activity associated with a job. It also has the side effect of stopping printing on the print queue. The `redo` command will redo or attempt to reprint a job. The `kill` command combines the abort and redo command, and is useful when there are problems with the job currently being printed and it should be reprinted.

9.5. Job Scheduling - topq, holdall, noholdall, hold, release

The `topq` command effectively puts the select job or jobs at the top of the queue for printing.

The `holdall`, `noholdall`, and `release` commands implement a simple holding queue for jobs. By default, `holdall` is disabled. When the `holdall` command enables it, then jobs will remain in the spool queue until explicitly released with the `release` command.

The `hold` command can also be used to hold individual jobs until released.

9.6. Queue Management - class, redirect, move

The `class` command is used to restrict printing to jobs whose class value, set using the `lpr -C class` option, is in the class list or matches one of the classes. This allows the administrator to restrict printing. For more details and an example of its use, see Form Support.

The `redirect` command allows the administrator to accepts jobs at this queue and then to have them forwarded or redirected to another print queue. This is useful when a printer temporarily is unavailable.

The `move` command allows an individual job (or jobs) to be forwarded or redirected to another print queue.

Chapter 10. checkpc - Configuration Validation Utility

The **checkpc** (check printcap file) is one of the most useful utilities in the **LPRng** package.

It will read all the configuration files, printcap files and tests whether devices are set up correctly. Optionally, it will also set the permissions for spool directories and device files. Additionally, it will truncate the accounting and log files to a maximum size. Another use for **checkpc** is to remove old entries from queue directories.

For a new installation, you will want to run

```
checkpc -f -V
```

to set the permissions right. The `-f` flag instructs the program to correct file permissions. If you don't run this as `ROOT`, you'll receive a warning about that fact and any `chown(2)` calls will (most likely) fail.

The program reports everything it changes. Since it isn't too clever about some things (visit the man page), you should keep an eye on the output, and run it again if needed. If it keeps failing, change the permissions yourself.

10.1. Maintenance

Later, you will want to use **checkpc** for the daily maintenance of your system. I have this line in user `lp`'s crontab:

```
32 5 * * * checkpc -t 10K -A3 -r >/dev/null 2>&1
```

This job will:

1. truncate all log and accounting files to 10KB (`-t 10K`). Actually, it will keep the last 10K from the file, starting on a complete line.
2. remove all stale files older than three days (`-A3 -r`).

I'm redirecting output to `/dev/null`, because **checkpc** is a little noisy to my taste. (But too noisy is better than too silent :)

10.2. Printcap Information

You can use `checkpc -V -P` to examine printcaps and tell you what they contain. This is identical to the `lpc server all` operation, but with a higher level of verbosity.

Chapter 11. Printer Communication and Protocols

Common communication methods between a printer and a host system are network connections, parallel ports, or serial ports; while Fibre Channel, SCSI, USB, FireWire, InfraRed, and other interesting technologies have been used, they are either very specialized or not directly support by the **LPRng** software. In this section we will discuss Network, Parallel Port, and Serial Printers, as well as the different protocols and standards that apply to them.

11.1. Network Printers

The most flexible and highest throughput printer interface is via a network (TCP/IP) connection. Most high performance printers have a built in network interface, or you can attach them to a *printer server* box which provides a network interface. The network interface usually supports multiple network printing protocols. The most common are the LPD (RFC1179), Socket API, AppSocket, SMB, and Novell Netware interfaces. **LPRng** directly supports the LPD (RFC1179) and Socket API interfaces, and you can use the **smbclient** program from the Samba Software Package for the SMB interface.

11.2. RFC1179 (LPD) Connection

In this mode of operation the print server actually operates as a very limited BSD print spooler. These limitations include:

1. No error messages or status capability
2. Limited or very primitive banner printing. On some systems it may be *impossible* to turn banner printing off.
3. On most known print servers high connection activity caused by multiple systems attempting to get status or spool jobs may cause catastrophic failure of the printer.

For the above reasons, using RFC1179 to transfer jobs to a printer should be regarded as the least desirable option. Please see The RFC1189 Protocol for a detailed discussion of the RFC1179 protocol.

In order to use the RFC1179 transfer operation you must have a printcap entry for the printer that provides:

- The IP address or name of the printer that can be resolved to an IP address
- The name of the spool queue. In practice, this is usually used only to determine which of several printer ports on the print server the job will be sent to, or what type of processing the print server will do. Most cards usually do not do any processing and simply pass the job through to the printer.

The following is an example of a simple printcap entry that can be used to send a job to a remote printer using the RFC1179 protocol:

```
# LPRng syntax
# :lp value is 'where to print the job'
lp:
  :lp=raw@10.0.0.1

# OR Vintage BSD Print Spooler Syntax
# (LPRng supports this as well)
# :rp = remote printer, :rm = remote machine or host
lp:
  :rp=raw:rm=10.0.0.1
```

If you wish to transfer jobs to a print spooler without using the full **LPRng lpr** program, the Perl `lpr_in_perl` program in the **LPRng** Distribution UTILS directory can be used for testing and tutorial purposes.

11.3. Socket API

The Socket API is a very flexible job transfer protocol. It is widely support by most Print Server manufacturers, with the Hewlett Packard JetDirect setting the *de facto* standard. The Socket API is extremely simple.

1. The user establishes a connection to TCP/IP port on the Printer or Network Print spooler. The HP JetDirect uses port 9100 by default, but other ports are used as well. This connection may be refused if the printer is busy printing a job.
2. When the network connection is established to a system which has an *internal printer* or for which the Network Print Spooler is an integral part of the system, the printer usually flushes all internal buffers and readies itself to receive a new job. However, when you are using an external Print Server box, you may need to send specific initialization sequences to the printer to ensure that it is reset correctly and is ready to receive new jobs.
3. When the connection is made, all bytes sent to the connection are either transferred to and external interface to directly to a *print buffer* used by the printer's Print Engine.
4. The connection is bidirectional, and information sent to the external port by an external printer or error messages and status generated by the printer's Print Engine will be transferred over the data link to the user.
5. The Network Print spooler will keep the connection open until it is closed by the user. During this period it may continue to report status or other information such as printer On Line, paper outages, and so forth.
6. If the connection to the printer is *half-closed*, that is, the `shutdown()` network system call is used to indicate to the remote printer that no further data will be sent, then the printer may immediately

terminate the network connection. This means that no further network or status messages will be sent to the user.

7. If the connection is to a External Print Server, then usually the connection can be immediately re-established. It is the responsibility of the user to ensure that a the printer has finished its work before sending a new job.
8. If the connection is to an internal Print Server, then usually the printer will not allow the connection to be made, or will refuse all data transfers on the connection until the printer finishes with the previous job and all internal buffers have been cleared.

The following is a sample printcap showing how to use the Socket API:

```
lp:
  # make a socket connection to port 9100
  :lp=10.0.0.2%9100
```

You can use the netcat (http://www.l0pht.com/~weld/netcat/) utility by Hobbit <`Hobbit@avian.org`> to test that the Socket interface is available and working. If `ellipse.ps` is a test file, then: The simplest and easiest way to print a file to a network printer appears

```
  nc printer.ip.addr 9100 < file
Example:
  nc 10.0.0.25 9100 < ellipse.ps
```

11.4. AppSocket TCP/IP Protocol

The AppSocket interface is supported by Tektronix and some other printer vendors. It is similar to the Socket API, with a couple of significant differences.

1. The printer has two ports for network connections: a TCP port 9100 for TCP/IP stream connections and a UDP port for UDP packet connections.
2. When a 0 length UDP packet or a UDP packet containing only `CR/LF` is sent to UDP port 9101, the printer will return a packet to the sender containing print status information. This information indicates the printers current status (busy, idle, printing) and any error conditions.
3. The format, reliability, and repeatability of the UDP format and information is totally undocumented, and the UPD port facility should be regarded as, at best, an advisory function of the printer.
4. To send a job to the printer, a connection to TCP port is made. This connection will be refused while the printer is busy or has a connection to another host.

5. When the TCP connection is established, the information to be printed can be sent over the TCP connection. Bytes sent on this stream will be placed in the input buffer of the Print Engine and processed.
6. An end of job (EOJ) sequence indication in the data stream will cause the printer to terminate the connection. This is different than the Socket API, where the printer will keep the connection open. This means that if the PostScript CTRL-D (end of job) character is sent in a job, then the connection will be terminated.
7. Some models of printers modify this behavior slightly and will not terminate the connection, but will simply ignore any data following the EOJ indication.
8. Some printers support bidirectional AppSocket communication, and while the connection is open will return error indications or status information.
9. Once all the data has been received and the job has finished printing, the connection will be terminated by the printer.

The **ifhp** filter, one of the helper programs for **LPRng**, is used with **LPRng** to provide AppSocket support. For details, please see the [IFHP-HOWTO] in the **ifhp** Distribution (http://www.private/) and Tektronix P450 and Family for details. The following is a typical printcap entry for the AppSocket protocol. The actual network connection to the printer is made by the **ifhp** filter:

```
lp:
  # LPRng opens a dummy connection for consistency
  :lp=/dev/null
  # we pass the ifhp filter options indicating that the
  # Tektronics printer will need the appsocket protocol
  # and to use port 35 at 10.0.0.1 to make the connection
  # The ifhp filter may open and close the connection several
  # times during the file transfer in order to ensure that
  # the printer handles the job correctly.
  :ifhp=model=tek,appsocket,dev=10.0.0.1%35
  :filter=/usr/local/libexec/filters/ifhp
```

11.5. Network Print Server Boxes

A *network print server* is usually a box (external model) or card in a printer (internal model) which has a network connection to a TCP network and software to implement a LPD print server. If it is an external model, The parallel or serial port of the printer is connected to the box, and the print server may support multiple printers. If it is an internal model, the server is usually nothing more than a Network Interface Controller and a ROM containing software that the microprocessor in the printer uses.

The print server may support multiple printing protocols, such as RFC1179 (TCP/IP printing using the LPD print protocol), Novell Printer Protocols, SMB print protocols, and AppleTalk protocols. One of the observed problems with Network Print servers is that while they can usually support one protocol and one user at a time quite well, when you try to use multiple protocols and/or multiple users try to transfer

print jobs to the printer, the printer may behave in a very odd manner. Usually this results in a printer failing to finish a job currently being printed, and unable to accept new jobs.

Several of the newer models of print servers have Simple Network Management Protocol (SNMP) agents built into them, and can provide detailed information about their internal functions. By using a SNMP manager such as SunNetmanage or HP-Openview, you can monitor your network printers activities.

I recommend that you use only a single protocol to send jobs to the printer. If you can, I also recommend that you use a print spooler and have only a single host system send a job to the printer.

My best advice on connecting to network printers is not to use the the built-in LPD server, but to use the direct TCP/IP connection to the print engine. Usually this is done to particular TCP/IP port on the printer. For the HP JetDirect and other HP products, this is usually TCP port 9100.

Once you have the direct connection, you can now use various filters to preprocess the print job, insert PJL and PCL commands, or convert text to PostScript or PCL for better print quality.

11.6. Network Print Server Configuration Information

The following is a list of print server manufacturers, models, and with hints on how to access these boxes with various protocols.

Table 11-1. Network Print Server Configuration Information

Manufacturer	Model	RFC1179 Port Name (rp=XXX)	Send to TCP port
Cannon Printer (http://www.cannon.com/)	Cannon 460 PS, no hard drive	xjdirect	- Unknown if supported -
	Cannon 460 PS hard drive	`xjprint` - print immediately,`xjhold` - print later	- Unknown if supported -
Digital Products Inc. (http://www.digprod.com/)	NETPrint Print Server	`PORTn`, where `n` is port on server	- Unknown if supported -
Electronics For Imaging Inc. (http://www.efi.com/)	Fiery RIP i series	`normalq` or `urgentq`	- Unknown if supported -
	Fiery RIP XJ series	`xjprint`	- Unknown if supported -
	Fiery RIP XJ+ and SI series	`print_Model`, e.g. `print_DocuColor`	- Unknown if supported -
	Fiery models ZX2100, ZX3300, X2, X2e	`print`	- Unknown if supported -
Emulex Corp. (http://www.emulex.com/)	NETJet/NETQue print server	PASSTHRU	- Unknown if supported -

Manufacturer	Model	RFC1179 Port Name (rp=XXX)	Send to TCP port
Extended Systems Inc. (http://www.extendsys.com/)	ExtendNet Print Server	`Printer`*n*, where `n` is port on server	- Unknown if supported -
Hewlett-Packard (http://www.hp.com/)	JetDirect interface card	`raw`	9100
Hewlett-Packard (http://www.hp.com/)	JetDirect Multiport Server	`port 1 - raw1, port 2 - raw2, etc.`	port 1 - 9100, port 2 - 9101, etc.
I-Data (http://www.i-data.com/)	Easycom 10 Printserver	`par1` (parallel port 1)	- Unknown if supported -
	Easycom 100 Printserver	`LPDPRT1`	- Unknown if supported -
IBM (http://www.printers.ibm.com/)	Network Printer 12, 17, 24, and 24PS	PASS	- Unknown if supported -
Lantronix (http://www.lantronix.com/)	EPS1, EPS2	`EPS_X_S1 (serial) port 1, EPS_X_P1 (parallel) port 2`, etc.	3001 (port 1), 3002 (port 2), etc.
QMS (http://www.qms.com/)	Various Models	RAW	35 (AppSocket)
Tektronix (http://www.tek.com)	Tektronix printer network cards	PS (PostScript), PCL (PCL), or AUTO(Auto-selection between PS, PCL, or HPGL). Not reliable.	9100 (AppSocket on some models)
Rose Electronics (http://www.rosel.com)	Microserve Print Servers	lp	9100
Xerox (http://www.xerox.com/)	Models 4505, 4510, 4517, 4520	PASSTHRU	2501 (AppSocket on some models)
	Model 4512	`PORT1`	10001 (programmable)
	Model N17	RAW	9100
	Models N24 and N32	RAW	2000
	Models 4900, 4915, 4925, C55	PS	2000
	Document Centre DC220/230	`lp`	- Unknown if supported -

All company, brand, and product names are properties of their respective owners.

11.7. HP JetDirect Interface

The HP JetDirect Interface is one of the most widely used for network printers. For this reason it also has the most widely known set of problems. The user is strongly urged to upgrade to the latest version of firmware available for the unit. Problems with older versions of firmware include system lockups that require powerup level resets.

Newer versions of the HP JetDirect Interface have a Web Browser based configuration system. After you have assigned an IP address to the printer you can connect to the configuration port and configure the printer using the browser. If you run into configuration problems then you will most likely need to use the Microsoft Windows based JetDirect Configuration Software to reset or reconfigure the printer.

Older HPJetDirect cards can be configured only through through the front panel or through a set of network files. Here is a summary of the methods used from UNIX systems, or to use when you are desperate, to configure the printer.

11.7.1. Resetting To Factory Defaults

Most internal HP JetDirect print servers can be reset to factory defaults (or cold-reset) by turning the printer off and holding down the `Online` or `Go` button while turning the printer back on. The printer control panel display should read `Cold Reset`, `Restoring Factory Settings`, or something similar.

11.7.2. Setting Up IP Networking and Address

You can set the network address from the front panel. Reset the printer, put it in offline mode. and then use the MENU, +-, SELECT keys as follows:

```
MENU  -> MIO MENU (use MENU to display MIO MENU)
ITEM  -> CFG NETWORK=NO*
+     -> CFG NETWORK=YES
ENTER -> CFG NETWORK=YES*
ITEM  -> TCP/IP=OFF* (use ITEM to display TCP/IP)
+     -> TCP/IP=ON
ENTER -> TCP/IP=ON*
ITEM  -> CFG TCP/IP=NO* (use ITEM to display TCP/IP)
+     -> CFG TCP/IP=YES
ENTER -> CFG TCP/IP=YES*
ITEM  -> BOOTP=NO*
    (Enable BOOTP if you want to - see below)
ITEM  -> IP BYTE 1=0*
    This is IP address MSB byte.
    Use +- keys to change value, and then ENTER to change
    Use ITEM keys to get IP BYTE=2,3,4
ITEM  -> SM BYTE 1=255*
     This is the subnet mask value
    Use +- keys to change value, and then ENTER to change
    Use ITEM keys to get IP BYTE=2,3,4
ITEM  -> LG BYTE 1=255*
```

```
    This is the Syslog server (LoGger) IP address
    Use +- keys to change value, and then ENTER to change
    Use ITEM keys to get IP BYTE=2,3,4
 ITEM  -> GW BYTE 1=255*
    This is the subnet gateway (router) IP address
    Use +- keys to change value, and then ENTER to change
    Use ITEM keys to get IP BYTE=2,3,4
 ITEM  -> TIMEOUT=90
     This is the connection timeout value.  It puts a limit
    on time between connections.  A value of 10 is reasonable.
```

11.7.3. BOOTP Information

If you have a bootp server, you can put this information in the bootptab file. To use this, you must enable the bootp option on the printer. The T144 option specifies a file to be read from the bootp server. This file is read by using the TFTP protocol, and you must have a TFTPD server enabled. Here is a sample bootptab entry.

```
# Example /etc/bootptab: database for bootp server (/etc/bootpd).
# Blank lines and lines beginning with '#' are ignored.
#
# Legend:
#
#       first field -- hostname
#                         (may be full domain name)
#
#       hd -- home directory
#       bf -- bootfile
#       cs -- cookie servers
#       ds -- domain name servers
#       gw -- gateways
#       ha -- hardware address
#       ht -- hardware type
#       im -- impress servers
#       ip -- host IP address
#       lg -- log servers
#       lp -- LPR servers
#       ns -- IEN-116 name servers
#       rl -- resource location protocol servers
#       sm -- subnet mask
#       tc -- template host (points to similar host entry)
#       to -- time offset (seconds)
#       ts -- time servers
#
# Be careful about including backslashes where they're needed.
# Weird (bad) things can happen when a backslash is omitted
# where one is intended.
#
peripheral1:
```

```
:hn:ht=ether:vm=rfc1048:
:ha=08000903212F:
:ip=190.40.101.22:
:sm=255.255.255.0:
:gw=190.40.101.1:
:lg=190.40.101.3:
:T144="hpnp/peripheral1.cfg":
```

If you are using the T144 option, you will need to create the configuration file. The sample configuration file from the HP Direct distribution is included below.

```
#
# Example HP Network Peripheral Interface configuration file
#
# Comments begin with '#' and end at the end of the line.
# Blank lines are ignored.  Entries cannot span lines.

# Name is the peripheral (or node) name.  It is displayed on the
# peripheral's self-test page or configuration plot, and when sysName
# is obtained through SNMP.  This name can be provided in the BOOTP
# response or can be specified in the NPI configuration file to
# prevent the BOOTP response from overflowing the packet.  The domain
# portion of the name is not necessary because the peripheral does
# not perform Domain Name System (DNS) searches.  Name is limited to
# 64 characters.

name: picasso

# Location describes the physical location of the peripheral.  This
# is the value used by the interface for the MIB-II sysLocation
# object.  The default location is undefined.  Only printable ASCII
# characters are allowed.  Maximum length is 64 characters.

location: 1st floor, south wall

# Contact is the name of the person who administers or services the
# peripheral and may include how to contact this person.  It is
# limited to 64 characters.  This is the value used by the interface
# for the MIB-II sysContact object.  The default contact is undefined.
# Only printable ASCII characters are allowed.  Maximum length is 64
# characters.

contact: Phil, ext 1234

# The host access list contains the list of hosts or networks of
# hosts that are allowed to connect to the peripheral.  The format
# is "allow: netnum [mask]", where netnum is a network number or a
# host IP address.  Mask is an address mask of bits to apply to the
# network number and connecting host's IP address to verify access
# to the peripheral.  The mask usually matches the network or subnet
# mask, but this is not required.  If netnum is a host IP address,
```

```
# the mask 255.255.255.255 can be omitted.  Up to ten access list
# entries are permitted.

# to allow all of network 10 to access the peripheral:
allow: 10.0.0.0  255.0.0.0

# to allow a single host without specifying the mask:
allow: 15.1.2.3

# Idle timeout is the time (in seconds) after which an idle
# print data connection is closed.  A value of zero disables
# the timeout mechanism.  The default timeout is 90 seconds.

idle-timeout: 120

# A community name is a password that allows SNMP access to MIB
# values on the network peripheral.
# Community names are not highly secure;
# they are not encrypted across the network.  The get community name
# determines which SNMP GetRequests are responded to.  By default,
# the network peripheral responds to all GetRequests.  The get
# community name is limited to 32 characters.

#
# For hpnpstat and hpnpadmin, the community name can be stored in
# /usr/lib/hpnp/hpnpsnmp.

get-community-name: blue

# The set community name is similar to the get community name.  The
# set community name determines which SNMP SetRequests are responded
# to.  In addition, SetRequests are only honored if the sending host
# is on the host access list.  By default, the network peripheral
# does not respond to any SetRequests.  The set community name is
# limited to 32 characters.
#
# The set community name can come from /usr/lib/hpnp/hpnpsnmp if it
# is the same as the get community name.  We recommend that the set
# community name be different from the get community name though.

set-community-name: yellow

# SNMP traps are asynchronous notifications of some event
# that has occurred.
# SNMP traps are useful only with network management software.
# Traps are sent to specific hosts and include a trap community
# name.  Up to four hosts can be sent SNMP traps.
# The trap community name is limited to
# 32 characters.  The default name is public.

trap-community-name: red

# The SNMP trap destination list specifies systems to which SNMP
```

```
# traps are sent.  Up to four IP addresses are allowed.  If no
# trap destinations are listed, traps are not sent.

trap-dest: 15.1.2.3
trap-dest: 15.2.3.4

# The SNMP authentication trap parameter enables or disables the
# sending of SNMP authentication traps.  Authentication traps indicate
# that an SNMP request was received and the community name check
# failed.  By default, the parameter is off.

authentication-trap: on

# The syslog-facility parameter sets the source facility identifier
# that the card uses when issuing syslog messages.  Other facilities,
# for example, include the kernel (LOG_KERN), the mail system
# (LOG_MAIL), and the spooling system (LOG_LPR).  The card only allows
# its syslog facility to be configured to one of the local user values
# (LOG_LOCAL0 through LOG_LOCAL7).  The selectible option strings,
# local0 through local7 (configured to LOG_LOCAL0 through LOG_LOCAL7,
# respectively) are case insensitive.  The default syslog-facility
# for the card is LOG_LPR.

syslog-facility: local2

# This parameter allows the card to treat hosts on other subnets as
# if the hosts were on the card's subnet.  This parameter determines
# the TCP Maximum Segment Size (MSS) advertised by the card to hosts
# on other subnets and affects the card's initial receive-window
# size.  The card will use a TCP MSS of 1460 bytes for local hosts,
# and 536 bytes for a non-local host.  The default is off, that is,
# the card will use the maximum packet sizes only on the card's
# configured subnet.
#
# The configuration utility does not allow access to this parameter.
# If you want to configure it, you must manually edit the NPI
# configuration file and add it to the bottom of the entry for the
# network peripheral.

subnets-local: on

# This parameter affects how the card handles TCP connection requests
# from the host.  By default, the JetDirect MPS card will accept a
# TCP connection even if the peripheral is off-line.  If this parameter
# is set to "on", then the card will only accept a TCP connection
# when the peripheral is on-line.

old-idle-mode: off
```

11.7.4. Telnet Configuration

You can telnet to the JetDirect interface and use the command line mode. There is a very simple help facility available that you can invoke by entering `?` at the prompt.

Different versions of HP JetDirect cards have different commands. You will have to experiment to find out which ones are supported.

11.7.5. Disabling Banner Page Generation

You can do this by making a telnet connection to the JetDirect interface and use the command line mode. There is a very simple help facility available that you can invoke by entering `?` at the prompt.

The configuration command `banner: 0` will disable banners. You may need to save the configuration after you have changed it and then do a power up reset of the printer to have it take effect.

11.8. Problems With Network Print Servers

Most of the Network Print Servers are implemented using extremely simply software. The following is a list of some problems and what options the **LPRng** software uses to handle them.

11.8.1. Network Print Server Not Responding

Only a single TCP/IP connection is accepted at a time. This means that when one user is sending a job then the unit will not accept other connections. There is another side effect of this problem, which is that some implementations will accept a network connection but not read any data from the connection until the previous connection is finished.

The deal with these problems the `connect_timeout`, `send_job_rw_timeout`, and `send_query_rw_timeout` are used to control job transfer and lpq status gathering. See Printing Job Files and Opening the Output Device for details.

11.8.2. Network Print Server Does Not Handle LPQ, LPRM

Some Network Print Servers do not respond to `lpq` or `lprm` queries correctly. The `remote_support` option can be used to solve this problem by specifying what operations the remote print server can handle:

```
:remote_support=RMQVC
  R = lpr, M = lprmg, Q = lpq, V = lpq -v, C = lpc
  :remote_support=R  # printer only handles LPR
```

11.8.3. Incomplete Job Transfers

This is the result of a defective or buggy TCP/IP stacks. A common problem is the habit that some Network Print Servers occaisionally discard data at the end of a print job when a network connection is *half-closed*. A *half-closed* connection is one where one end of the sending connection indicates that no further data will be sent. Unfortunately, the Network Print Server will then try to close the connection in the other direction. When this does not immediately succeed, it will terminate the network connection, discarding any unprinted data.

The `half_close` flag can be used to solve this problem. See Normal Termination for more details.

```
lp:lp=lp@remote        # shutdown(fd,WRITE) connection, wait for end
lp:lp=lp@remote:half_close@    # close() connection and do not wait
```

11.9. Printing to a SMB (MicroSoft) Printer

Microsoft use the SMB (Simple Message Block) protocol to transfer files and print jobs to hosts and printers. SMB can be used over TCP/IP, NetBEUI, IPX, and other lower level network protocols.

Unfortunately, most printers do not provide detailed status or error reports when using the SMB protocol. There are a very large number of printers that have deficient SMB support that causes problems when used in a high traffic or high throughput environment.

It is highly recommended that this protocol not be used unless there is no alternative.

If you have a printer or a remote print spooler that supports SMB You can use the SAMBA **smbclient** program to send a print job to an SMB client. The following is a sample Shell Script script which you can use:

```
#!/bin/sh -x
# This script is an input filter for printing on a unix machine. It
# uses the smbclient program to print the file to the specified smb-based
# server and service.
# The 'smb' printcap entry below shows how to configure LPRng
# for printing
#
# smb:
#  :lp=|/usr/local/samba/smbprint
#  :sd=/var/spool/smb:
#  :filter= ... filter ...
#
# The /var/spool/smb/.config file should contain:
#   server="PC_SERVER"
#   service="PR_SHARENAME"
#   password="PASSWORD"
#
# Set PC_SERVER to the server, PR_SHARENAME to the printer,
# and PASSWORD to the password for this service.
```

```
#
# E.g.
#   server=PAULS_PC
#   service=CJET_371
#   password=""
#
#
config_file=.config
if [ -f $config_file ] ; then
    eval `/bin/cat $config_file`
fi
#
# NOTE You may wish to add the line 'echo translate'
# if you want automatic
# CR/LF translation when printing.
(
#   echo translate
    echo "print -"
    /bin/cat
) | /usr/local/bin/smbclient "\\\\$server\\$service" \
   "$password" -U "$server" -N -P 1>&2
```

If the above script was in `/usr/local/libexec/filters/smbprint`, the printcap entry for this printer would be:

```
pauls_pc:
  :sd=/var/spool/lpd/%P
  # we filter the output
  :lp=|/usr/local/libexec/filters/smbprint
  # you can add filters if you want to do specific actions
  :ifhp=model=hp4
  :filter=/usr/local/libexec/filters/ifhp
```

11.10. Printing to AppleTalk Printers

The netatalk package comes with the **pap** program that can be used to transfer jobs using the AppleTalk protocol. A printcap entry for a network printer looks like the following:

```
atalk:
  :lp=| -$ /usr/local/atalk/bin/pap -e -p "npbname"
  :sd=/var/spool/lpd/atalk
  :ifhp=model=ps,status@
  :filter=/usr/local/libexec/filters/ifhp
```

The -$ suppress the addition of extra parameters to the **pap** command line. The **pap** program must be SETUID root and executable only by root or group daemon. This can be done by using the following script:

```
h4: {309} cd /usr/local/atalk/bin
h4: {310} chown root pap
h4: {311} chgrp daemon pap
h4: {312} chmod 550 pap
h4: {313} chmod s+u pap
```

11.11. Parallel Port Printers

When installing a parallel port printer, there are three elements of concern: the physical hardware connection (cable and connectors), the IO device and Operating System support, and finally the print spooler support for the device.

Originally most parallel port devices were strictly *write only* with a minimum amount of error information - only hardware signals for *online*, *out of paper* and *fault* were present. Due to the lack of any other way to interface devices to the Intel based PC platform, desperate hardware designers in search of a cheap (free?) way to interface their IO devices developed ways to manipulate the signals and do *bidirectional* communication. Needless to say, no two companies developed the same methods, and no two companies were compatible with any other companies method.

In 1994, the IEEE 1284 standard was first developed,. The following information is courtesy of Warp Nine Engineering, of San Diego, CA, from the http://www.fapo.com/1284int.htm web page.

When IBM introduced the PC, in 1981, the parallel printer port was included as an alternative to the slower serial port as a means for driving the latest high performance dot matrix printers. The parallel port had the capability to transfer 8 bits of data at time whereas the serial port transmitted one bit at a time. When the PC was introduced, dot matrix printers were the main peripheral that used the parallel port. As technology progressed and the need for greater external connectivity increased, the parallel port became the means by which you could connect higher performance peripherals. These peripherals now range from printer sharing devices, portable disk drives and tape backup to local area network adapters and CD ROM players.

The problems faced by developers and customers of these peripherals fall into three categories. First, although the performance of the PC has increased dramatically, there has been virtually no change in the parallel port performance or architecture. The maximum data transfer rate achievable with this architecture is around 150 kilobytes per second and is extremely software intensive. Second, there is no standard for the electrical interface. This causes many problems when attempting to guarantee operation across various platforms. Finally, the lack of design standards forced a distance limitation of only 6 feet for external cables.

In 1991 there was a meeting of printer manufacturers to start discussions on developing a new standard for the intelligent control of printers over a network. These manufacturers, which included Lexmark, IBM, Texas Instruments and others, formed the Network Printing Alliance. The NPA defined a set of parameters that, when implemented in the printer and host, will allow for the complete control of printer applications and jobs. While this work was in progress it became apparent that to fully implement this

standard would require a high performance bi-directional connection to the PC. The usual means of connection, the ordinary PC parallel port, did not have the capabilities required to meet the full requirements or abilities of this standard.

The NPA submitted a proposal to the IEEE for the creation of a committee to develop a new standard for a high speed bi-directional parallel port for the PC. It was a requirement that this new standard would remain fully compatible with the original parallel port software and peripherals, but would increase the data rate capability to greater than 1M bytes per second, both in and out of the computer. This committee became the IEEE 1284 committee. The IEEE 1284 standard, "Standard Signaling Method for a Bi-directional Parallel Peripheral Interface for Personal Computers", was approved for final release in March of 1994.

Even if your hardware has support for the IEEE 1284 high speed bidirectional data transfers, your Operating System drivers must support it. Unfortunately, there is no universal agreement on the capabilities that should be provided by the low level printer port device drivers, and the method for supporting them. A good example of the problem of OS support and drivers is given by the Linux Parallel Port group, currently headed by Tim Waugh, and which is documented in the http://people.redhat.com/twaugh/parport/ and http://people.redhat.com/twaugh/parport/html/parportguide.html web pages.

Given this state of affairs, it should be no surprise that there is no support for bidirectional parallel port printers in **LPRng**. In fact, it turns out that there are severe problems with many Unix implementations that cause extreme headaches. These include:

- While a parallel port may be opened Read/Write, a `read()` will either block indefinitely or cause a major system failure (crash).
- There is no buffering of data in the low level driver; that is, once a `write()` starts, then the process will block until the data is written out. If the `write()` call is interrupted by a *signal* and not immediately restarted, then the status returned by the `write()` may be an error indication (`-1`) or the numbers of bytes that were actually written. If an error is returned, then an unknown number were written and the print job must be aborted.
- Many Operating systems do not implement a `select()` functionality for the parallel port, which means that it is difficult to do a multi-threaded implementation. Instead, a polling method must be used.

The good news is that on all known systems, if the parallel port device is opened exclusively for writing, and a blocking `write()` is used, and the `write()` is not interrupted, and there are no device errors, then data is delivered correctly to the device.

In most UNIX systems the printer port has the name `/dev/lpt`, `/dev/prn`, or something similar. On most systems the **dmesg** utility will print a list of IO devices found during system configuration. Use the following commands to get the information and scan for the device. You should also make sure that the printer device is available.

```
dmesg >/tmp/a
grep lp /tmp/a
ls /dev/lp*
```

Gordon Haverland <haverlan@agric.gov.ab.ca> supplied this little script, that will assist with this:

```
#!/bin/sh
#set -v -x              # uncomment for debugging
PATH=/bin:/usr/bin
printer=
for printer in /dev/lp* ;
do
    echo PRINTER TEST to $printer 1>&2
    for i in 1 2 3 4 5 6 7 8 9;
    do
        echo PRINTER $printer $i > $printer;
    done
    echo -e \\r\\f > $printer
done
exit 0;
```

If your printer is connected to the device name you provided, then you should get a page of something out. If the output suffers from the *staircase* effect, you will see the numbers marching across the page, otherwise the numbers will all be in a single column.

11.12. Serial Printers

If your printer is attached by a serial line, then you may need to set the serial line characteristics before sending the job to the printer. Here are a set of guidelines to following when attaching a serial port printer to a serial line.

1. Check to make sure that the line is not enabled for login. Logins are usually managed by the **getty** (BSD) or **ttymon** (Solaris, SystemV). Check your system documentation and make sure that these daemons are not managing the serial line.

2. Check the permissions and ownership of the serial line. For the most easy testing, set the permissions to 0666 (everybody can open for reading and writing). After you have made sure that you can send jobs to the printer, you might want to change the ownership of the serial line to the **lpd** server and change the permissions to 0600.

3. Make sure that you can print a test file on the printer via the serial port. This may require setting the line characteristics and then sending a file to the printer. You should try to use 8 bit, no parity, with hardware flow control and no special character interpretation, and definitely no LF to CR/LF translation. The problem is that different versions of UNIX systems have different sets of stty(1) commands to do this. The following simple test script can help in this.

```
#!/bin/sh
# 9600, no echo, no CR
FLAGS= 9600 -raw -parenb cs8 crtscts
DEV= /dev/tty01
(stty $FLAGS; stty 1>&2; cat $1 ) <$DEV >$DEV
```

This shows using stty to set the flags, then to print the current settings, and then using cat a file to the output. If you attach a dumb terminal to the serial port, you can even use this script to ensure that input from the device is echoed to the output with the correct speed, parity, etc.

Experience has shown that serially connected printers are the least reliable and lowest speed. Where possible, it is strongly recommended that they be attached to a *network print box* which will provide a Socket API interface and handle the low level network to serial port protocol conversions.

Chapter 12. Printcap Database

As described in the Print Spooling Overview, the heart of the **LPRng** system is information in the `printcap` file. The printcap information specifies:

1. The print queues available to users.
2. How client programs communicate with the **lpc** print server.
3. The configuration, location, and other information for each print queue on the print server.
4. How the **lpd** server processes jobs in each print queue.

In order to explain a complex subject, we will start with a set of simple printer configurations, and explain the purpose and effect of each entry in the printcap.

For details about individual printcap options, see the printcap(5) man page from the **LPRng** distribution, or use the Index To All The Configuration and Printcap Options to find a specific printcap option and its effects.

12.1. The Printcap Parsing Rules

Options used:

- `client` FLAG *printcap entry valid only for client programs*
- `oh=` *hosts where printcap entry valid*
- `server` *printcap entry valid only for lpd server*
- `tc` *add named printcap entry contents*

In this section, we will discuss the remaining tricky parts of the **LPRng** printcap database: combined client and server printcaps, host specific printcap entries, and the `tc` *include* facility.

The following is a complete description of how a printcap file is processed:

1. When processing a printcap file, the **LPRng** software reads and parses each entry individually. Leading whitespace is removed. Lines starting with `#` and blank lines are ignored.
2. Lines ending with `\` will have the `\` discarded, and all lines of a printcap entry are joined by removing the line separators (`\n`) and replacing them with a space.
3. The printcap entry is parsed, and the printcap name, aliases, and options are determined. Colons `:` act as option separators, and leading and trailing whitespaces are removed.
4. Options are sorted and except for the `tc=...` option only the last option setting is retained.

5. If an option value requires a colon, then the `\:` or `\072`, the same escaped character value as used in the C, Perl, tcl, and other programming languages, can be used.
6. Client programs will discard a printcap entry with a `server` option and server programs will discard a printcap entry with a `client` options.
7. The `oh` (*o*n this *h*ost) option specifies a list of IP addresses and mask pairs or glob strings which are used to determine if this printcap entry is valid for this host (see discussion below).
8. After the above processing, if there is an existing termcap entry with the same name, the two sets of options are combined, with the last option setting retained except for the `tc` entries which are combined.
9. When a printcap entry is actually used, the printcap entries listed by the `tc` include option are extracted and combined in order. This allows include entries to appear after the referring printcap entry. Then printcap options will be combined with the included ones. This has the effect that the options specified in the printcap entry will override the ones from the `tc` included entries.
10. Finally, each string printcap option with a *%X* value has *%X* replaced by the following values. Unspecified values will not be modified.

Key	Value	Purpose
`%P`	printcap entry primary name	
`%Q`	queue requested	
`%h`	short host name (host)	
`%H`	fully qualified host name (host.dns.whatever)	
`%R`	remote printer (rp value)	
`%M`	remote host (rm value)	
`%D`	date in YYYY-MM-DD format	

11. When parsing multiple printcap files, these are processed in order, and all of their printcap entries are combined according to the above procedures. The `tc` resolution and *%X* expansion is done after all the files have been processed.

The following examples show how to use the above rules to your advantage. You can combine both client and server printcap information in a single file as well as dividing a printcap entry into several parts. Here is an example:

```
# seen by both client and server
lp1:lp=lp@pr1:mx=100
lp1:sd=/usr/local/spool/lp1:mx=0
# seen only by client
lp2:lp=lp@pr2:client
# seen only by server
lp2:lp=/dev/lp:server
```

1. Printcap entries with the same name are combined. The first printcap entry, `lp1`, the information is seen by both client and server. The next printcap entry, with the same name `lp1`, will be combined with the first one. The order of options is important - the entries are scanned in order and an option will have the last value set. Thus, after having read both the `lp1` printcap entries, both client and server will have:

   ```
   lp1:lp=lp@pr1
     :mx=0
     :sd=/usr/local/spool/lp1
   ```

2. The `lp2` has a client and server version. This is recommended when complex printcaps on multiple hosts and servers are used. Thus, the **LPRng** clients will see:

   ```
   lp1
     :lp=lp@pr1
     :mx=0
     :sd=/usr/local/spool/lp1
   lp2
     :client
     :lp=lp@pr2
   ```

   and the server will see:

   ```
   lp1
     :lp=lp@pr1
     :mx=0
     :sd=/usr/local/spool/lp1
   lp2
     :lp=/dev/lp
     :server
   ```

If you have multiple printers of the same type whose configuration is almost identical, then you can define a set of *tc* only printcap entries containing common information and use the `tc` include facility.

Printcap entry names may start with period (`.`), question mark (`?`), or exclamation mark (`!`), followed by one or more alphanumeric , underscore (_) or hyphen (`-`) characters. Queue or printer names start with an alphanumeric character. Printcap entries whose names do not start with an alphanumeric character can only be used as targets of the `tc` include facility. For example:

```
.hp:
  :sd=/usr/local/spool/%P
  :mx=0
hp1:tc=.hp,.filter
  :lp=lp@10.0.0.1
hp2:tc=.hp,.filter
  :lp=lp@10.0.0.2
.filter
  :filter=/usr/local/libexec/filters/ifhp
```

1. The `.hp` and `.filter` printcap entities are not spool queue definitions. After `tc` include processing is completed, the printcap information would resemble:

```
hp1
  :lp=lp@10.0.0.1
  :filter=/usr/local/libexec/filters/ifhp
  :mx=0
  :sd=/usr/local/spool/%P
hp2
  :lp=lp@10.0.0.2
  :filter=/usr/local/libexec/filters/ifhp
  :mx=0
  :sd=/usr/local/spool/%P
```

2. The `%X` processing will replace `%P` with the printcap name, so we would have:

```
hp1
  :lp=lp@10.0.0.1
  :filter=/usr/local/libexec/filters/ifhp
  :mx=0
  :sd=/usr/local/spool/hp1
hp2
  :lp=lp@10.0.0.2
  :filter=/usr/local/libexec/filters/ifhp
  :mx=0
  :sd=/usr/local/spool/hp2
```

12.2. Simple Client Printcap Entry

Options used:

- `client` FLAG *client printcap entry*
- `lp`=*destination printer information*
- `rm`=*remote host (machine)*
- `rp`=*remote printer*

I'll use this simple example to explain the basics of the **LPRng** printcap format and introduce some of the **LPRng** network configuration options. Here is a simple printcap file used to provide client programs (`lpr, lprm,` etc) with *remote printer* and *server* information.

```
# printer lp1
lp1|printer1
  :rm=localhost
```

```
# printer lp2 with continuation
lp2:\
  :lp=pr@10.0.0.1:client
# printcap lp3, to printer pr, with overrides
lp3:rp=pr:rm=hp.private
  :force_localhost@
# Simplest possible printcap entry - defaults for everything
lp4
```

1. Lines starting with a `#` sign are comments, and all leading and trailing *whitespace*, i.e. - spaces, tabs, etc, are ignored. Empty lines are ignored as well.
2. A printcap entry starts with the printcap entry *name*, followed by one or more *aliases*, followed by one or more options. In the above example we have three printcap entries: `lp1` with an alias `printer1` and `lp2`, `lp3`, and `lp4` with no aliases.
3. Aliases start with the `|` character and options with the `:` character; tabs and spaces before and after the `|` or `:` characters and at the start and end of lines are ignored. You can use backslash (`\`) at the end of a line to create a multi-line value for an option. The backslash will cause the next line to be appended to the current line; watch out for comments and ends of printcap entries if you use this facility. As you can see from the example, there is no `Name` printcap entry - this is part of the `cm` option on the previous line.
4. Options take the form of a keyword/value pair, i.e.-

   :option=value
   :option#value (legacy, not advised for new systems)
   :option
   :option@

5. Option names are case insensitive, but option values are not. While `Ts` and `ts` are the same option name, `ts=Testing` and `ts=testing` have their case preserved. A string or integer value is specified by `option=value` or `option#value`.
6. The use of the legacy `option#value` form is NOT recommended as some preprocessors and database systems will treat # as the start of a comment and delete the remainder of the line. This has caused great consternation for sysadmins who wonder why their NIS distributed printcap entries have been mysteriously truncated.
7. If you want to set a string option to *empty* value, use `option=`. The `option` will set it to `1`. If an option value contains a colon, then use the C (or Perl or Tck/Tk) string escape `\072` to represent the value.
8. Boolean options are set TRUE (1) if no value follows the keyword and FALSE (0) by appending a `@`. For example `sh` will set `sh` to TRUE and `sh@` to FALSE.

There may be multiple options on the same line, separated by colons.

Now let's examine the first printcap entry in detail. It is reproduced here for convenience:

```
# printer lp1
lp1|printer1
```

```
  :rm=localhost
```

1. We start with a comment, followed by the printcap entry name and and alias. Aliases are useful when you want to refer to a single printer or print queue by different names. This can be useful in advanced printcap and print queue setups. By default, the remote printer name is the printcap entry name.
2. The `rm` (remote machine or host) option specifies the name or IP address of the **lpd** host running **lpd**. In this example the remote host is `localhost` or the machine that the client is running on and we assume that the **lpd** server is running on the localhost. Thus, we would communicate with printer `lp1@localhost`.

Let's look at the next printcap entry:

```
# printer lp2 with continuation
lp2:\
  :lp=pr@10.0.0.1:client
```

1. The `lp2` printcap entry illustrates the use (and abuse) of the `\` continuation. If you think about this, we have really defined a printcap entry of the form:

   ```
   lp2: :lp=pr@10.0.0.1:client
   ```

   Luckily, **LPRng** ignores empty options like `::`. While it is strongly recommended that `\` be avoided it may be necessary for compatibility with other system utilities.
2. The `lp=pr@10.0.0.1` literal is an alternate way to specify a remote queue and server. If the `force_localhost` default is being used, then the **LPRng** clients will ignore the `10.0.0.1` address and still connect to `pr@localhost`. There is further discussion about this in the next section.
3. The `client` option explicitly labels client only printcap information. The **lpd** server will ignore any printcap with the `client` option. When constructing complex printcaps, this option is used to keep ensure that you have consistent printcap information.

The following printcap entry shows how to override the `force_localhost` default, and force the **LPRng** clients to connect directly to a remote server:

```
lp3:rp=pr:rm=hp.private
  :force_localhost@
```

1. The `rp=` (remote printer) remote print queue name to used when sending commands to the **lpd** print server.
2. The `force_localhost@` literal is an example of a *flag* option. The `@` sets the literal value to 0 (false). We set `force_localhost` to false, which now allows the **LPRng** clients to connect directly

to the specified remote printer. In this example, the `hp.private` could be a HP LaserJet Printer with a JetDirect interface, which supports the RFC1179 protocol.

3. One disadvantages of sending a job directly to a printer using the above method is that **lpr** program will not terminate or exit until all of the files have been transferred to the printer, and this may take a long time as the printer processes the files as they are received.

Now let's look at the last printcap entry:

```
# Simplest possible printcap entry - defaults for everything
lp4
```

The last example is the simplest possible printcap entry. This will cause **LPRng** clients to use the default values for everything. The printer will be `lp4`, i.e. - the name of the printcap, and the server will be `localhost` if `force_localhost` is set, or the value of the `default_remote_host` configuration option if it is not.

12.3. Simple Server Printcap Example

Options used:

- `cm`=*comment for status*
- `filter`=*job filter*
- `lf`=*log file*
- `af`=*accounting file*
- `lp`=*output device*
- `mx`=*maximum job size*
- `sd`=*spool directory file*

The previous section discussed printcap entries for use by the client programs. Now we will discuss printcap entries for use by the **lpd** server. In simple configurations or when we have the `force_localhost` option enabled we can use the same printcap for both **LPRng** clients and the **lpd** server.

```
# Local ASCII printer
lp1|printer
  :server
  :cm=Dumb printer
  :lp=/dev/lp1
  :sd=/var/spool/lpd/lp1
  :lf=log:af=acct
```

```
:filter=/usr/local/libexec/filters/ifhp
:mx=0
```

1. The printcap entry name is `lp1`. This information will be displayed when requesting status information using the **lpq** program.
2. The `printer` alias. This allows a single spool queue to have multiple names.
3. The `:server` option specifies this printcap entry is only used by **lpd** server.
4. The `:cm` field supplies a information field for **lpq** (printer status) output.
5. The `:lp` value specifies the destination file, device or remote spool queue to which data is sent. In this example it is the device `/dev/lp1`. By default, IO devices are opened for *write-only* operation.
6. The `:sd=/path` specifies the *spool directory* where print job files are stored until they are printed.
7. The `:lf` and `:af` options specify the names of the log and accounting files, respectively. These have the default values `log` and `acct` respectively. If not an absolute path, the file is relative to the spool queue directory. If these files don't exist, they will not be created, and no logging or accounting will be done. You will need to create them manually (e.g., by using `touch`) or by using the *checkpc* program. If you do not want a log or accounting file, then use `:lf=`, i.e. - no value.
8. The `:filter=/path` option specifies a filter program to be used to process job files. Filters and print formats are discussed in Filters.
9. `mx` indicates the maximum file size for a print job. Specifying 0 means that there is no limit.

12.4. Using :oh To Select Printcap Information

When administering a large number of printers over a large area, it is sometimes desirable to have a *default* printer for each host. This default printer may be different for each host, and can be selected by using the `oh` entry. The `oh` value is a list of the following entries

```
[!] IP/n     - address + mask length    10.0.0.0/8
[!] IP/IP    - address + mask           10.0.0.0/255.0.0.0
[!] vvv      - glob for hostname        pc*.org.com
```

The **LPRng** software will determine the hostnames and IP addresses assigned to the host and then check to see if there is a match in the listed hostnames or IP addresses. The optional test inversion (`!`) causes the sense of the match to be inverted. The list of addresses or entries are tested in sequence until a match is found. If no match is found the printcap entry will not be used. For example:

```
lp:oh=*.admin.org.com,10.0.0.5,10.2.0.0/16:lp=pr1@server1
lp:oh=*.eng.org.com:lp=hp@server2
color:oh=*.eng.org.com:lp=color@server3
```

```
color:oh=!*.eng.org.com:lp=color@server4
```

In the above example, if our host name is `booster.admin.org.com`, then we would use `lp=pr1@server1`, as the `*.admin.org.com` glob pattern would match our host name.

if our host name is `booster.dev.org.com` and our IP address is 10.2.0.1, then we would use `lp=pr1@server1`, as the `10.2.0.0/16` ip address would be in the specified address range.

Finally we have an example of the use of the match inversion operator (`!`). All hosts whose name matches `*.eng.org.com` will use `color@server3` and the others will use `color@server4`.

12.5. Using the Wildcard Printcap Entry

The wildcard printcap name `*` is used to select a default or printcap entry when a match is not found in the printcap database.

```
# %P and %Q set to printer name
*:lp=%P@server

# %P set to 'printer', %Q set to printer name
printer|*:lp=%P@server
```

When searching for printcap information, the **LPRng** software will first search for an exact match for a printcap entry against the printcap names and aliases. If non is found, it then searches for a wildcard entry and uses the first one found with a wildcard name or alias.

If the wildcard `*` is the printcap name, then the printer name (`%P` value) and queue (`%Q` value) are set to the name being searched for. If the wildcard is an alias, then then printer name is set to the printcap main entry name and the queue to the name being searched for.

12.6. Enterprise Strength Printcap Example

Most system administrators want to have a single printcap file that can be distributed or referenced from all the hosts under their administrative control. The following show how several large institutions have organized their printcap information. It uses the following principles:

- Many times it is necessary to partition groups of users into areas that have particular printing setups, but use the same queue name for different printers. This is done by using the `:oh` (on this host) option to select the printcap to be used.
- Provide as little information as possible to the systems that do not have print spoolers. This is because they will simply forward their requests to a print spooler and it is not necessary to have the information for a printer. As we will see in an example below, this can be very simple to do.

- Provide a default printer so that when the user does not have a printer explicitly selected then he will get well behaved results.
- For each family of printers, develop a standard method of interfacing to them. This means that the filter, options, and other things should be identical.
- For each server, provide a uniform set of standards for the spool queues for a printer. If the spool queue is simply used to store and forward jobs, then provide a standard default operation for this queue. If the spool queue directly feeds a printer, use the default options for the printer, and then refine them with printer specific information.

```
# client setups; note brutality of this method that
#  assigns servers to clients based on their names
#  you could do this for IP addresses as well
# lp1 is default printer for Engineering
#   Default client information
.client=force_localhost@
#
lp1:oh=*.eng.com:lp=%P@server1.eng.com:tc=.client
*:oh=*.eng.com:lp=%P@server1.eng.com:tc=.client
color:oh-*.admin.com:lp=%P@server2.admin.com:tc=.client
*:oh=*.admin.com:lp=%P@server2.admin.com:tc=.client
lowres:lp=%P@general.services.com:tc=.client
*:lp=%P@general.services.com:tc=.client

# Standard Printer Configurations
# this is for the HP4simx, we use a standard filter setup
.cf_hp4simx
 :ifhp=model=hp4simx
 :filter=/usr/local/libexec/filters/ifhp
.cf_hplj5000
 :ifhp=model=hp5000
 :filter=/usr/local/libexec/filters/ifhp
# now we define the printers that use them
.pr_eng1:cm=Engineering's Printer 1
 :tc=.cf_hplj5000:lp=10.0.0.2%9100
.pr_eng2:cm=Engineering's Printer 2
 :tc=.cf_hplj5000:lp=10.0.0.23%9100

# now we define the server entries
# We set up server entries and then forward
#  to a single server that sends the the printer

.server
 :sd=/var/spool/lpd/%P
 :mx=0
pr1:oh=!10.0.0.5:lp=%P@10.0.0.5:server:tc=.server
pr1:oh=10.0.0.5:tc=.server
 :tc=.pr_eng1
```

12.7. Remote Printer Using RFC1179

Options used:

- `lp`=*destination*
- `rm`=*remote host (machine)*
- `rp`=*remote printer (machine)*

You can have the **lpd** server forward jobs to another server or print which supports the RFC1179 protocol by using the following printcap:

```
# Simplest
remote|Remote Printer
   :lp=raw@server
# historical
remote:
  rp=raw:rm=server
# Sometimes you have to connect to a non-standard port
special:lp=lp@server%2000
```

1. If the `lp` printer entry is present, it will override the `rm` and `rp` printer entries.
2. The `lp=pr@host` format specifies that the output device is actually a remote spool queue, and jobs should be transferred using RFC1179 protocol.
3. By default, **LPRng** will attempt to *sanitize* all jobs that it originates or forwards. This sanitization will result in an RFC1179 compliant `control file`, and will not modify any of the job information.

12.8. Remote Printer Using Socket API

If the spool queue destination is a remote printer supporting the Socket API, then you can have **LPRng** open a connection directly to the printer. These include the older Apple printers with TCP/IP support and the HP JetDirect supported printers.

```
# Simplest
remote
   :lp=10.24.2.3%9100
```

1. The `lp=server%port` or `lp=IPaddr%port` format specifies that **lpd** should open a TCP/IP connection to the remote host and simply transfer verbatum the files to be printed.

2. The `sh` and `sf` will prevent **lpd** from trying to generate banner pages or put form feeds between jobs.

While this is the simplest printcap, it is also the most dangerous as there is nothing to prevent a malformed job from being sent to the printer. The next printcap example is much more robust:

```
# Simplest
remote
   :lp=10.24.2.3%9100
   :of=/usr/local/libexec/filters/ifhp
   :filter=/usr/local/libexec/filters/ifhp
```

1. This version will use the **ifhp** filter to precondition the printer and to process jobs. See **ifhp** Filter for details. The **ifhp** filter will perform the appropriate printer resets, translate job information, and ensure correct printer operation in the presence of errors. It will also produce voluminous error messages and status information.

12.9. Parallel Printer

The parallel printer printcap is very simple.

```
# parallel printer
lp:
   :lp=/dev/lpr
```

1. The `lp=/dev/lpr` specifies that **lpd** should open the device for APPEND and simply transfer job files to it.
2. The `sh` and `sf` will prevent **lpd** from trying to generate banner pages or put form feeds between jobs.

If you discover that UNIX print jobs result in a *staircase* appearance, then you need to force your printer to do LF (linefeed) to `CR/LF` (carriage return/line feed) translation, or do the translation yourself.

```
# Simple parallel printer
lp:
   :lp=/dev/lpr
   :filter=/usr/local/bin/lpf
```

By using the `if=...lpf` filter, the job will be passed through the **lpf** filter, which will do the LF to `CR/LF` translation.

If you have a more complex printer that handles PostScript, PCL, and PJL, then you will need to use the more powerful **ifhp** filter:

```
# Simple parallel printer
lp:
   :lp=/dev/lpr
   :ifhp=model=hp4,status@
   :of=/usr/local/libexec/filters/ifhp
   :filter=/usr/local/libexec/filters/ifhp
```

See **ifhp** Filter for details. This entry will specify that the printer is an HP4, and that no status information is available. This is usually the case with a parallel port.

12.10. Serial Printer

Options used:

- `br`=*serial port bit rate*
- `rw` FLAG *device opened RW*
- `stty`=*stty options for serial port configuration*

The following is a typical printcap for a serial printer:

```
# Local Serial ASCII printer
lp2
  :lp=/dev/ttya
  :rw
  :cm=Serial printer
  :sd=/var/spool/lpd/lp2
  :stty=9600 -echo -crmod -raw -oddp -evenp pass8 cbreak ixon
  :filter=/usr/local/sbin/lpf
  :mx=0
```

Let's examine the new options:

1. A serial port is usually *bidirectional*, and printers will report errors back to the host computer. The `rw` flag will cause the printer port to be opened `read-write`, and the **lpd** server will report status information.
2. The `stty` option specifies the `stty(1)` flags and line speed needed to configure the serial line (See Serial Printers for details).
3. The legacy `br` (bit rate) option can be used to specify the line speed as well.

12.11. Bounce Queue

Options used:

- `lpd_bounce` FLAG *produces a single file for forwarding*
- `bq_format`=*format of filtered files*

When the destination of a spool queue is another spool queue the job is simply forwarded without any modifications. However, sometimes it is essential that the job be modified before forwarding, as when the remote spool queue is actually a printer, and jobs need to be converted to the format acceptable by the remote printer or banner pages added.

The `lpd_bounce` flag marks a spool queue as a bounce queue. `Lpd` will perform all of the usually job processing steps, such as banner generation, filtering files, etc, but saves the output to a file. This file is then sent to the destination print queue for further processing.

```
# Simple example of a bounce queue
bounce:lp=bounce@bouncehost
bounce:server
    :lp=lp@remote
    :lpd_bounce
    :sd=/var/spool/lpd/%P
    :filter=/usr/local/bin/lpf
    # uncomment ab if you want banner
    #:ab
```

Some comments:

1. The `lpd_bounce` option marks the job as a bounce queue, and the **lpd** server will process the job through the appropriate filter programs.
2. The printcap has filter specifications for different job formats. These are the programs that will be used by **LPRng** to process the job.
3. The `bq_format` specifies the job format for the output file sent to the remote spool queue. If not specified, it defaults to `l` (literal or binary).
4. The `ab` (always print a banner) flag will force a banner to be added to the job. The banner generation is done as discussed in Banner Printing.

12.12. Job Format Translation

Options used:

- `translate_format=`*change outgoing job file formats*
- `translate_incoming_format=`*change incoming job file formats*

A rarely encountered problem is when a job is printed with one format but for compatibility needs to be processed with another. The simple `translate_format=vlxf` option will rename format `x` files to `f` format.

This can be used to resolve problems with PC based software, which spools jobs using the `v` format. Unfortunately, many RCF1179 print spoolers do not understand the `v` format and mishandle the job. A simple forwarding queue with the following entries will rename `v` format to `l` (binary) format.

```
lp
  :sd=/var/spool/lpd/%P
  :translate_format=vl
  :lp=lp@printerserver
```

The `translate_incoming_format` will do the same thing, but this time on incoming jobs.

12.13. Dynamic Routing

Options used:

- `destinations=`*destinations for jobs*
- `router=`*router program*

LPRng has the ability to route a job to one or more destinations in a dynamic manner. This is not the same as *load balancing*, as the destinations are hard coded and not able to be changed. This is accomplished by having a `router` filter generate a set of destinations. Here is a sample printcap entry:

```
t2|Test Printer 2
    :sd=/var/spool/LPD/t2
    :lf=log
    :destinations=t1@server1,t1@server2,t1@localhost
    :router=/usr/local/LPD/router
```

When a job arrives at the **lpd** server, the 'router' filter is invoked with the standard filter options which include the user, host, and other information obtained from the control file. STDIN is connected to a

temporary copy of the control file, and the CONTROL environment variable is set to the value of the actual control file itself.

The routing filter exit status is used as follows:

- 0 (JSUCC) - normal processing
- 37 (JHOLD) - job is held
- any other value - job is deleted from queue

The router filter writes to STDOUT a file specifying the destinations for the job. The destinations entries in this file file have the following format. Entry order is not important, but each destination must end with the 'end' tag.

```
dest (destination queue)
copies (number of copies to be made)
priority (priority letter)
X(controlfile modifications)
end
```

Example of router output:

```
dest t1@localhost
copies 2
CA
priority B
end
dest t2@localhost
CZ
priority Z
end
```

In this example, two copies of the job will be sent to the t1 and t2 spool queue servers. The Class (C letter value) and job priority information will be rewritten with the indicated values.

If routing information is specified by the router filter the job will be sent to the default destination.

lpq will display job information in a slightly different format for multiple destination jobs. For example:

```
Printer: t2@h4 'Test Printer 2' (routed/bounce queue to 't1@h2.private')
  Queue: 1 printable jobs in queue
 Rank  Owner/ID        Class Job Files                 Size Time
active  papowell@h4+707 A 707  /tmp/hi                    3 10:04:49
 - actv papowell@h4+707.1 A 707 ->t1@localhost <cpy 1/2>       3 10:04:49
 -      papowell@lprng2+707.2 A 707 ->t2@localhost              3 10:04:49
```

The routing information is displayed below the main job information. Each destination will have its transfer status displayed as it is transferred. By convention, the job identifier of the routed jobs will have a suffix of the form .N added; copies will have CN added as well. For example, papowell@lprng2+707.1C2 will be the job sent to the first destination, copy two.

Routed jobs can be held, removed, etc., just as normal jobs. In addition, the individual destination jobs can be manipulated as well. The LPC functionality has been extended to recognize destination jobids as well as the main job id for control and/or selection operations.

The optional `destinations` entry specifies the possible set of destinations that the job can be sent to, and is for informational purposes only. In order for **lpq** and **lprm** to find the job once it has been sent to **lpd**, **lpq** and **lprm** uses the list of printers in the `destinations`, and iterates over list looking for the job that you are interested in.

12.14. Printer Load Balancing

Options used:

- `ss=`*queue served by printer*
- `sv=`*printers where jobs are sent (servers)*

In a large site, you could have several equivalent printers, which will be used by many people. The reason for this is, of course, to increase the printer output by enabling several jobs to be printed at once. A load balance print queue is one that feeds jobs to other queues and has a `sv=q1,q2,...` printcap entry that specifies the destination or server queues. These must be print queue entries and have spool directories on the server.

The service queues have a `ss=`*mainqueue* printcap entry This informs the **lpd** server that the queue operates under the control of the *mainqueue* print queue, and is fed jobs from it.

During normal operation, when the **lpd** server has a job to print in the *mainqueue*, it will check to see if there is an idle *service* queue. If there is, it will transfer the job to the service queue spooling directory and start the service queue printing activities.

Even though the queues are not meant for direct use, people can print directly to individual queues. This allows a specific load sharing printer to be used. If you wanted to *hide* the load sharing printers, i.e. - not allow direct spooling to them, then you would simply remove the non-server entries from the printcap.

12.15. Locations of Printcap Files

Options used:

- `printcap_path=`*printcap file locations*
- `lpd_printcap_path=`*lpd server printcap information*

The `printcap_path` and `lpd_printcap_path` configuration options (see lpd.conf(5)) specify a set of paths for the printcap information. If the `lpd_printcap_path` is nonblank then the **lpd** server uses it, otherwise it uses the `printcap_path` information. Client programs use only `printcap_path`. The path names can be separated with whitespace, commas, semicolons, or colons. The default values are:

```
printcap_path      ${sysconfdir}/printcap
lpd_printcap_path  (blank or empty)
```

12.15.1. Separate Server and Client Printcap Files

Since only the **lpd** server uses the printcap file specified by the `lpd_printcap_path`, you can place server specific information there. If this file is used it must also contain any required information in the `printcap_path` file.

12.15.2. `all` Printcap Entry

The `all` printcap name and `all` option is reserved to provide a list of printers available for use by the spooling software. This is a desperation, last ditch, back to the wall option for administrators with systems that do not have ways to provide a list of printcap entries. The 'all' printcap entry has the form:

```
all:all=pr1,pr2,...
```

The value of the `all` option should be a list of printcap names whose values will then be extracted.

12.16. Single Printcap File for Large Installation

One of the major problems faced by administrators of large sites is how to distribute printcap information. They would like to have a single printcap file either distributed by a file server (NFS) or by some other method such as `rdist`. By using the `server` and `oh` tags, information for the specific sites can be separated out. For example:

```
# printcap
pr1:lp=pr1@serverhost1:oh=*.eng.site.com,130.191.12.0/24
pr2:lp=pr1@serverhost1:oh=*.eng.site.com,130.191.12.0/24
pr1:lp=pr2@serverhost2:oh=*.admin.site.com
pr2:lp=pr2@serverhost2:oh=*.admin.site.com
pr1:server:oh=serverhost1.eng.com:lp=/dev/lp:tc=.common
pr2:server:oh=serverhost2.admin.com:lp=/dev/lp:tc=.common
.common:sd=/usr/local/lpd/%P
```

The above example has some interesting effects. The `pattern` is used as a *glob* pattern and is applied to the fully qualified domain name (FQDN) of the host reading the printcap file. For example, `*.eng.site.com` would match host `h1.eng.site.com` but would not match `h1.admin.site.com`. Thus, the effects of the first couple of entries would be to specify that the `pr1` and `pr2` printers on the `eng` hosts would be `pr1@serverhost1`, and on the `admin` hosts would be `pr2@serverhost2`,

Also, the lpd daemons on `serverhost1` and `serverhost2` would extract the additional information for `pr1` and `pr2` respectively, overriding the common `lp` entries.

12.17. Management Strategies for Large Installations

One very effective way to organize print spooling is to have a small number of print servers running a **lpd** daemon, and to have all the other systems send their jobs directly to them. By using the above methods of specifying the printer and server host you eliminate the need for more complex management strategies.

However, you still need to inform users of the names and existence of these printers, and how to contact them. One method is to use a common `printcap` file which is periodically updated and transfered to all sites. Another method is to distribute the information using the NIS or some other database. **LPRng** has provided a very flexible method of obtaining and distributing database information: see
Using Programs To Get Printcap Information for details.

12.18. Using Programs To Get Printcap Information

In the `lpd.conf` file you can specify:

```
printcap_path=|program
```

This will cause the **LPRng** software to execute the specified program, which should then provide the printcap information. The program is invoked with the standard filter options, and has the name of the printcap entry provided on STDIN. The filter should supply the printcap information on `stdout` and exit with a 0 (success) error code. By convention, the printcap name 'all' requests a printcap entry that lists all printers.

This technique has been used to interface to the Sun Microsystem NIS and NIS+ databases with great success. By having the invoked program a simple shell script or front end to the `nismatch` or `ypmatch` programs, the complexity of incorporating vendor specific code is avoided.

12.18.1. How to use NIS and LPRng

This note is based on material sent to the lprng@lprng.com mailing list by Paul Haldane <`paul@ucs.ed.ac.uk`>.

We generally don't use NIS for printcap files (we've moved to hesiod) but I can show you what we've done in the past.

The input to NIS is a normal printcap file:

```
# Classical printcap entry
lp23a|lp23|lp|main printhost printer - KB, EUCS front Door:\
        :lp=lp23a@printhost:\
        :sd=/var/spool/lpr/lp23a:

#lprng printcap entry
lplabel|lpl|TEST - Labels printer:
        :lp=:rm=printhost:rp=lplabel:
        :sd=/var/spool/lpr/lplabel:
        :rg=lpadm:mx=1:
```

To build the NIS printcap.byname map we add the following to the NIS makefile (along the other bits and pieces that the makefile needs to know about a new map).

```
PRINTCAP=${sysconfdir}/printcap
# warning : [  ] is actually [<space><tab>] in the script
printcap.time: $(PRINTCAP) Makefile
  if [ -f $(PRINTCAP) ]; then \
    sed < $(PRINTCAP) \
      -e 's/[   ][  ]*$$//' -e '/\\$$/s/\\$$/ /' \
    | awk '$$1 ~ /^#/{next;} $$1 ~ /^[:|]/ {printf "%s", $$0; next;} \
        {printf "\n%s", $$0 }' \
    | sed -e 's/[   ]*:[  ]*:/:/g' -e 's/[  ]*|[  ]*/|/g' \
      -e '/^[   ]*$$/d' > .printcap.$$$$; \
    cat .printcap.$$$$; \
    if [ $$? = 0 -a -s .printcap.$$$$ ]; then \
      awk <.printcap.$$$$ '{ FS=":"; OFS="\t"; } { \
          n = split($$1, names, "|"); \
          for (i=1; i<=n; i++) \
              if (length(names[i]) > 0 \
              && names[i] !~ /[ \t]/) \
                  print names[i], $$0; \
      }' | $(MAKEDBM) - $(YPDBDIR)/$(DOM)/printcap.byname; \
      awk <.printcap.$$$$ '{ FS=":"; OFS="\t"; } { \
          n = split($$1, names, "|"); \
          if (n && length(names[1]) > 0 && names[1] !~ /[ \t]/) \
              print names[1], $$0; \
      }' | $(MAKEDBM) - $(YPDBDIR)/$(DOM)/printcap.bykey; \
      rm -f .printcap.$$$$; \
      touch printcap.time; echo "updated printcap"; \
    fi \
  fi
  @if [ ! $(NOPUSH) -a -f $(PRINTCAP) ]; then \
      $(YPPUSH) printcap.byname; \
      $(YPPUSH) printcap.bykey; \
      touch printcap.time; echo "pushed printcap"; \
  fi
```

To specify that you want YP database rather than file access, use the following entry in your `/etc/lpd.conf` file:

```
printcap_path |/usr/local/libexec/pcfilter
```

Put the following shell script in /usr/local/libexec/pcfilter

```
#!/bin/sh
#/usr/local/libexec/filters/pcfilter
read key
# specify the full pathname to the ypmatch program
# the location depends on the version of Solaris or your
# system install
/full/pathname/to/ypmatch "$key" printcap.byname
```

You can test this by using:

```
h4: {314} # lpc client pr
pr
 :lp=pr@server
h4: {315} # lpc server pr
pr
 :lp=pr@server
```

12.18.2. How to use NIS and LPRng - Sven Rudolph

```
 Date: Wed, 11 Sep 1996 00:11:02 +0200
From: Sven Rudolph <sr1@os.inf.tu-dresden.de>
To: lprng@lprng.com
Subject: Using :oh=server: with NIS
```

When I use a cluster-wide printcap, I want the entries for each printer to appear, e.g.:

```
---------- start of printcap snippet
lp1
 :lp=lp1@server
lp2
 :lp=lp2@server
lp1
 :server:oh=servername
 :sd=/var/spool/lpd/lp1
 :lp=/dev/lp1
```

```
 :mx=0
---------- end of printcap snippet
```

When I create a NIS map out of this the printer name is used as a key and must be unique. The NIS makedbm will drop all but the last entry for each printer. This makes the printer on the clients unavailable. I solved this by a hack where the second entry is called lp1.server and the NIS client script has to request the right entry.

1. Assumptions

   Perl is available at the YP server in `/usr/bin/perl`. A Bourne Shell is available at all clients in `/bin/sh` The printcap that is to be exported is in `/etc/printcap`. The printcap is written in the new format. In the examples the printer is called lp1.

2. Add the following to your YP `Makefile` (`/var/yp/Makefile`) on the YP server (these lines are for Debian GNU/Linux, other systems might require other modifications):

   ```
   ---------- start of /var/yp/Makefile snippet
   PRINTCAP  = /etc/printcap
   printcap: $(PRINTCAP)
       @echo "Updating $@..."
       $(CAT) $(PRINTCAP) | \
           /usr/lib/yp/normalize_printcap | $(DBLOAD) -i $(PRINTCAP) \
           -o $(YPMAPDIR)/$@ - $@
       @if [ ! $(NOPUSH) ]; then $(YPPUSH) -d $(DOMAIN) $@; fi
       @if [ ! $(NOPUSH) ]; then echo "Pushed $@ map." ; fi
   ---------- end of /var/yp/Makefile snippet
   ```

3. Install the programs **match_printcap** and **normalize_printcap** in the `/usr/lib/yp` directory; **normalize_printcap** is only required on the YP server. The **normalize_printcap** processes only the **LPRng** printcap format.

   ```
   ---------- start of /usr/lib/yp/normalize_printcap
   #! /usr/bin/perl
   $debug = 0;
   $line = "";
   $new = "";
   while (<>) {
       chomp;
       next if ( /^\s*\#.*/ );
       s/^\s*$//;
       next if ( $_ eq ” );
       print "new: " . $_ . "\n" if $debug;;
       if (/^\s/) { # continuation line
           $line = $line.$_;
           print "continued: $line\n" if $debug;
           next;
       } else {
           $line =~ s/\s+\:/:/g;
           $line =~ s/\:\s+/:/g;
           $line =~ s/\:\s*\:/:/g;
   ```

```
        print "line: $line\n" if $debug;
        push(@lines, $line) if $line;
        $line = $_;
    }
}
$line =~ s/\s+\:/:/g;
$line =~ s/\:\s+/:/g;
$line =~ s/\:\s*\:/:/g;
push(@lines,$line) if $line;
@lines = sort(@lines);
foreach $line (@lines) {
    ($printers) = split(/\:/,$line);
    @printers = split(/\|/,$printers);
    foreach $printer (@printers) {
      $num{$printer}++;
      push(@allprinters,$printer);
      print "allprinters: @allprinters\n" if $debug;
      print $printer."_".$num{$printer}."\t$line\n";
    }
}
@pr = keys %num;
print "printers @pr\n" if $debug;
if ($#allprinters >=0) {
    print "all_1\tall:all=".join(",",@pr)."\n";
}
---------- end of /usr/lib/yp/normalize_printcap
```

The result of processing the sample printcap file is:

```
lp1_1 lp1:lp=lp1@server
lp1_2 lp1:server:oh=servername:sd=/var/spool/lpd/lp1:lp=/dev/lp1:mx=0
lp2_1 lp2:lp=lp2@server
all_1 all:all=lp1,lp2
```

Observe that each of the real printer entries has a key consisting of the printer name with a numerical suffix. This leads to the following method of extracting the printcap information using `ypmatch`:

```
---------- start of /usr/lib/yp/match_printcap
#!/bin/sh
read p
n=1
# specify the full pathname to ypmatch - this depends on your
# OS version and installation
while /full/pathname/to/ypmatch "${p}_${n}" printcap 2>/dev/null; do
    n=`expr $n + 1`
done
---------- end of /usr/lib/yp/match_printcap
```

4. Now test the YP arrangement:

```
h4: {316} #  cd /var/yp; make
    # this should create the printcap map
h4: {317} #  ypcat printcap
    # should provide the whole normalized printcap
h4: {318} #  echo lp1 |/usr/lib/yp/match_printcap
    # yields lp1 printcap
```

5. Modify the `printcap_path` entry in the `lpd.conf` file:

```
printcap_path=|/usr/lib/yp/match_printcap
```

6. Test the use of the printcap path entry:

```
h4: {319} #  lpc client lp1 # shows the printcap for lp1
h4: {320} #  lpc server lp1 # shows the printcap for lp1
```

7. Restart the lpd server and check to see that it accesses the right printcap information. Use the same **lpq** command, and then try `lpc printcap lp1`.

12.19. Lexmark Printers

Some Lexmark printers do not send *end of job* status back unless configured to do so. Here is what is needed to force this. (This capability has been incorporated into the **ifhp** filter.)

```
From: Matt White <whitem@bofh.usask.ca>
To: lprng@lprng.com
Date: Wed, 21 Jan 1998 18:25:50 -0600 (CST)
Subject: Re: [LPRng] ifhp with Lexmark Optra N printer

On Wed, 21 Jan 1998, Simon Greaves wrote:

> Apologies in advance if this is way off mark, but we've been
> evaluating a commercial print charging package (Geomica) which
> works by talking to the printer in what I think is a similar way
> to the ifhp filters. Lexmarks are currently a big headache because
> they seem to fail to return the message that they have finished
> printing which screws things up somewhat. In our case, it is believed
> to be a problem with the Lexmark firmware which they are looking
> into.

There is a fix for that...it is originally from the Lexmark 4039
series, but it still works on the Optra S 1650 machines that we
have (and should work on the rest of the optra line).  Just send
this little chunk of postscript to the printer once:

-----------snip----------
%! Postscript to set the 4039 printer into synchronous mode
```

```
serverdict begin 0 exitserver
statusdict begin true setenginesync end
----------snip----------

Basically, it causes the printer to wait until it is finished
printing before actually reporting that it is done.  I've got 3
Optra S printers running with ifhp right now with no extra options
(just defaults).

------------------------------------------------------------
- Matt White                  whitem@arts.usask.ca
- Network Technical Support   http://arts.usask.ca/~whitem
- College of Arts & Science  University of Saskatchewan
------------------------------------------------------------
```

12.20. Tektronix Phaser Printers

The **ifhp** filter supports the AppSocket protocol used by Tektronix. You will need a printcap similar to the following:

```
phaser:
 :sd=/var/spool/lpd/%P
 # need a dummy device for output
 :lp=/dev/null
 # You need to specify the IP address of the printer's network interface
 #  In this example it is 10.0.0.1 - replace with the correct value
 # The filter will actually open the connection.
 :filter=/usr/local/libexec/filters/ifhp
 :ifhp=dev=10.0.0.1%9100,model=tek
```

12.21. Duplex Printing

Duplex printing is when you print on both sides of a page. Some printers which do duplex printing require that you send them special commands to force this mode. This is usually done by the FILTERS. The **ifhp** filter makes a stab at sending the PJL or PostScript commands to the printer. Many people have reported problems doing duplex printing, so here is a check list.

1. Make sure you have enough memory for the worst case print job. Usually the printer has to rasterize both pages before it can produce an impression. It may require much more memory than you expect.

2. Check your printer manual to discover the EXACT form of the `enter duplex mode` command and make sure that either the command is part of the job (PJL language at the start of the job, postscript header, etc), or that the filter generates the correct form.

   Note there is a PostScript Printer Description file (PPD) for most printers that support PostScript, and they even have the PJL and PostScript code for this in the PPD file.

3. It has been observed that even with what would apparently be sufficient memory, that many duplex jobs print 'oddly', that they are not aligned on the same side in the same way, etc etc. This may not be the fault of the software, but of the support for duplex operation.

4. Read the **ifhp** documentation, and create a configuration section in the `ifhp.conf` file for your printer.

I know this is painful, but until there is a uniform way to get the correct commands extracted from either PPD or some other database then this appears to be the only way to do it.

Patrick Powell

12.22. Solaris, Newsprint and FrameMaker

The following is a guide to using **LPRng** and Sun Microsystems Newsprint by Christopher Hylands, Ptolemy Project Manager of the University of California.

The Sun Newsprint printer is actually an OEM version of the Tektronix PhaserII; Sun Microsystems appears to have dropped support for Newsprint, and the recommended migration path is to buy a PostScript printer. If you want more information on using the Newsprint system, notes are available via `http://ptolemy.eecs.berkeley.edu/~cxh/lprng.html`.

Looking through the mailing list logs, it looks like everyone was having a hard time getting lprng to work with Sun's brain-dead newsprinters. I tried using GhostScript, but the fonts were, IMHO, ugly, so I spent a little time getting the newsprint fonts to work.

The key thing was to grab the file `/usr/newsprint/lpd/if` from a SunOS4.1.3 newsprint installation. If you cannot get this code, then the installation will be extremely difficult.

To install lprng on a Solaris2.x machine, you need to first stop the existing print services and install the startup scripts for **LPRng**. Note that if there is a local printer, you may have to also fix the permissions of the device. Typical commands are:

```
chown daemon /devices/sbus@1,f8000000/SUNW,lpvi@1,300000:lpvi0
```

We use the following simple `:if` script.

```
#/bin/sh
# extremely simple filter script
/bin/cat
```

The Sparcprinters use licensed fonts from NeWSprint. To use the licensed fonts, you must have the lprng spool directory for the sparcprinter in the same location as spool directory of the brain dead Solaris lp system. If your printer is named xsp524, then this directory would be `/etc/lp/printers/xsp524`.

The printcap entry looks like:

```
sp524|524:
    :mx=0
    :lp=:rm=doppler:rp=xsp524:
    :sd=/var/spool/lpd/sp524d:
    :lf=/var/spool/lpd/sp524d/log:
xsp524|Sun SPARCprinter NeWSprint printer:
    :mx=0:rs:
    :lp=/dev/lpvi0:
    :sd=/etc/lp/printers/xsp524:
    :lf=/etc/lp/printers/xsp524/log:
    :af=/var/spool/lpd/xsp524/acct:
    :filter=/usr/local/libexec/newsprint/if:
```

The `/usr/local/libexec/newsprint/if` was copied from `/usr/newsprint/lpd/if` in a SunOS4.x installation of the newsprint software. Unfortunately, the newsprint engine is so brain dead that it needs many environment variables set, so it is fairly difficult to come up with a clean script to start the engine. I made the following changes to the file.

1. First, set the path in the script. You may also need to change defaults to suit your preferences:

   ```
   PATH=/usr/ucb:/usr/bin:/etc:/usr/etc:\
     /opt/NeWSprint/bin:/opt/NeWSprint/np/bin:
   PATH=$PATH:$NPHOME/pl.$ARCH/bin:$NPHOME/np/bin; export PATH
   ```

2. You will also need a `/etc/lp/printers/printername/.params` file. If you are using the same spooler directory as the directory that the Solaris lp system uses, then the .param file should appear there. If you are using a different spooler directory, then you will need to copy the .param file from elsewhere and edit it accordingly.
3. If you are going to move a license to a new printer, you should probably save the .param file in the old printer spooler directory. Run /opt/NeWSprint/bin/fp_install and remove the license from the old printer and assign it to the new printer. You could run /opt/NeWSprint/bin/rm_np_printer and remove the printer, but that will get rid of the .param file
4. FrameMaker under Solaris2.x uses the lp command. The fix is to edit $FMHOME/fminit/FMlpr and comment out the lp line and add an lpr line

   ```
   sunxm.s5.sparc)
       lpr -P"$PRINTER" "$FILE"
       #lp -c -d"$PRINTER" "$FILE"
   ```

Christopher Hylands, Ptolemy Project Manager University of California
cxh@eecs.berkeley.edu   US Mail: 558 Cory Hall #1770

```
ph: (510)643-9841 fax:(510)642-2739      Berkeley, CA 94720-1770
home: (510)526-4010 (if busy -4068)      (Office: 493 Cory)
```

Chapter 13. Spool Queues and Files

When files are accepted by the **lpd** server for printing, they are stored in a spool queue directory, together with other files controlling the print operation. This section describes these files and how the **LPRng** software uses them.

For descriptive purposes, we will use the following printcap entry as a guide:

```
pr|alias
  :sd=/var/lpd/pr_public
  :cd=/var/lpd/pr
```

13.1. Spool Queue

- `sd=`*Spool queue directory name*

The `:sd` option in the printcap entry specifies the spool queue directory. If there is no `:sd` entry or value, then the printer can only be used by the clients such as **lpq** to locate the destination for a print job. All information, files, etc., for a print queue is stored in the spool directory.

13.2. Queue Lock File

- `spool_lock_file`*spool queue lock file - default %P*

When the **lpd** server starts printing, it will fork individual worker processes to service each queue. To prevent multiple processes from working on the same queue, a printer lock file specified by the `queue_lock_file` option (default `%P` - the %P is expanded to the print queue name) is used. In our example, the lock file would be: `/var/lpd/pr/pr`.

The process ID of the currently active printer is stored in the lock file. By reading the lock file and testing to see if the process is still active, programs such as **lpq** can determine queue activity.

Similarly, the worker process may need to create other processes to assist it. These in turn will create lock or temporary files in the spool directory as well.

13.3. Spool Control File

- `spool_control_file`*spool queue control file - default control.%P*

The spool control file is used to control the operations of the spooler, and is in the spool or control directory. The file name specified by the `queue_control_file` option (default `control.%P` - the %P is expanded to the print queue name); in our example, the control file would be: `/var/lpd/pr/control.pr`.

The **lpc** program sends spool control requests to the **lpd** daemon, which updates the control file and then signals the appropriate spool server processes that an update has been performed. The control file contents have the form:

```
key value
```

The following keys and their values are currently supported.

Key	Value	Purpose
`printing_disabled`	0 or 1	disable printing of jobs in queue
`spooling_disabled`	0 or 1	disable placing jobs in queue
`holdall`	0 or 1	hold jobs until released
`redirect`	`printer`	transfer jobs to indicated printer
`class`	`glob expression`	print only jobs whose class matches glob expression
`server_order`	`printer name list`	preferred order of printer use
`debug`	`debugging options`	debugging and tracing

The `printing_disabled` and `spooling_disabled` are managed using the `lpc start`, `lpc stop`, `lpc enable` and `lpc disable` commands. Similarly, `holdall` is enabled and disabled by `holdall` and `noholdall` commands respectively. When holdall is enabled, jobs placed in the print queue will be held until they are explicitly released for printing by an `lpc release` command.

The `redirect` entry is used to redirect or transfer jobs which are spooled to this queue to another queue, and is managed by the redirect command. The `lpc redirect off` removes the redirect entry from the control file.

The `class` entry is similar in operation to the `holdall`, but allows jobs whose class identification matches the glob expression to be printed. This can be useful when you have special forms or paper required for a print job, and want to run only these jobs when the paper is in the printer.

The `server_order` entry is created and updated for a multiple printer queue. It records the order in which printers should next be used for normal print operations. This allows *round robin* use of printers, rather than having all jobs printed to the first printer in the list of printers.

The `debug` entry is set by the `lpc debug` command, and is used to enable or disable debugging and tracing information for a spool queue. This facility is for diagnostic purposes only.

13.4. Log and Status Files

- `create_files`*create log, accounting and status files*
- `lf=`*log file name (default: log)*
- `max_log_file_size#`*maximum log file size (Kbytes)*
- `min_log_file_size#`*minimum log file size (Kbytes)*
- `max_accounting_file_size#`*maximum accounting file size (Kbytes)*
- `min_accounting_file_size#`*minimum accounting file size (Kbytes)*
- `max_status_line#`*maximum status line length (characters)*
- `max_status_size#`*maximum status file size (Kbytes)*
- `min_status_size#`*minimum status file size (Kbytes)*
- `ps=`*filter status file name (default: status)*
- `queue_status_file=`*queue status file (default: status.%P)*
- `short_status_date=`*display short (hh:mm) timestamp (default: true)*

During operation, the **lpd** server records the current printing operations in the spool queue status file specified by the `spool_status_file` option (default `status.%P` - the %P is expanded to the print queue name); for our example, this would be `/var/lpd/pr/status.pr`. In order to prevent this file from growing too large, the server will periodically truncate the file. You can force creation of these files by setting the `create_files` option. The `max_status_size` configuration or printcap option sets the maximum size (in Kbytes) of the status file; if the file exceeds this, only the last `min_status_size` bytes or 25% of the maximum size (default if not specified) will be preserved.

The server logs its operations in the log file specified by the `lf` (log file) option (default is `lf=log`). The `max_log_file_size` value (default 0) specifies the maximum length of the log file in Kbytes. If this value is non-zero, then the log file is truncated to `min_log_file_size` bytes or 25% of the maximum file size. Again, the last portion of the log file is preserved. If the `max_log_file_size` value is 0, then the log file grows without limit.

The server records accounting information in the log file specified by the accounting file `af` (accounting file) option (default is `af=acct`). The `max_accounting_file_size` value (default 0) specifies the maximum length of the log file in Kbytes. If this value is non-zero, then the log file is truncated to `min_accounting_file_size` bytes or 25% of the maximum file size. Again, the last portion of the log file is preserved. If the `max_accounting_file_size` value is 0, then the log file grows without limit. See Open The Output Device and Accounting Printcap Options for more details.

Some filters require an additional filter status file that they use for recording additional filter status or other operational information. The `ps` names this file, and it is passed to a print filter using the `$s` option (see Filter Command Line Options and Environment Variables).

The STDERR output for filters is put into the printer status file. This allows the filter to produce informative messages that can be displayed as part of the user status. In addition, a separate status file specified by the `ps` (Printer Status) can be used as well. This file is *not* truncated by the **LPRng** system.

When reporting status information, the length of line returned can be a problem. The `max_status_line#79` option restricts the status line to a maximum of 79 characters.

The `short_status_date` (default is true) option causes short (hour:minute) timestamps to be displayed on status queries.

13.5. Job Files

- `longnumber`*long job number*
- `default_priority=`*default job priority*
- `nline_after_file`*N line after data file*

A print job consists of a control file and one or more data files. RFC1179 specifies the general format of these files and how they are to be transfered between servers. **LPRng** has extended the contents of the control files and the transfer protocol to provide a more powerful set of features, but has extensive provisions for backwards compatibility with non-**LPRng** software. A sample control file is shown below:

```
Hh4.private
J/tmp/file1 /tmp/file2
CA
Lpapowell
Ppapowell
fdfA002230h4.private
N/tmp/file1
UdfA002230h4.private
fdfB002230h4.private
N/tmp/file2
UdfB002230h4.private
```

The first part of the control file contains general information generated by the **lpr** or other spooling program. The information lines start with an uppercase letter or digit. Some other spooling systems also start information lines with various punctuation marks such as underscores (_) or periods (.).

Following this are a set of entries about each of the various files to be printed. These lines start with a lower case letter, followed by the print file name. The lower case letter is the *format* to be used to process the file. See print file formats for more information about its use.

Table 13-1. Control File Lines

Key	Meaning	Generated By
Key	Meaning	Generated By
A	identifier *	**LPRng** internal
C	class	lpr -C class
D	date	lpr
H	originating host	lpr
I	indent	lpr -i indent
J	jobname	lpr -J jobname (default: list of files)
L	bnrname	lpr -U username
N	filename	(see text)
M	mailname	lpr -m mailname
P	logname	lpr
Q	queuename	lpr -Q
R	accntname	lpr -R accntname
S	slinkdata *	lpr
T	prtitle	lpr -T prtitle
U	unlnkfile	(see text)
W	width	lpr -w width
Z	zopts *	lpr -Z zopts
1	font1	lpr -1 font1
2	font2	lpr -2 font2
3	font3	lpr -3 font3
4	font4	lpr -4 font4

The entries marked with * are used only by **LPRng**. `N` and `U` lines are associated with a print file. The `N` line is the original name of the print file. By default, **LPRng** places this line *before* the corresponding data file. You can use the `nline_after_file` option to have **LPRng** place the N line after the data file line. The `U` line originally was used to indicate that the named file was to be unlinked after printing. This information is now ignored by **LPRng**. These lines are always grouped with a print file entry.

The names of control and data files follow a very strict pattern. Control files have the format `cfX`*`number`*`host`, where X is an upper case letter, *number* is (usually) a 3 digit number, and *host* is the host name. RFC1179 restricted the total length of the control file name to 32 characters; **LPRng** has a much looser limit.

Data file names must follow the same pattern as the control file name, and have the format `dfX`*`number`*`host`. The X can be in the range A-Za-z, allowing at most 52 data files for a job. The *number* and *host* must be identical to the corresponding control file.

By convention, **LPRng** uses the X of the control file name to set a priority for the job. A job with control file name `cfA...` will have *lower* format than a job with format `cfB...`, and so forth. The **lpr** program uses the first letter of the class name or an explicit priority value to set the letter value. If none of these are specified, then the `default_priority` value from the configuration or printcap entry is used.

The job number is usually a 3 digit value. However, in systems where a large number of jobs are spooled and need to be kept for printing at scheduled times, this can lead to problems. The `longnumber` option will use 6 digit job numbers. This must be used with care when operating with non-**LPRng** software.

13.6. Job Hold File

The information used to control the printing of a job is string in a *hold file*. The entries in this file have the form:

```
key=[value]
```

The following is an example of a hold file:

```
server=0
A=papowell@astart.com
J=jobname
transfername=cfA001astart.com
datafiles=N=file\002transfername=dfA001astart.com
subserver=0
attempt=3
error=cannot open printer
hold=0
priority=0
remove=0
routed=0
```

The contents of the control file are stored as X=<value> entries, where X is the upper case letter corresponding to the control file entry. The `datafiles` entry contains the data file information, as a set of fields separated by field separator characters.

The `server` and `subserver` entry records the process ID of the server process and the subserver process that is printing the job. The `attempt` field records the total number of attempts to print the job. The `error` field records any error that would prevent the job from being printed. This information is reported by the **lpq** program.

The `hold` field is non-zero when the `lpc hold` command is used to explicitly prevent the job from being printed; `lpc release` will clear the field and allow the job to be printed.

The `priority` field is modified by the `lpc topq` command and is used to provide an overriding priority to printing the file.

The `remove` field is non-zero when the file has been printed and should be removed.

The `routed` field is used to indicate that there is routing information present in the hold file, and that special handling is needed. The routing information is provided by a routing filter. The information is recorded by information in the hold file. The following is an example of routing information. Normally this information is stored in a compressed format with one line per destination, but for clarity this has been broken out into plain text form:

```
active 0
attempt 0
done 0
hold 0
priority 0
remove 0
routed 880892602
route 1
  dest t1
  ident papowell@h4+705.1
  error
  copies 1
  copy_done 0
  status 0
  active 0
  attempt 0
  done 0
  hold 0
  sequence 0
  priority B
  CB
  end
route 2
  dest t1
  ident papowell@h4+705.2
  error
  copies 0
  copy_done 0
  status 0
  active 0
  attempt 0
  done 0
  hold 0
  sequence 1
  end
```

Routing information lines start with `route` followed by individual routing entry information. The `route dest`, `copies`, `priority`, and `Xnnnn` entries are derived from the output of the router program; other fields are used during the printing process. The `copy_done` records the numbers of copies done, while the `done` records that the entry has been completed. The `status` is the process ID of the server process doing the printing.

The output from route filter that generated the above file was:

```
dest t1
copies 1
priority B
CB
end
dest t1
end
```

13.7. Job State

Options used:

- `ah` FLAG *Automatically hold jobs*

A job can be in the following state:

1. Initial. This is the state during job submission. Jobs in the initial state do not have any status displayed for them.
2. Held. Once a job is submitted, it can either be printed or *held*. The `ah` printcap option specifies that all jobs are automatically held on submission. The `lpc release` and `lpc redo` command will cause these jobs to be printed and the **lprm** command can remove these jobs.
3. Active. The job is being processed for printing or transfer to another queue.
4. Pending. Jobs which can be printed but are not active. This can be due to the printer being busy or the job *class* not being printed.
5. Error. Jobs which have encountered an error during printing. The `lpc release` and `lpc redo` command will cause these jobs to be printed and the **lprm** command can remove these jobs.
6. Done. Jobs which have completed printing, but which are not yet removed from the print queue. See the *save_when_done* flag for more information. The **lprm** command can remove these jobs.

Normally the job sequences is initial, pending, active, and done. However, a job may be put in the error state by problems processing the job or by actions of the **lpc** command.

13.8. Job Identifier

For each job in a spool queue, the **LPRng** software creates a unique identifier. This identifier is recorded in the control file `A` line. It can be used by the various client programs for identifying jobs, and is displayed by the **lpq** program as status information.

Chapter 14. Configuration File, Defaults and Overrides

Options used:

- `allow_getenv` FLAG *use GETENV environment variable*

The **LPRng** options are obtained as follows:

- The compile time defaults. These are in the `LPRng/src/common/vars.c` file.
- If the **LPRng** software has been compiled with the regression testing GETENV option enabled, the configuration information in the file specified by the `LPD_CONF` environment variable will be used. This can only be used if you are not setuid ROOT or as ROOT as it opens severe security loopholes.
- The file specified by the `config_file` compile time option, usually `/etc/lpd.conf` or `/usr/local/etc/lpd.conf`, and referred to a the `lpd.conf` file. If the `config_file` option value has the form `|/pathname`, then `/pathname` must be an executable program and will be run with the standard set of filter options. It must write configuration option values to its STDOUT and exit with a 0 status.
- In order to protect system security, the `lpd.conf` (and the `printcap`) file should be read only.
- If the `require_configfiles` option is set in the compile time options, then the preceeding step must be successful, i.e. - there must be a configuration file or the program must execute and exit with a 0 status.
- If a printer or spooling operation is done, then the values in the `printcap` entry for the spooler are used to override the default and `ifhp.conf` file values.

14.1. Configuration File Format

The configuration file format is similar to a the fields of a printcap entry with the difference that the leading colon is optional and there can only be one option per line:

```
# comment
# set option value to 1 or ON
ab
:ab
# set option value to 0 or OFF
ab@
:ab
# set option value to string
str=name
```

During system installation the **LPRng** software processes the default values in the `LPRng/src/common/vars.c` file and generates a sample `lpd.conf` file that has the format:

```
# Purpose: always print banner, ignore lpr -h option
#   default ab@ (FLAG off)
# Purpose: query accounting server when connected
#   default achk@ (FLAG off)
# Purpose: accounting at end (see also af, la, ar, as)
#   default ae=jobend $H $n $P $k $b $t  (STRING)
# Purpose: name of accounting file (see also la, ar)
#   default af=acct  (STRING)

# change:
# --- we change the af value to none, i.e. - no accounting
# --- file by default
af=
```

You can change option values by editing the file as shown above then then to force the **lpd** server to use the new options, use the `lpc reread` command.

14.2. Legacy Compatibility

The following arguments have been provided for compatibility with legacy systems.

Chapter 15. Job Processing

Much of the flexibility of the **LPRng** software is obtained from the ability to control the details of each step of job processing. The following section details each step in the processing of a job, and explains the printcap options used to control each operation.

Assume the `pr` printcap entry has the form:

```
pr
    :lp=/dev/lp  OR  :lp=rp@rm
    :sd=/var/spool/lpd/pr
    :lf=log
    :filter=/usr/local/bin/lpf
```

Assume that we have used the following command to print a set of files.

```
lpr -Ppr file1 file2
```

This will create a control file in the `/var/spool/lpd/pr` directory with the following contents (this is an example - in practice there may be minor differences between the example and an actual control file):

```
Hh4.private
J/tmp/file1 /tmp/file2
CA
Lpapowell
Ppapowell
fdfA002230h4.private
N/tmp/file1
UdfA002230h4.private
fdfB002230h4.private
N/tmp/file2
UdfB002230h4.private
```

We will refer to this example throughout the following sections.

15.1. Configuration and Setup Options

Options used:

- `ipv6` FLAG *use IPV6 Network facilities*
- `default_tmp_dir`=*temporary directory*
- `lockfile`=*lpd server lock file*

- `report_server_as`=*server name for status reports*
- `spool_dir_perms`=*spool directory permissions*
- `spool_file_perms`=*spool file permissions*

The `ipv6` specifies that the IPV6 protocol, rather than IPV4 will be used.

The `lockfile` specifies the location of the lock file used by the **lpd** server. This file has the port number in the `lpd_port` value appended to form a unique lock file name.

The `spool_dir_perms` and `spool_file_perms` (default 0700 and 0600 respectively) values are the (numeric) permissions for the spool directory and spool files.

The `report_server_as` option allows an administrator to masquerade a server with another name. This could be useful if various load sharing activities are being carried out, or if there are problems reconfiguring DNS to cause the correct server name to be reported.

The `default_tmp_dir` option specifies a temporary directory to be used to hold files or information temporarily if there is no spool directory available.

15.2. Submitting Jobs and Service Requests

Options used:

- `lpd_port`=*Listening Port for* ***lpd***
- `unix_socket_path`=*Unix socket for* ***lpd*** *connections*

After the **lpd** server has done its initialization, it will attempt to bind to the **lpd** listening port specified by the `lpd_port` value. This value has the format `[ipaddr%]port`. If the `ipaddr` is specified then the lpd server binds to the interface with the specified address otherwise it binds to all interfaces. The port value can be a number or name the name of a service; The port corresponding to the service name is used. The `printer` services port is 515. If the port binding operation is successful and the server has not been request to run in *foreground* mode by the `-F` command line option, then a child process is forked and the parent process will exit. The child process then takes steps to disconnect itself from the control terminal of the process that started it.

The `unix_socket_path` option specifies the pathname of a *fifo* socket that local processes can use instead of the TCP/IP port.

The main **lpd** process will then start a *queue checking* process that will check all of the spool queues used by the server for queues that have pending jobs. This process sends a message to the main **lpd** process requesting that it start a service process for this queue.

The **lpd** process will then sit in a loop waiting for one of the following events:

1. An incoming connection request. If the maximum number of children has not been exceeded, then a new process will be forked to handle this connection.

2. A child process exiting. The server will check to see if there is a pending request to start a server for a queue that could not be accommodated due to too many processes running.
3. A request to start a service process for a queue. If the number of active processes is less than the maximum allowed a service process will be started, otherwise the request will be placed on a list for service when the number of processes active decreases.
4. A timeout alarm for the queue rescanning operation. This is discussed in the next section in detail.

When connection is accepted by the **lpd** spooler, the following steps are taken to processes the job.

1. First, a timeout is established for the transfer of the information from client to the **lpd** server. This is done to prevent a denial of service attack by processes that do not close connections in a timely manner.
2. A single line is read into an internal buffer. This line must be terminated with a `NEWLINE` character.
3. The input line is parsed and the actions required are determined.
4. If the activity requires access to the spool queue information, then the current directory of the process is changed to the spool directory. This allows all file accesses to then be relative to this directory.
5. If the processing requires starting a spool queue server process, a message is sent to the main **lpd** server process to start a spool queue server process. By having all the processes serving spool queues children of the main server process it is possible to monitor and limit the total number of active processes. This is important on systems with a very large number of queues.
6. After the processing of the original request has been completed, the process with then check to see if the Spool Queue for the printer should be processed.

15.3. Job Reception

- `longnumber` FLAG *Long job number (6 digits)*
- `fifo` FLAG *enforce FIFO order for reception*
- `lpd_listen_port`=***lpd** will listen on this port*
- `incoming_control_filter`=*filter to modify incoming job control file*
- `translate_incoming_format`=*change data file formats*
- `accounting_fixupname`=*change accounting name infomration*

When a print job is received, the **lpd** server will assign a job number to the new job. Historically these have been in the range of 0 to 999, but the `longnumber` option allows numbers from 0 to 999,999 to be assigned. The server then checks to see that all of the data files for a job have been transferred correctly.

The `fifo` flag forces all jobs receieved from a particular host to be processed in First In, First Out (fifo) order. No new jobs will be processed until the incoming job has been released into the spool queue.

If an incoming control file filter is specified, then the incoming job's control file will be passed through the `incoming_control_filter` filter if it is specified. This allows the modification of the control file.

The majority of control file modifications are simple job file format changes. The `translate_incoming_format` option provides a simple way to do this. See the *translate_format* for details.

The `accounting_namefixup` option was introduced to allow a simple mapping of host and user names to names to be used for accounting purposes. By convention, the `R` field in the job control file specifies the name to be used for accounting purposes.

```
accounting_namefixup=list[,list]*
   where list is:   host(,host*)[=user(,user*)]
```

The incoming job is checked to see if the originating host is in the list of hosts; the first matching one found is used.

Each host list has the format: host,host... where host has the same format used for the `oh` and other host name matching options. You can use '!host' to invert matching. For example: `host1,127.*,!somehost`.

When a host match is found, the name to be used for the user is determined from the user list; if none is specified then no changes are made. Each entry in the user list has the format `${option}` or `name`; the `${option}` values are extracted from the control file (capital letters) or printcap/configuration information (lower case letters/names). The first non-empty value list value used. For example, the `${R},${L},${accounting_name},anon` will select the control file 'R' option value, then the 'L' option value, then the printcap/config option 'accounting_name' value, and then finally the 'anon' value.

The control file is then passed through the `router` routing filter. This allows the incoming job to be redirected to one or more print queues. For details about all of the capabilities of the routing filter, see Dynamic Routing.

Finally, the **lpd** server is requested to start a spooling process that will print the newly arrived job.

15.4. Spool Queue Processing

Options used:

- `lpd_force_poll`=*Force **lpd** to periodically poll print queues*
- `lpd_poll_time`=*Time between polls*
- `max_servers_active`=*Maximum number of active servers*

When the **lpd** server starts, it will fork a set of subserver processes, each which will handle an individual queue.

If a system has a large number of queues, then this forking operation may result in the **lpd** server exhausting the process resources. To control this, the `max_servers_active` value restricts the number of active children to the specified value. If this value is 0, then 50% of the maximum system processes value will be used.

Due to the limits on the number of processes, there may be times when a job is placed in a queue, but the **lpd** server is unable to start handling the job. When all of the children of the main **lpd** server have exited, the server starts a timer. After `lpd_poll_time` seconds, it will scan the queues, looking for jobs to process, and starts a process to service them. If it does not find any jobs it remains idle.

The `lpd_force_poll` flag causes the server to periodically poll the queues. This is useful when there is a high possibility that jobs could fail to be printed due to high loads on the server.

15.5. Opening the Output Device

Options used:

- `achk` FLAG *Accounting check at start*
- `af`=*Accounting File*
- `ar` FLAG *Remote printer accounting enabled*
- `as`=*Accounting at start*
- `connect_grace`=*Time between jobs*
- `connect_interval`=*Connection interval*
- `connect_timeout`=*Connection timeout*
- `control_filter`=*Control file filter*
- `ff`=*form feed*
- `fo` FLAG *form feed on open*
- `la` FLAG *Local printer accounting enabled*
- `ld`=*leader on open (initialization string)*
- `lk` FLAG *Lock IO device*
- `lp`=*IO device pathname*
- `nb` FLAG *Nonblocking device open*
- `network_connect_grace`=*Interval in secs between jobs*
- `of`=*of filter*
- `retry_econnrefused` FLAG *Retry if open failed*
- `retry_nolink` FLAG *Retry if open failed*
- `rm`=*the remote machine to send the job to*
- `rp`=*the remote print queue to send the job to*
- `rw` FLAG *device opened RW flag*
- `server_tmp_dir`=*temporary directory*

Sequence of Operations:

1. During the server operations, it will try to create temporary files in the print queue spool directory. If this is not desirable, it will create them in the `server_tmp_dir` directory.
2. If the accounting file specified by `af` exists, it is opened (af_fd) and the af_fd is passed as file descriptor 3 to all filters. If the `af` value has the form `af=|/program` then the program is started and the program STDIN is used as af_fd. If the `af` value has the form `af=host%port`, then a TCP/IP connection to the corresponding port on the remote host is made and the port used as af_fd. In the latter two cases, the filter STDIN (file descriptor 0) is actually opened read/write, and is used when information is needed from the accounting filter or remote server. See Accounting Printcap Options for more information on the **LPRng** accounting support.
3. If `la` (local accounting) is true and we are printing a job or `ar` (remote accounting) is true and we are transferring a job, the `as` value is examined. If it is a filter (program) specification, then the program is started with its STDIN attached to `/dev/null`, STDOUT will be read by the print spooler, STDERR output will be written to the error log. The lpd program will wait until the accounting filter program terminates, and examine the error code for action, as for the other filters (see errorcodes below). If the exit status is 0, (JSUCC) then the printing process will continue, if JHOLD the job will be held, if JREMOVE the job will be removed, if JFAIL the job processing will terminate with a JFAIL indication, otherwise the job processing will terminate with a JABORT indication.
4. If the accounting filter exited with a JSUCC (no error code) and the `achk` (accounting check) flag is set, the line read from the accounting filter STDOUT will be examined. This line should be `accept`, `hold`, `fail`, `remove`, otherwise the job processing terminates with a JABORT indication. An `accept` will allow the job to be printed, `hold` will hold the job, `fail` will cause the job to fail, `remove` will cause the job to be removed.
5. If the `connect_grace` value is non-zero and the server is opening a device or `network_connect_grace` is non-zero and a network connection is being made, the server will pause the specified time. This is to accommodate devices which need a recovery time between jobs.
6. The `lp` option is checked to determine the type of IO device.

Format	Meaning
`/pathname`	Absolute pathname of IO device
`pr@host`	transfer to `pr` on remote `host`
`host%port`	open a TCP/IP connection to port on host. host can be name or IP address
`|filter`	run the filter program; it STDIN will be used as device

7. The IO device specified by `lp` is opened write-only or read-write if the `rw` flag is true, and the resulting file descriptor is io_fd. If the `nb` flag is set, a non-blocking open will be done as well. If the `lk` (lock device) flag is true, the device will be locked against use by other **lpd** servers.
8. If a `host%port` combination, a TCP/IP connection will be opened to the remote port and the connection will be used as io_fd.
9. If a filter program is specified, the filter program will be run and the STDIN of the filter will be used as the device file descriptor.
10. If a `rp@rm` combination, or none of the above combinations are true and the `rm` and `rp` values are

non-zero, then the job will be transferred to a remote printer. The type of operation will be a job transfer, rather than printing operation.

11. If the `connect_timeout` value is non-zero, a timeout is setup for the device or socket open. If the device or connection open does not succeed within the timeout, then the open operation fails.
12. If a connection is to a network address (i.e. - `connect()` system call) and the connection attempt fails with an ECONNREFUSED error, if the `retry_econnrefused` flag is set then the connection attempt is retried, but this time using an alternative port number. See RFC1179 for details. This is repeated until all of the possible originating port numbers are exhausted.
13. If the open or connect operation fails, and the `retry_nolink` flag is set, then the server will pause for a minimum of `connect_grace` plus a multiple of `connect_interval` seconds based on the number of attempts before retrying the open operation. Note that the interval may increase as the number of attempts increases.
14. If printing a job and the `:of` filter is specified, it is created with its STDOUT (fd 1) attached to the io_fd. Its stdin (of_fd) will be used in the steps listed below. If there is no `:of` filter, then the of_fd value will be the io_fd descriptor.
15. If transferring a job and the `control_filter` option is specified, then the program specified by the `control_filter` value will be run. It will have its STDIN set to the control file, and its STDOUT output will be used as the new value of the control file to transfer to the remote host. See Filter Command Line Options and Environment Variables for details of options passed to the control filter, and errorcodes for the exit codes of the filter.
16. If the operation is a job transfer, the operation proceeds as outlined in RFC1179, and then the Normal Termination operations are carried out.
17. If the operation is a print operation and the `ld` (leader on open) value is provided, the string is translated (escapes removed) and written to the of_fd file descriptor.
18. If the `fo` (form feed on open) flag is true, then the `ff` (form feed) string is translated (escapes removed) and written to the of_fd file descriptor.

15.6. Printing Banners

Options used:

- `ab` FLAG *Always print banner (default FALSE)*
- `be`=*End banner generator program*
- `bl`=*Short banner line format*
- `bp`=*Banner generator program*
- `bs`=*Start banner generator*
- `generate_banner` FLAG *Generate banner for forwarded jobs*
- `hl` FLAG *Banner (header) Last*

- `of`=*Banner and File Separator Filter*
- `sb` FLAG *Short banner (default FALSE)*
- `sh` FLAG *Suppress header (banners) (default FALSE)*

Banner printing is one of the more complicated configuration options of **LPRng**. This is due mainly to historical evolution of the software, as well as a lack of a well defined standard for filter responsibilities. In the original BSD print spoolers, the philosophy was that banner printing should be delegated to the filters, as they were the most aware of the capabilities of the printers. This required an *out of band* method to convey banner printing information to the filter, and resulted in a complicated interface. The original interface was:

1. The filter doing banner printing was invoked as a special `:of` filter, or passed a special flag.
2. The print spooling software would send a special *single line* of information telling it what the banner information should be. Note that this line was never documented except for the source code, and was inconsistent from version to version. Also, there was no indication of what to do with additional lines, if any.
3. The filter would generate the banner, discard the line, and then pass other lines to the output device.

Adding to the confusion, the original print spoolers had a `:sh` (suppress header or banner) flag, which was supposed to suppress banner printing. It did this by having the print spooler not generate the magic banner information line.

A more sophisticated banner printing system would allow the print spooler software to generate the banner, and would then have the `:of` filter act as a pass through. Thus, we need configure the `:of` filter NOT to use the first line as banner printing information, and to pass through all information to the device.

Complicating this whole mess is the `ld` (leader option) and `tr` (trailer option) which is a string sent to the output device (`:of` filter) when the device (filter) is initialized or terminated. This can sometimes be interpreted as the banner line, leading to unexpected results.

Sequence of Operations:

1. If the `sh` (suppress header) flag is true, no banner is printed, and the actions in this section are skipped. No *banner information line* is generated for the `:of` filter, and no banner printing program is invoked. If there is an `:of` filter and it is expecting such a line and you have `ld` or `tr` information then you may get unexpected results (actually, catastrophic failure is a better term, but I digress).
2. If the `hl` (header last) flag is true the banner is printed at the end of the job and the actions in this section are done at the end of the job.
3. If the user does not want banner pages she can use the `lpr -h` option. This will cause the **lpr** program to delete the `L` (banner name) line in the control file. If there is no `L` line in the control file and `ab` (always print a banner) is false (the default), then no banner is printed and the other actions in this section are skipped. If `ab` is true and the `L` line is missing then the `N` (user login name) is used; if it is missing as well, then ANONYMOUS is used for the user name.
4. If a banner printing program is specified by `bp`, `bs`, or `be` options, then **LPRng** will invoke the program to generate a banner and then send the generated banner to the printer via the `:of` filter.

The banner printing program will be invoked using the standard filter command line flags (see Filter Command Line Options and Environment Variables for details), with is STDIN attached to /dev/null and STDOUT attached to a file to hold the output banner.

5. If no banner printing program is specified and the `sb` (short banner) option is TRUE (default is true), then the `bl=...` (banner line) option value is expanded and sent to the `of_fd` (`:of` filter or device. The default `bl` value is: `bl=$-'C:$-'n Job: $-'J Date: $-'t`. Using our example, this will get translated to:

```
papowell:A Job: file1 file2 Date: Thu Nov 27 23:02:04 PST 1997
```

6. If no banner printing program is specified and we have `sb@` (no short banner) then we skip banner generation, i.e. - we do *not* send a banner generation line to the output (`:of` filter).
7. If the queue is a normal forwarding queue, then the `generate_banner` option will invoke the `bp`, `bs` or `be` program as appropriate to create a banner page file which is then made the first (default) or last (`hl` flag or `be=...` present) file in a job. This option has no effect in other types of queues. See the *translate_format* option as well.

15.7. Printing Job Files

Options used:

- `Xf`=*Format Filter*
- `sf` FLAG *Suppress Form Feed Separators*
- `if`=*Default F Format Filter*
- `pr`=*pr formatting program*
- `send_job_rw_timeout`= *print job read/write timeout*
- `send_query_rw_timeout`= *status query operation read/write timeout*
- `sf` FLAG *Suppress form feed between job files*

Sequence of Operations: for each job in listed in the control file, the following operations are done in turn.

1. If there is an `:of` filter present, the suspend string `\031\001` is written to of_fd and the no further action is taken until the of filter is suspended.
2. The control file line for the job is examined, and the first letter of the data file specification is used as the format.
3. If the format is `p`, the job is first processed by the program specified by the `pr` program, and the program output used as the print file.

4. If the format is `f`, `l`, or `p` then the `:if` filter is used, otherwise the keyword `Xf` is used. Note that certain formats such as `p, a, l`, may not be used as formats.
5. The filter program is started with an appropriate set of command line options (see Filter Command Line Options and Environment Variables), and with its STDOUT attached to the printing device (io_fd), STDERR to a pipe which results in the output being written to the status file. If debugging is enabled, then the STDERR output is also written to the error log file (lf).
6. When doing a read/write operation to a device or remote system, a timeout can be specified. When doing a print or job transfer operation, the `send_job_rw_timeout` value is used. When doing a status or query operation, the `send_query_rw_timeout` value is used. If a write or write operation does not complete within the specified timeout seconds, then we have an error condition and job processing or the query operation is terminated with JFAIL status. If the timeout value is 0, then no timeout is done.
7. **lpd** will then wait for the filter to exit. The following exit codes are used by **lpd**:

```
Key       Value   Meaning
JSUCC     0       Successful
JFAIL     1, 32   Failed - retry later
JABORT    2, 33   Abort - terminate queue processing
JREMOVE   3, 34   Failed - remove job
(Unused)  4, 35
(Unused)  5, 36
JHOLD     6, 37   Hold this job - reprint later
JNOSPOOL  7, 38   No spooling to this queue
JNOPRINT  8, 39   No printing from this queue
JSIGNAL   9,  40   Killed by unrecognized signal
JFAILNORETRY 10, 41 Failed, no retry
Other             Abort - terminate queue processing
```

8. If the filter exit status was JSUCC (0), or no error indicated, then processing will continue otherwise the job termination takes (see Abnormal Termination).
9. If the `:of` filter is present, then it is reactivated with a `kill -CONT` signal.
10. The the `sf` (suppress FF print file separators) is turned off a form feed is sent between each file of a job.

15.8. Printing Banner At End of Job

Options used:

- `hl` FLAG *Header (Banner) Last*

The actions taken in this step are identical to those for the Printing Banner, with the exception that the `be` (end banner program) is used to select the banner generation program rather than the `bs` (start banner program).

If we have `hl` true, then we print a banner at the end of the job, rather than start.

15.9. Normal Termination

Options used:

- `fq` FLAG *Form Feed on Close*
- `la` FLAG *Local Printer Accounting*
- `tr`=*Trailer on Close*
- `ae`=*Accounting at end*
- `save_when_done` FLAG *Save when done*
- `save_on_error` FLAG *Do not delete on error*
- `done_jobs=N`*Save status of last N jobs*
- `wait_for_eof` FLAG *Wait for EOF before closing device*
- `socket_linger`*socket linger timeout*
- `half_close` FLAG *use shutdown() and not close()*

Sequence of Operations:

1. If we are printing and the `fq` flag is set and the `sf` (suppress interfile FF) flag is set, then the `ff` (form feed) string will be interpreted and sent to the of_fd.
2. If we are printing, the `tr` (trailer) string will be interpreted and sent to the of_fd.
3. If printing and the `la` (local printer accounting) flag is set or transferring a job and the `ar` (remote accounting) flag is set, the `ae` is examined and accounting is done as described for the *as field.*
4. If the `:of` filter is present, its STDIN is closed, and the **lpd** server waits for it to exit. The exit status is used as described above.
5. If the device is a socket or network connection, the socket linger time is set to `socket_linger` value if nonzero.
6. If the `half_shut` flag is set, then a `shutdown(fd,WR_DONE)` will be done on the connection. This tells the TCP/IP stack that all data transmission has been completed. Errors or other information can still be read from the connection. If the `half_shut` flag is clear, then a `close(fd)` will be done and no errors or other information will be read.
7. If the `wait_for_eof` option is true (default) then a read is done on the connection until an EOF is found. The device (io_fd) is then closed.
8. The job is marked as completed in the spool queue.

9. If the `save_when_done` flag is clear and the `done_jobs` and `done_jobs_max_age` values are zero (0), the job is removed.
10. If the `done_jobs` or `done_jobs_max_age` values are nonzero, the spool queue is periodically checked and for an excess number of jobs or jobs with old status. This action is suppressed if either the `save_when_done` or `save_on_error` flag is set.

15.10. Abnormal Termination

Options used:

- `mail_from`=*Mail from user name*
- `mail_operator_on_error`=*Mail to operator on error*
- `send_try`=*Maximum printing or transfer attempts*
- `save_on_error` FLAG *Do not delete on error*
- `done_jobs=N`*Save status of last N jobs*
- `done_jobs_max_age=N`*Remove status when older than N seconds*
- `send_failure_action`=*Action on Failure*
- `sendmail`=*sendmail path name and options*
- `stop_on_abort` FLAG *Stop processing queue on filter abort*

If the job processing terminates abnormally, the following sequence of events occurs:

1. The job is marked as having an error during processing.
2. The **lpd** server will attempt to kill all filters and other associated processes by sending a SIGINT and SIGCONT (`kill -INT` and `kill -CONT`) to them.
3. If there is a `mail_operator_on_error` value, the specified operator will be mailed an error indication. The `sendmail` option specifies the pathname of the *sendmail* program and the options needed to have it read mail addresses from its standard input. For example, `sendmail=/usr/sbin/sendmail -oi -t` is a commonly used set of options.
4. The `mail_from` value specifies the user name used for mail origination. If not specified, the default is to use the print spool queue or printer name.
5. If there is a `send_failure_action` specified, then it is decoded and the corresponding action taken. If the value is `remove`, `hold`, `abort`, or `retry`, then the job is removed, held, aborted, or retried. If the value is `|/program`, the program is executed and the number of attempts are written to the filter STDIN. The exit status of the filter will be used to determine the consequent actions. That is, JSUCC (0) will be success, and the standard success action will be taken; JFAIL will cause retry, JREMOVE will cause the job to be removed, JHOLD will cause the job to be held, JABORT or other status will abort processing.

6. If the status is ABORT and the `stop_on_abort` flag is set, then further processing of jobs is terminated. The job is not removed from the queue.
7. If the error status indicates removal, the `save_on_error` flag is clear, and the `done_jobs` and `done_jobs_max_age` values are zero (0), then the job is removed from the spool queue.
8. If the error status indicates that no further operations should be performed on the queue, then the **lpd** server will stop processing jobs.
9. If the error code indicated that the job should be retried, and the `send_try` value is 0 or the number of attempts is less than the `send_try` value, then the job is retried. Between each attempt to transfer a job to a remote site. This pause will double after each attempt, reaching a maximum of `max_connect_interval` seconds. If `max_connect_interval` is 0, there is no limit on the interval value.

15.11. Forwarding Jobs

Options:

- `bk`*Berkeley compatible control file*
- `bq_format=`*format of filtered output*
- `lpd_bounce`*filter job and transfer output*
- `control_filter=`*Control file filter*
- `control_file_line_order=`*Control file line order*
- `nline_after_file`*N line after data file*
- `send_data_first`*send data files first*

If a spool queue is doing store and forward operations, then rather than printing a job the control files and data files are sent to the remote printer. In order to do this, the following items must be arranged.

- If necessary, the job must be processed by filters on the local host.
- The control file must be prepared and updated according to the requirements of the remote site.
- A connection must be established to the remote site.
- The data files and control files must be transferred to the remote site.

One of the more serious problems is when a print spooler (LPR) program does not generate print jobs in a manner compatible with a remote system. While **LPRng** performs checks for improper implementations of RFC1179, it will try to accept a job even under the most severe abuse of the protocol. However, other spoolers are not so forgiving. Some spoolers require that the contents of the control file be in *exactly* the order that the original 1988 BSD **lpr** software generated them. While some entries can be missing, all the entries present in the file must be in an explicit order.

The `bk` (Berkeley **lpd** compatible control file) option causes **lpr** and **lpd** to reformat the control file, removing objectionable entries. The control file of a job being sent to a remote printer will have its control file entries restricted to letters in (and the same order) as HPJCLIMWT1234. You can use the `control_file_line_order` option to specify an even more restricted set, and use the `nline_after_file` option to have the file information line (`N` value) come after the data file entry.

However, there are some very odd commercial implementations that require *more* information than is present. To assist with this, the `control_filter` option can be used. This specifies a program that will process the control file before it is sent to a remote destination. The `control_filter` program is run with the standard set of filter options. STDIN is attached to the control file and the STDOUT will be used as the control file value sent to the remote host.

The exit code of the `control_filter` is used to determine whether to proceed in processing. See Errorcodes for details.

Sequence of Operations:

1. A copy of the control file information is made and the copy will be modified during processing, rather than the original.
2. If the `lpd_bounce` option is specified then a temporary file is created and the job is printed using the procedures for printing to a device, but to the file. This includes all of the filter operations, banners, and so forth. The working copy of the control file is set to have the temporary file as the data file to be sent to the remote destination, and the data file format is set to the `bq_format` value.
3. The control file is rewritten according to the requirements of the routing information, if any. For each destination in the routing information and each copy, a new job identifier value will be generated.
4. The control file is rewritten according to the `bk` and `control_file_line_order` options. If a control filter is specified, the control filter program is run and the output of the program is used as the new control file.
5. A connection is made to the remote host, and the data and control files are transferred to the remote host using the RFC1179 protocol. If the `send_data_first` option is specified the data files are sent first.
6. If the job was sent successfully, the job status is updated in the same manner as for a printed job.

15.12. Debugging

Options used:

- `debugging=`*debugging options*
- `full_time` FLAG *full or extended time format*
- `ms_time_resolution` FLAG *millisecond time resolution*
- `syslog_device=`*syslog alternative device*
- `use_info_cache` FLAG *cache printcap and other information*

The **LPRng** software has a very powerful debugging capability. Since most printing problems occur on remote systems where it is impossible to run debuggers, and since most systems do not do core dumps of SETUID ROOT programs, the **LPRng** software provides a very verbose set of log file trace messages.

First, serious errors or other information are logged using the `syslog()` facilities. If these are not present on a system, then the messages are logged to the device specified by `syslog_device`.

For client programs, the debugging options are specified on the command line and output is directed to STDERR. For the **lpd** server, debugging commands can be specified on the command line OR as the `db=options` printcap value. Output is directed to the log file (`lf` option value, default log).

A typical debug entry has the format `2,network+1,database`. This sets the general debugging level to 2, network debugging to 1 and the database debugging level to the default. The following debugging options and levels are supported.

- nnn - general purpose debugging level
- network - network debugging
- database - database debugging
- receive - job or command reception debugging
- print - detailed job printing debugging

The `full_time` flag forces the logging and other information which has timestamps to have a full (year, month, day, etc.) timestamp. The `ms_time_resolution` flag forces millisecond time resolution in the time stamp.

The `use_info_cache` (default ON) causes **lpd** to cache printcap and configuration information. This is desirable except when trying to change values in printcap files and test the results. By using `use_info_cache@` in the configuration information, you can get immediate responses. Also, see lpc reread for another method.

Chapter 16. Filters

This section gives an overview of how **LPRng** uses filter programs, and gives a detailed discussion of how the printcap options and filters interact.

16.1. Filter Functions

Print filters are one of the most powerful tools in BSD-style printer systems.

In general UNIX terms, a *filter* is a program that takes its input file(s), does something with it, and sends the result to its standard output. Most UNIX utilities are designed as filters. (But since you are a system manager, you should already know that :))

In the context of a BSD-style print spooler (and also **LPRng**), the term *filter* refers to a program that processes file while it is being transferred to a printer.

The filter is executed with STDIN reading from the file to be printed STDOUT to the printer device or a temporary file. STDERR (file handle 2) is redirected to the status file, and file handle3 to an accounting file or program.

A filter can be as simple as a LF to `CR/LF` translator, or it can incorporate a complete accounting system, automatic file type translations, or even redirect the job to another printing system.

The **lpf** filter supplied as part of the **LPRng** distribution is a a very simple CR to CR/LF conversion filter. The **ifhp** filter provides support for more complex PostScript, PCL, and text printers.

16.2. Filter Exit Codes

When a filter exits, the exit code value is used by the parent process to determine what actions to take. Since filters are used in several places in the printing process, not just to do format conversion, there is a large number of recognized exit values.

```
Key       Value   Meaning
JSUCC     0       Successful
JFAIL     1, 32   Failed - retry later
JABORT    2, 33   Abort - terminate queue processing
JREMOVE   3, 34   Failed - remove job
(Unused)  4, 35   (Unused)
(Unused)  5, 36   (Unused)
JHOLD     6, 37   Hold this job - reprint later
JNOSPOOL  7, 38   No spooling to this queue
JNOPRINT  8, 39   No printing from this queue
JSIGNAL   9,  40   Killed by unrecognized signal
JFAILNORETRY 10, 41 Failed, no retry
Other             Abort - terminate queue processing
```

16.2.1. JSUCC

A zero or JSUCC exit value always indicates success; a non-zero exit value indicates failure or a problem condition and requires special handling by the parent process.

16.2.2. JFAIL

When printing or performing some action that can be repeated, such as connecting to a remote printer, a 1 or JFAIL status indicates a transient failure condition. Depending on various configuration options, the printing or other operation can be retried.

16.2.3. JABORT

The 2 or JABORT is a more serious error, and indicates that there is no expectation that the operation would succeed if retried. It may also indicate that no other similar operation should be performed. Jobs whose print filters exit with JABORT are usually unprintable, and by default are removed from the print queue.

16.2.4. JREMOVE

The JREMOVE status indicates that the job should be removed from the print queue. This is a refinement of the JFAIL and JABORT status. The job is usually unconditionally removed from the print queue, even if it is normally kept in the queue for reprinting. This status is usually returned by filters which are responsible for permission checking and is returned when the user has no permission to print.

16.2.5. JHOLD

The JREMOVE status indicates that the job should be held and reprinted at a later time. This status is returned by various filters during the processing of a job, and usually indicates that the resources needed for a job are not available. Held jobs need to be explicitly released by the administrator.

16.2.6. JNOSPOOL and JNOPRINT

The JNOSPOOL and JNOPRINT are used as part of the management of load balancing queues and the *check idle* filter. **LPRng** has the ability to run a program to check to see if a spool queue is available for printing on a dynamic basis. If the filter that does this checking exits with JNOSPOOL or JNOPRINT then jobs should not be sent to the spool queue.

16.2.7. JSIGNAL

This status is usually returned when the exiting process is terminated by a signal or abort, and does not exit using the `exit` facility. It is usually handled like a JABORT exit status, and is the indication of a

severe and possibly non-restartable system failure.

16.2.8. JFAILNORETRY

This code is used under an extremely odd set of circumstances and was used to support a sophisticated print retry system.

Normally when a print filter or other filter returns this code, it is treated as JFAIL. The job is marked as having an error condition and is not `immediately` retried. Other jobs can then be tried for printing in the queue. It is not removed from the print queue, but marked as `unprintable`.

When a *round-robin retry* print scheduling algorithm is used, if there are no other jobs available for printing then the jobs that failed with JFAILNORETRY are retried. Thus, jobs that are submitted go to the head of the queue for printing, and jobs that are pending for repeat are printed after them. This algorithm is deprecated, and that the details of this algorithm are undocumented.

16.2.9. Other Values

If a filter exits with other than the indicated value, or a value inappropriate for its purpose, then the result is treated like JABORT.

16.3. Print Job Formats

Options used:

- `:if`, `cf`, `df`, `gf`, `nf`, `:of`, `rf`, `tf`, `vf`, `Xf`, *Filter programs*

LPRng has inherited a set of so-called '*print formats*' from its BSD ancestor. The format was used to specify the type of file that was being printed. The **lpd** server used the print format to select the filter for processing the file. The de*f*ault format is `f`.

The user can specify the format (i.e., the file type) by giving the appropriate option to **lpr**:

- `-b` or `-l`: Binary (literal) file. No processing should be done. The `l` format is recorded as the file format.
- `-c`: cifplot(1) output.
- `-d`: TeX DVI file.
- `-g`: Output from the plot(3X) routines.
- `-n` or `-t`: (di)troff output.
- `-p`: Text file that should be pre-processed by the `pr` command, and then by the standard text filter.
- `-v`: Benson Varian raster image.

Alternatively, one can also use `-Fx`, where `x` is the format specifier. (E.g., `-Fc` instead of `-c`.) This last form also allows you to use other (non-standard) format specifiers.

The filter for format `X` is the value for the `Xf` printcap option, with some minor exceptions. The following `Xf` options have a pre-defined meaning.

- `:if` The `f` format filter, i.e. - for the default `f` format. All print jobs are passed through this one, unless another format is selected.
- `cf` Cifplot data filter (for `-c` format).
- `df` Filter for DVI files (`-d`).
- `gf` Graph data filter (`-g`).
- `nf` Ditroff data filter (`-n`).
- `:of` This filter is used for processing the (optional) banner at the start and/or end of the print job, and also for the interjob separators. See OF Filter for details.
- `rf` Filter for Fortran style files (`-r`).
- `tf` Troff filter (`-t`).
- `vf` (Versatek) raster image filter (`-v`).

16.4. OF Filter

The `:of` filter is used to process banners and job separators. The `:of` filter is responsible for performing appropriate processing of this information and sending to the printer for action.

While the various file filters are invoked on a once per print file basis, the `:of` filter is invoked on a once per print job basis.

This filter is the first one to be started, and should perform whatever specialized device initialization is needed. It should also do whatever accounting procedure is desired for start of job accounting.

The `:of` filter will be given any banner printing or job separation information for a job. As part of its operation, it can detect a specific string, corresponding to a banner print request, and generate a banner. (See the Job Processing Steps and Printcap Options for details.)

During operation, the **lpd** server will send the special *stop* sequence of `\031\001` to the `:of` filter. The filter must then suspend itself using a `kill -STOP` operation. The **lpd** server will detect that the `:of` filter has suspended itself and then will perform other printing operations.

After the other printing operations have been completed, the `:of` will then be sent a `kill -CONT` signal.

This sequence will continue until all information has been printed, and then the `:of` filter's STDIN will be closed. The filter will then perform whatever cleanup operations are needed, update accounting or other information, and exit.

16.5. lpr -p format

Options used:

- `pr`=*pr program for p format*

The `-p` format is requires filtering the the input files by the `pr` utility and then passing the result through the `:if` filter.

This is widely regarded as a kludge and may not be supported on your print spooler.

16.6. lpr binary (-l) format

The binary (or literal) format is `-l`. The `:if` filter is used to process the file, and is invoked with the `-c` (`c`ancel processing?) flag.

The filter will not modify the file when sending it to the printer, but may apply various setups to the printer.

16.7. Chaining Filters

If a filter command has a pipe (`|`) or IO redirection indicator (< or >) in it, or starts with an open parenthesis (`(`), the filter is run by passing the entire command to the shell specified by the `shell` configuration option. This allows a wide variety of options and operations to be carried out. The `$*` value should be used to have the filter options passed to the correct entry in the filter chain.

For example,

```
lp:
  :filter=( echo "starting `date`" >/var/log/status; /usr/local/ifhp $* )
```

16.8. Filter Command Line Options and Environment Variables

Options used:

- `bk_filter_options`=*Backwards Compatible Filter options*
- `bk_of_filter_options`=*Backwards Compatible OF Filter options*
- `bkf` FLAG *Backwards Compatible Filters*

- `filter_ld_path`=*Filter LD_LIBRARY_PATH environment*
- `filter_options`=*Filter options*
- `filter_path`=*Filter PATH environment*
- `of_filter_options`=*OF Filter options*
- `pass_env`=*Environment variables to copy to Filter environment*
- `pl`=*line count for page*
- `pw`=*column count for page*
- `px`=*pixel width for page*
- `py`=*pixel length for page*

A filter (or program) specification in the **LPRng** printcap database has the form:

```
:option=| [flags] /path [argument | "argument" | 'argument' ]*
:option=[flags]   /path [argument | "argument" | 'argument' ]*
```

The first case is used where the option value can be a string or filter, and the second where a program is always expected. The following procedure is used to run a filter program. Arguments in single or double quotes are passed as a single value, as for a shell.

The sequence of operations to run a filter is as follows:

1. The program must be specified with an absolute path name.
2. By default, the program is run as the user if invoked from a client program such as **lpr**, **lpc**, etc. If invoked from **lpd**, it is run as the user ID specified by the `:user` (default `daemon`) configuration entry.
3. The `filter_path` (default `/bin:/usr/bin:/usr/local/bin`, and `filter_ld_path` (default `/lib:/usr/lib:/usr/local/lib`, configuration options specifies the value of the `PATH` and `LD_LIBRARY_PATH` environment variables.
4. The `filter_path` (default `/bin:/usr/bin:/usr/local/bin`, and `filter_ld_path` (default `/lib:/usr/lib:/usr/local/lib`, configuration option specifies the value of the `PATH` and `LD_LIBRARY_PATH` environment variables. The other enviroment variables are described in **LPRng** ftp mirror sites Filter Environment Variables
5. ROOT Flag. If the ROOT flag is specified the filter is executed with Userid and Effective Userid ROOT (User ID 0). By default it is executed with the `user` and `group` configuration option user and group ids. Running a filter as ROOT is extremely dangerous, and should only be used for programs that require root permissions to open files or make network connections from privileged ports.
6. `$-` or `-$` Flag. This flag suppresses appending options to the filter command line. If the *$-* or `-$` flag is not present, the `:filter_options` or `:of_filter_options` for the `:of` filter values are appended to the filter command line. If the `:bkf` (Berkeley **lpd** filter compatible flag) is `TRUE` then the `:bk_filter_options` and `:bk_of_filter_options` values are used instead of the `:filter_options` and `:of_filter_options` values.

Table 16-1. Print Filter Command Line Options

Option	DefaultValue
Option	DefaultValue
`filter_options`	$C $F $H $J $L $P $Q $R $Z $a $c $d $e $f $h $i $j $k $l $n pr $s $w $x $y $-a
`of_filter_options`	(same as `filter_options`)
`bk_filter_options`	$P $w $l $x $y $F $c $L $i $J $C $0n $0h $-a
`bk_of_filter_options`	$w $l $x $y

7. By default, for programs that are not being invoked as print job file filters, the `filter_options` arguments are added. For print job filters, if the `:bkf` flag is set, then the `bk_filter_options` and `bk_of_filter_options` entries are used. The default `bk` filter options are the same as originally used with the BSD **lpr** filters. For the `:of` filter, either the `of_filter_options` or `bk_of_filter_options` arguments will be added.
8. The program arguments will then be scanned and interpreted. Arguments of the form `$letter` will be translated into values from the print job control file and/or printcap entry. The letters have the following meaning:

Table 16-2. Filter Command Line Options and Values

Option	Purpose or Value
`a`	printcap `af` (accounting file name)
`b`	job size (in K bytes)
`c`	binary file (`l` format for print file)
`d`	printcap `cd` or `sd` entry
`e`	print job data file name (currently being processed)
`f`	print job original name when spooled for printing (N info from control file)
`h`	print job originating host (H info from control file)
`i`	indent request (I info from control file)
`j`	job number in spool queue
`k`	print job control file name
`l`	printcap `pl` (page length)
`m`	printcap `co`
`n`	user name (L info from control file)
`p`	remote printer (when processing for bounce queue)
`r`	remote host (when processing for bounce queue)

Option	Purpose or Value
s	printcap sf (status file)
t	time in common UNIX format
w	printcap pw (page width)
x	printcap px (page x dimension)
y	printcap py (page y dimension)
F	print file format
P	printer name
S	printcap cm (comment field)
Capital letter	Corresponding line from control file
{key}	printcap value for key

9. If there is no value for the specified argument, then the argument is removed from the list. If there is a value, the actual form of the substitution is controlled by additional flags as follows.

Table 16-3. Filter Command Line Option Format

Form	TranslatedValue
$x	'-x*value*'
$-x	'*value*'
$0x	-x '*value*'
$'x	-x *value*

Each entry in quotes is treated as a single value, as in /bin/sh. The $'x does not quote the value. Combinations of the various flags are allowed. For example, $-x would simply substitute the value for x, and then pass the whitespace separated components as individual arguments. This last form is useful for adding in additional flags on the command line.

10. The command line is parsed, metacharacters are ruthlessly stripped from all arguments and pathnames and replaced by _ (underscores), and an argument list suitable for the execve system call is formed.
11. A sanitized environment is set up for the program execution, with the following environment variables.

Table 16-4. Filter Environment Variables

Variable Name	Meaning
CONTROL	control file image
HF	hold file image
DATAFILES	list of data file names
HOME	Home directory (client only)
IFS	" \t"
LD_LIBRARY_PATH	:filter_ld_path configuration information
LOGDIR	Home directory (client only)

Variable Name	Meaning
LOGNAME	L control file line
PATH	filter_path configuration information
PRINTCAP_ENTRY	printcap information
SHELL	:sh configuration information (default /bin/sh)
SPOOL_DIR	:sd printcap information
TZ	Time zone
USER	User name (client only)

12. If the filter is to be run by a client program such as **lpr**, then the environment variables specified by the pass_env configuration or printcap option will be extracted from the environment, have any metacharacters removed, and then placed in the environment variable list. Commonly, the PGPPASS, PGPPASSFD, and PGPPATH are specified.
13. The program is started, with STDIN, STDOUT, and STDERR attached to the appropriate files or file descriptors. If none is specified, then they are attached to /dev/null.

16.9. LPRng Supported Filters

There already exists a large library of ready-to-use filters. Some of them have **LPRng**-specific versions, which can be found at the **LPRng** ftp mirror sites.

16.9.1. Filter Support Conventions

By convention, most filters are either totally standalone (very rare) or require a set of support files or configuration files. There are two types of configuration files: per print queue configuration information and global configuration information.

Since a print filter executes with the spool queue directory as the current directory, most filters put per print queue configuration information in a file in the spool directory. Some *vintage* filters insist on having these files *hidden* with names such as *.setup*. This can make it difficult for administrators to determine where the configuration files are.

Global configuration files are usually placed in commonly accessible directories such as */usr/local/libexec/filters* and its subdirectories. This allows the **LPRng** administrator to set the privileges on these directories such that only the **lpd** process can access them.

When a filter is invoked, it is passed a large number of options, many of which are totally ignored in filter operation. However, for many purposes it is necessary to provide options to the filters to tailor their operation to the particular spool queue needs.

An alternative to using information in a file is to place options in the printcap entry and have the filter extract them from the PRINTCAP_ENTRY environment variable value. This is much easier to implement, but is specific to **LPRng**.

16.10. lpf

Source code: **LPRng** Distribution

This filter is distributed as part of the **LPRng** source code, and has a very limited functionality. By default, it only translates `\n` to `\r\n` sequences, and detects the OF Filter Stop sequence when invoked as an OF filter.

- Options: `-Tcrlf` - suppress `\n` to `\r\n` translation

16.11. ifhp Filter

Source code: **LPRng** Distribution, ifhp-version.tgz

The **ifhp** filter supports a wide variety of *smart* printers, or to be more specific, printers which support PostScript, PCL or PJL languages. For details on using the **ifhp** filter see the **ifhp** filter documentation for details. The following is a quick set of examples of printcap entries:

```
# network connection to jet direct box,
#   no banners, HP compatible
lp
  :lp=ipaddr%9100
  :filter=/usr/local/libexec/filters/ifhp
#
# banner added,  model information added
#
lp
  :lp=ipaddr%9100
  :ifhp=model=hp4
  :bp=/usr/local/libexec/filters/pclbanner
  :of=/usr/local/libexec/filters/ifhp
  :filter=/usr/local/libexec/filters/ifhp
#
# for a parallel port printer or when you want VERY fast
#  throughput, no pagecounts, error messages, etc.  The
#
lp
  :lp=/dev/lp0
  :ifhp=model=hp4,status@
  :filter=/usr/local/libexec/filters/ifhp
```

Chapter 17. Permissions and Authentication

The contents of the `/etc/lpd.perms` file are used to control access to the **lpd** server facilities. The model used for permission granting is similar to packet filters. An incoming request is tested against a list of rules, and the first match found determines the action to be taken. The action is either ACCEPT or the request is granted, or REJECT and the request is denied. You can also establish a default action.

The following is a sample `lpd.perms` file.

```
# allow root on server to control jobs
ACCEPT SERVICE=C SERVER REMOTEUSER=root
REJECT SERVICE=C
#
# allow same user on originating host to remove a job
ACCEPT SERVICE=M SAMEHOST SAMEUSER
# allow root on server to remove a job
ACCEPT SERVICE=M SERVER REMOTEUSER=root
REJECT SERVICE=M
# all other operations allowed
DEFAULT ACCEPT
```

Each line of the permissions file is a rule. A rule will ACCEPT or REJECT a request if all of the patterns specified in the rule match. If there is a match failure, the next rule in sequence will be applied. If all of the rules are exhausted, then the last specified default authorization will be used.

The sense of a pattern match can be inverted using the NOT keyword. For example, the rules with `ACCEPT NOT REMOTEUSER=john,bill` succeeds only if the REMOTEUSER value is defined and is not `john` or `bill`.

Each entry in a rule is a keyword which has is assigned a value or list of values followed by an optional set of patterns that are matched against these values. The following table is a summary of the available keywords.

Table 17-1. Permission Keywords and Purpose

Keyword	Match
DEFAULT	default result
SERVICE	Checking lpC, lpR, lprM, lpQ, and Printing
USER	P (logname) field name in print job control file.
REMOTEUSER	user name in request from remote host.
HOST	DNS and IP address information for the H (host) field name in print job control file
REMOTEHOST	DNS and IP address information for the connection from the remote host making the request
IP	Alias for HOST

Keyword	Match
REMOTEIP	Alias for REMOTEHOST
REMOTEPORT	Originating TCP/IP port for the connection from the remote host making the request
PORT	Alias for PORT
UNIXSOCKET	Connection is on a UNIX socket, i.e. from localhost
SAMEUSER	USER and REMOTEUSER matches
SERVER	request originates on lpd server
FORWARD	destination of job is not host
REMOTEGROUP	REMOTEUSER is in the specified group or netgroup in the **lpd** server group database.
GROUP	USER is in the specified group or netgroup in the **lpd** server group database.
LPC	LPC command in the LPC request.
CONTROLLINE	match a line in control file
AUTH	authentication type
AUTHUSER	authenticated user
AUTHSAMEUSER	same authenticated user
AUTHFROM	authenticated forwarder
AUTHJOB	authenticated job in queue
AUTHCA	SSL signing certificates for job

17.1. Permission Checking Algorithm

Options used:

- `default_permission`=*Default Permission (accept)*

The **lpd** server uses the following algorithm to do permission checks.

1. The configuration information initially establishes a default permission using the `default_permission` configuration value. This is used if an explicit permission is not determined by the other steps in this algorithm.
2. Each line of the permissions file is a lists of tests (patterns) and a permission value that is used if all of the tests (patterns) on the line are successful. A DEFAULT line sets the default result if all lines fail.
3. Each line is executed in sequence until a match is found. The first matching line terminates the permission checking and the corresponding permission value is used.

4. Each keyword has a value (or set of values) that are matched against a set of patterns. If the keyword does not have a value (or the *null* value) then the match will fail. Initially, all the keywords have a `null` value.
5. When a connection is received by the **lpd** server, REMOTEHOST and REMOTEPORT are set to the the IP addresses and hostnames, and the TCP/IP port of the host originating the IP address respectively. REMOTEIP and IFHP are aliases for REMOTEPORT and PORT is an alias for REMOTEPORT and are provided for backwards compatibility with older versions of **LPRng**. If the connection was on a UNIX socket, then the UNIXSOCKET flag is set. For example, a request originating from `10.0.0.2`, port 1011 would set REMOTEIP to 10.0.0.2 and PORT to 1011.
6. The REMOTEHOST value is set to the result of doing a reverse DNS lookup on the REMOTEIP address. This value is the list of names *and* ip addresses in standard IP notation (nnn.nnn.nnn.nnn) that are returned by the lookup. If the DNS lookup fails then the REMOTEHOST value is set to the REMOTEIP value. For example, lookup of 10.0.0.2 would result in the names `h2.private` and `patrick.private`, and the only IP address assigned to it was `10.0.0.2`. The REMOTEHOST value would then be the list `h2.private,patrick.private,10.0.0.2`.
7. The SERVICE value is set to `X` and then the permissions database is scanned for a matching entry. The result is the permission value of the first matching line or the default permission. If the result is REJECT then the connection is closed.
8. Next, a single line is read from the connection. This line contains the request type, the print queue name, and depending on the request type an optional user name and options. The SERVICE value is set to `R, Q, M,` and `C,` for a `lpR`, `lpQ`, `lprM`, and **lpc** request respectively and PRINTER to the print queue name.
9. If the request is for an **lpc** operation, the LPC value is set to the name of the operation. For example, and `lpc lpd` operation
10. If the request contains a user name then REMOTEUSER is assigned the user name.
11. If the request originates from the **lpd** server as determined by the connection arriving from the `localhost` address or an address assigned to one of the network interfaces for this host then the SERVER value is set to true (or matches).
12. If the request is for an authenticated transfer, (see Authentication and Encryption), then the authentication procedures are carried out. After they have been performed, the AUTH value is set to true, AUTHTYPE is set to the name of the authentication method, AUTHUSER to the authenticated identifier of the originator of the request, and AUTHFROM to the authenticated identifier of the originator of the connection.
13. Other matching keywords such as REMOTEGROUP use values set at this time. These are discussed in the next section.
14. The permission database is rescanned, this time to see if there is permission to operate on the specified spool queue. The permission database is first checked to see if the requesting user has control (SERVICE=C) permission. If they do, then they can perform any operation on the spool queue. The scan is then repeated for the actual request.
15. If there is no permission to perform the operation then an error code and messages is returned on the requesting connection.
16. If the operation is for a spool queue or server, no other permissions checking is done. This includes the **lpq** command, and most of the **lpc** commands control queue operations.

17. If the operation is for for individual jobs in a spool queue, then the queue is scanned and job information is extracted for each job in the queue. The USER value is set to the job control file `P` line. The value of the `H` line in the control file is used to perform a DNS lookup, and the HOST value is set to the results of this lookup. IP is an alias for HOST, and is retained for backwards compatibility.
18. The SAMEUSER value is set to true (or match) if the REMOTEUSER value is identical to the USER value. Similarly, SAMEHOST is set to true if the REMOTEHOST value matches the HOST value. See the following sections for other keywords such as GROUP.
19. The permission checking is done for each individual job in a spool queue, and if it succeeds the action is carried out on the job.

These checks are applied on the arrival of a job from an external connection. Unfortunately, there are a set of print spooler implementations that do not handle job rejection due to lack of permissions. These printers will continually and repeatedly attempt to send a job for which there is no printing permission until the job is removed by administrative action. To accommodate these printers, we must accept jobs for printing and then dispose of them. This is done by using the SERVICE=P (printing) checks. These checks are performed *after* the job has been accepted.

1. When a print spool is active and is printing or forwarding jobs, before it processes a job it will read the job control file and set the USER and HOST values as discussed in the previous sections. It will also set the AUTH, AUTHUSER, and AUTHJOB values as well, if the job was spooled by using an authenticated method.
2. The permissions database will be scanned and the resulting permission determined. Note that the values of the REMOTE keys are undefined, and tests using them will have unpredictable effects.
3. If the job does not have permission to be printed, it will normally be removed from the spool queue.

While this model is very simple it can handle a wide range of situations. However, it is really based on the simple *trust* that users will not *impersonate* other users or hosts. If this is not the case, then more elaborate procedures based on encryption and authentication are called for.

There is a problem with permissions checking for **lpq** (SERVICE=Q) requests. Since the user name is not passed as part of the request, it is impossible to use the REMOTEUSER clause to restrict **lpq** operations.

The `SERVICE=R` and `SERVICE=P` facilities are provided to handle problems with print spoolers that do not recognize a *lack of permission* error code, and will indefinitely retry sending a job to the **lpd** server. If this is the case, then the `SERVICE=R` clause can be used to accept jobs, and then the `SERVICE=P` clause will cause the **lpd** server to remove of the job when it is scheduled for printing.

17.2. Rule Matching Procedures

```
[not] key                                  assigned value
[not] key=pattern                          substring match
[not] key=pattern1,pattern2,pattern3,...   glob and exact
```

```
[not] key=IP1/mask1,IP2/mask2,...          IP address
```

Each of the indicated values is matched against a list of patterns. The following types of matches are used:

1. assigned value. The keyword has an assigned value which is true (match) or false (no match). Examples are SAMEHOST and SERVER.
2. substring match. The indicated entry is present as a substring in the pattern.
3. GLOB matches. The pattern is interpreted as a GLOB style pattern, where * matches 0 or more characters, and ? matches a single character, and `[L-H]` specifies a range of characters from `L` to `H`, in ASCII order.
4. IP address match. The address must be specified in the standard `nn.nn.nn.nn` format. The mask must be either an integer number corresponding to the number of significant bits, or in the standard `nn.nn.nn.nn` format. Addresses are compared by doing

   ```
   ( IPaddr XOR IP ) AND mask
   ```

   If the result is 0, then a match results. Note that there may be one or more addresses being checked for; this can occur when a host may have multiple IP addresses assigned to it.
5. integer range match. The pattern has the form `low-high`, where low and high are integer numbers. The match succeeds if the value is in the specified range.
6. Same IP Address Match. This compares two lists of IP addresses; a match is found when there is one or more common addresses.

17.2.1. DEFAULT

```
DEFAULT ACCEPT
DEFAULT REJECT
```

The DEFAULT rule specifies the default if no rule matches. Normally, there is one DEFAULT entry in a permissions file.

```
Example:

DEFAULT ACCEPT
```

17.2.2. SERVICE

Match type: substring

The SERVICE key is based on the type of request.

Key	Request
Key	Request
`C`	LPC Control Request
`M`	**lprm** Removal Request
`P`	Printing
`Q`	**lpq** Status Request
`R`	**lpr** Job Transfer
`X`	Connection Request

Each of the above codes corresponds either directly to the user command, or a set of subcommands.

If you have an LPC request, you can add an `LPC=xxx` clause to refine the permissions checking to allow or disallow **lpc** commands such as `lpc status, printcap, active, .`

```
Example:

# control only from root on server
ACCEPT SERVICE=C SERVER USER=root
REJECT SERVICE=C
# accept all others
ACCEPT SERVICE=*
```

17.2.3. USER

Match type: GLOB

The USER information is taken from the `P` (person or logname) information in the print job control file.

```
Example:

# we allow jobs to be spooled
ACCEPT SERVICE=R
# now we do the checking at print time
ACCEPT SERVICE=P USER=root
REJECT SERVICE=P
```

17.2.4. REMOTEUSER

Match type: GLOB

The REMOTEUSER information is taken from the user information sent with a service request.

Note that one of the flaws of RFC1179 is that an **lpq** (print status) request does not provide a REMOTEUSER name.

Example:

```
ACCEPT SERVICE=C REMOTEUSER=root,papowell,admin SERVER
ACCEPT SERVICE=C LPC=status,lpd REMOTEUSER=admin
REJECT SERVICE=C
```

17.2.5. HOST

Match type: GLOB

The `H` (host) information in the print job control file is used to do a DNS lookup, and the resulting list of names and addresses is used for matching purposes.

Example:

```
# we allow jobs to be spooled
ACCEPT SERVICE=R
# now we do the checking at print time
# allow from our private subnet
ACCEPT SERVICE=P HOST=10.0.0.0/8,*.othernet.com
REJECT SERVICE=P
```

17.2.6. REMOTEHOST

Match type: GLOB

The REMOTEHOST information is obtained by doing a reverse IP name lookup on the remote host IP address and the resulting list of names and addresses is used for matching purposes. If there is no FQDN available, then the IP address in text form will be used.

Example:

```
# allow from our private subnet
ACCEPT SERVICE=R REMOTEHOST=10.0.0.0/8,*.othernet.com
REJECT SERVICE=R
```

17.2.7. REMOTEPORT

Match type: integer range

The REMOTEPORT value is the originating port of the TCP/IP connection. The match succeeds if it is in the specified range.

Example:

```
# require connections to originate from privileged port
ACCEPT SERVICE=X REMOTEPORT=1-1023
REJECT SERVICE=X
```

17.2.8. PORT

Alias for REMOTEPORT.

17.2.9. IP

Alias for HOST.

17.2.10. REMOTEIP

Alias for REMOTEHOST.

17.2.11. LPC

Match type: GLOB

The requested **lpc** command. This allows the following permissions line to be used:

Example:

```
#allow remoteuser admin on server to use LPC topq and hold
ACCEPT SERVICE=C SERVER REMOTEUSER=root
ACCEPT LPC=topq,hold SERVER REMOTEUSER=papowell
REJECT SERVICE=C
```

17.2.12. SAMEUSER

Match type: exact string match

Both the REMOTEUSER and USER information must be present and identical.

Example:

```
# LPC users can do anything
ACCEPT SERVICE=C SERVER REMOTEUSER=root
REJECT SERVICE=C
# allow users who sent jobs from the same host to remove them
ACCEPT SERVICE=M SAMEUSER SAMEHOST
REJECT SERVICE=M
```

17.2.13. SAMEHOST

Match type: Same IP Address

The REMOTEHOST and HOST address lists are checked; if there is a common value the match succeeds.

Example:

```
# allow root on the same host as user
# to remove files
ACCEPT SERVICE=M SAMEHOST REMOTEUSER=root
REJECT SERVICE=M
```

17.2.14. SERVER

Match type: Matching IP Address

One of the REMOTEHOST addresses must be the same as one of the addresses of the **lpd** server host, or must be one of the addresses found by looking up the `localhost` name using `gethostbyname()`.

Example:

```
# allow root on the server full LPC permissions
ACCEPT SERVICE=C SERVER REMOTEUSER=root
REJECT SERVICE=C
```

17.2.15. FORWARD

Match type: Address Match

The list of REMOTEHOST and HOST addresses must not have a common entry. This is usually the case when a remote **lpd** server is forwarding jobs to the **lpd** server.

```
Example:

# do not accept forwarded jobs or requests
REJECT SERVICE=* FORWARD
```

17.2.16. GROUP

Match type: modified GLOB

The USER must be present in one of the groups in `/etc/group` or whatever permissions mechanism is used to determine group ownership which matches the GLOB pattern. If the pattern has the form `@name`, then a check to see if the user is in the named netgroup is done.

```
Example:

ACCEPT SERVICE=P GROUP=admin,@netgroup
REJECT SERVICE=P
```

17.2.17. REMOTEGROUP

The same rules as for GROUP, but using the REMOTEUSER value.

```
Example:

ACCEPT SERVICE=R REMOTEGROUP=admin,@netgroup
REJECT SERVICE=R
```

17.2.18. CONTROLLINE

Match type: GLOB

A CONTROLLINE pattern has the form

```
X=pattern1,pattern2,...
```

X is a single upper case letter. The corresponding line must be present in a control file, and the pattern is applied to the line contents.

This pattern can be used to select only files with specific control file information for printing.

17.2.19. AUTH

Match type: value

If the current transfer or the transfer used to send a job was authenticated, then AUTH is true or matches.

```
Example:

# reject all non-authenticated transfers
REJECT NOT AUTH
```

17.2.20. AUTHTYPE

Match type: glob

If the current transfer or the transfer used to send a job was authenticated, then AUTHTYPE is set to the name of the authentication method.

```
Example:

# require kerberos, pgp, or md5 authentication
REJECT NOT AUTHTYPE=kerberos*,pgp,md5
```

17.2.21. AUTHUSER

Match type: GLOB

The AUTHUSER rule will check to see if the authenticated user identification matches the pattern.

```
Example:

ACCEPT SERVICE=C AUTHTYPE=kerberos* AUTHUSER=admin@ASTART.COM
```

17.2.22. IFIP

Match type: IPmatch, but for IPV6 as well as IPV4

There is a subtle problem with names and IP addresses which are obtained for 'multi-homed hosts', i.e. - those with multiple ethernet interfaces, and for IPV6 (IP Version 6), in which a host can have multiple addresses, and for the normal host which can have both a short name and a fully qualified domain name.

The IFIP (interface IP) field can be used to check the IP address of the interface that accepted the network connection, as reported by the information returned by the accept() system call. Note that this information may be IPV4 or IPV6 information, depending on the origination of the system. This information is used by gethostbyaddr() to obtain the originating host fully qualified domain name (FQDN) and set of IP addresses. Note that this FQDN will be for the originating interface, and may not be the canonical host name. Some systems which use the Domain Name Server (DNS) system may add the canonical system name as an alias.

This entry is deprecated and may not be supported in future releases.

17.3. Permission File Location

Options used:

- `perms_path=` *path*

The `perms_path=` configuration variable specifies the location of the default permissions file. The default value is:

```
perms_path=${sysconfdir}/lpd.perms
```

The `lpd.perms` file can be obtained by running a program, in a similar manner to the `printcap` file. See Filters for details on how the program would be invoked. For example, assume the configuration information specified:

```
perms_path=|/usr/local/libexec/get_perms
```

The **lpd** server will write either a blank line for connection (`SERVICE=X`) and global **lpc** permissions (`SERVICE=C` and `LPC=reread,lpd,default`) or the name of the spool queue to the `get_perms` STDIN, and expects to read permission information from its STDOUT. If the filter method is used, it should always return the complete set of connection (`X`) and control (`C` service values.

17.4. Example Permission File

```
# allow root on server to control jobs
ACCEPT SERVICE=C SERVER REMOTEUSER=root
ACCEPT SERVICE=C LPC=lpd
REJECT SERVICE=C
#
# allow same user on originating host to remove a job
ACCEPT SERVICE=M SAMEHOST SAMEUSER
# allow root on server to remove a job
ACCEPT SERVICE=M SERVER REMOTEUSER=root
REJECT SERVICE=M
# all other operations allowed
DEFAULT ACCEPT
```

In the above sample, we first specify that lp`C` commands from user `root` on the lpd server will be accepted. This is traditionally the way that most lpc commands operate. We also allow anybody to use the `lpc lpd` command. We reject any other **lpc** requests.

We accept lpr`M` requests from the host and user that submitted the job, as well as from root on the server, and reject any others.

Finally, all other types of commands (lpq, lpr) are allowed by default.

17.5. Complex Permission Checking

One of the more useful types of permission checking is to restrict access to your printers from users outside your networks. The IP pattern can specify a list of IP addresses and netmasks to apply to them.

For example `IP=10.3.4.0/24` would match all hosts with the IP addresses `IP=10.3.4.0` to `IP=10.3.4.255`.

Similarly, the HOST pattern can specify a set of hostnames or patterns to match against based on the GLOB notation.

For example `REMOTEHOST=*.private` would match all hosts with a DNS entry which ended with `private`.

The NOT keyword reverses the match sense. For example `REJECT NOT REMOTEHOST=*.private,*.murphy.com` would reject all requests from hosts which did not have a DNS entry ending in `private` or `murphy.com`.

17.6. More Examples

The following is a more complex lpd.perms file.

```
# All operations allowed except those specifically forbidden
```

```
DEFAULT ACCEPT
#Reject connections which do not originate from hosts with an
# address on 130.191.0.0 or from localhost,
# or name is not assigned to Engineering pc's
REJECT SERVICE=X NOT IFIP=130.191.0.0/16,127.0.0.1/32
REJECT SERVICE=X NOT REMOTEHOST=engpc*
#Do not allow anybody but root or papowell on
#astart1.private or the server to use control
#facilities.
ACCEPT SERVICE=C SERVER REMOTEUSER=root
ACCEPT SERVICE=C REMOTEHOST=astart1.private REMOTEUSER=papowell
#Allow root on talker.private to control printer hpjet
ACCEPT SERVICE=C HOST=talker.private PRINTER=hpjet REMOTEUSER=root
#Reject all others
REJECT SERVICE=C
#Do not allow forwarded jobs or requests
REJECT SERVICE=R,C,M FORWARD
# allow same user on originating host to remove a job
ACCEPT SERVICE=M SAMEHOST SAMEUSER
# allow root on server to remove a job
ACCEPT SERVICE=M SERVER REMOTEUSER=root
```

17.7. Authentication and Encryption

One of the major problems in a print spooler system is providing privacy and authentication services for users. One method is to construct a specific set of protocols which will be used for providing the privacy or authentication; another is to provide a simple interface to a set of tools that will do the authentication and/or encryption.

LPRng provides native support for the MIT Kerberos 4 extensions and Kerberos 5 authentication.

LPRng uses the OpenSSL libraries to support SSL authentication and encrypted data transfers.

LPRng has native support for the PGP (Pretty Good Privacy) program and can sign and optionally encrypt command and responses between servers and clients. Due to legal restrictions, an external PGP program must be used for this purpose.

A simple MD5 hash based authentication scheme is also provided as an example to illustrate how new or different authentication methods can be adddd.

Finally, **LPRng** provide a general purpose interface allowing users to insert their own authentication methods, either at the program level or at the code level.

A careful study of the authentication problem shows that it should be done during reception of commands and/or jobs from a remote user and/or spooler. At this time the following must be done:

1. The received command must be checked for consistency, and the remote user and host must be determined.
2. The remote user and host must be authenticated.

3. The command and/or spooling operation must be carried out.
4. The results must be returned to the remote system.

To accomplish these goals, the following printcap entries are used:

- `auth=`*AUTHTYPE* - pgp, kerberos, etc
- `AUTHTYPE_path=`*pathname* - the pathname of a program to be used to support this authentication type
- `AUTHTYPE_id=`*identification* - the identification of the server for the authentication method. For example, the kerberos principal for the server, the PGP key id for the server, and so forth.
- `AUTHTYPE_server_key=`*identification* - location of a file on the server where a key used to unlock or encrypt a message is kept.
- `AUTHTYPE_forward_id=`*identification* - the identification of the remote destination, This is used by the server when forwarding a job to a remote destination. By default, the AUTHTYPE_id value is used by the server as its identification.
- `AUTHTYPE_default_client=`*identification* - when forwarding a job and the job arrived via an unauthenticated method, use this as the default client identification.

17.8. User Identification

When a user logs into a system, they are assigned a user name and a corresponding UserID. This user name is used by the **LPRng** software when transferring jobs to identify the user.

When we look into the problem of authentication, we will possibly have a more global user identification to deal with, the authentication identifier (AuthID). One way to deal with this problem is to give **LPRng** intimate knowledge of the UserID and AuthID relationship. While this is possible it is difficult to deal with in a simple and extensible manner. An alternate solution is to provide a mapping service, where the authentication procedure provides a map between the UserID and AuthID.

17.9. RFC1179 Protocol Extensions

The RFC1179 protocol specifies that a **lpd** server command sent on a connection has the form:

```
\nnn[additional fields]\n
```

`\nnn` is a one octet (byte) value with the following meaning:

```
REQ_START   1    start printer
REQ_RECV    2    transfer a printer job
REQ_DSHORT  3    print short form of queue status
```

```
REQ_DLONG   4    print long form of queue status
REQ_REMOVE  5    remove jobs
```

The **LPRng** system extends the protocol with the following additional types:

```
REQ_CONTROL 6    do control operation
REQ_BLOCK   7    transfer a block format print job
REQ_SECURE  8    do operation with authentication
```

The REQ_CONTROL allows a remote user to send LPC commands to the server. The REQ_BLOCK provides an alternate method to transfer a job. Rather than transferring the control and data files individually, this format transfers one file. The REQ_AUTH provides a mechanism for providing an authentication mechanism and is described in this document.

17.10. Authentication Operations

Options used:

- `auth=`*client to server authentication type*
- `auth_forward=`*server to server authentication type*
- `XX_id=`*server identification*
- `XX_forward_id=`*Server identification*

A **LPRng** client **lpr**, **lpq**, **lprm**, or **lpc** to **lpd** server authenticated transfer proceeds as follows. If an authenticated transfer is specified by the `auth=protocol` entry in the printcap or configuration information, the client sends a request for an authenticated transfer to the server.

Part of the authentication request is the authentication type. If authentication type XX is requested the server will examine the information in the printcap and configuration entries for an `XX_id` value. If this value is present then the server supports authentication of this type. Further permission checks are carried out and finally the server will accept or reject the authentication request. If the request is accepted the server returns a positive acknowledgment (single 0 byte) to the requester, otherwise it returns a nonzero value and an error message.

If the request is accepted then an authentication specific protocol exchange is carried out between client and server. The commands and/or data files are encrypted and/or signed and transferred to the server. The protocol specific software on the server will then decrypt and/or check signatures, perform the requested actions, and in turn generate a status information. The status information is encrypted and/or signed by the server and sent to the client, where the client decrypts and/or checked for correct signature.

The `auth` authentication type is used as a prefix to look up authentication specific information in the printcap or configuration information. For example, if `auth=kerberos`, then `kerberos_id`,

`kerberos_forward_id`, and other information would be extracted from the printcap or configuration information.

A **lpd** server to **lpd** server authenticated transfer proceeds as follows. If an authenticated transfer is specified by the `auth_forward=protocol` entry in the printcap or configuration information, the originating server sends a request for an authenticated transfer to the destination server. The originating server plays the part of the client and performs the same set of actions.

The following printcap or user level information needs to be provided for an authenticated exchange.

1. The `auth` option specifies the authentication type to be used for client to server transfers. For example, `auth=kerberos` or `auth=kerberos5` or would specify Kerberos 5 authentication, `auth=kerberos4` would specify Kerberos 4 authentication, `auth=pgp` would specify PGP authentication, `auth=md5` would specify MD5 authentication, etc. The special form `auth@` specifies no authentication.
2. The `auth_forward` option specifies the authentication type to be used for server to server transfers. For example, `auth_forward=kerberos5` would specify Kerberos 5 authentication, etc. The special form `auth@` specifies no authentication.
3. The `AUTHTYPE_id` option specifies the identification to be used for the destination in client to server transfers. For example, `kerberos_id` would specify the principal name of the `lpd` server to be used by the client for the client to server kerberos authentication.
4. When forwarding a job to a remote server, the `AUTHTYPE_forward_id` option specifies the identification to be used for the destination in server to server transfers. For example, `kerberos_id` would specify the principal name of the originating `lpd` server and the `kerberos_forward_id` option specifies the destination server kerberos id.
5. When forwarding a job to a remote server, if the original job does not have an authenticated user name or identification, then the `AUTHTYPE_default_client_id` option specifies the name to be used to identify the job. As discussed in later sections, this will be the `A` record value of the transferred control file.
6. The authenticated transfer request sent to a server has one of the following forms, depending on the originator:

   ```
   \008printer C user_id authtype \n  - for commands (lpq, lpc, etc.)
   \008printer C user_id authtype size\n - for print jobs (lpr)
   \008printer F server_id authtype \n - forwarded commands (lpq, lpc, etc.)
   \008printer F server_id authtype size\n - forwarded print jobs (lpr)
   ```

   The single character with the `\008` value signals that this is an authentication request the `printer` is the name of a print queue, and the `C` (client) or `F` indicates that the request is from a client program or is a forwarded request from a server. The `user_id` or `server_id` field is an identifier supplied by the originator and is discussed below. If the `size` value is present then the request is for a job transfer and this value represents the job size. It is used to determine if there is sufficient space in the spool queue for the job.
7. The `user_id` or `server_id` fields in the authentication request are obtained as follows. If the request originates from a client, then the `user_id` is the user name of the originator obtained from password information. If the request originates from a server, then the `server_id` is the printcap or configuration `xx_id=server_id` value, where `xx` is the value of the `auth_forward=xx` entry.

8. When the authenticated transfer request is received, the destination will either return a single zero byte, or a non-zero byte value followed by additional refusal information. A refusal terminates the protocol exchange.
9. Further exchanges are then determined by the authentication protocol specific requirements.
10. Once the initial exchanges have been completed a user file and/or command will be transferred to the destination server.
11. An authentication protocol specific AUTHFROM and AUTHUSER strings will be supplied to the lpd server for purposes of permission checking.
12. The lpd server then carries out the requested operation, and will write error and status information into a file.
13. After the requested activity has finished, protocol specific module transfer the status information in the file to the requesting system and terminate the protocol exchange.

17.11. Permission Checking

When an authenticated transfer has been performed, the following permission information will be provided.

- AUTH This value is `true` or `match` if an authenticated request was received.
- AUTHTYPE=authtype This has the value of the `authtype` field in the authentication request.
- AUTHUSER=userinfo This is the AUTHUSER information provided by the authentication protocol, and is usually the originating user's identification.
- AUTHFROM=frominfo This is the AUTHUSER information provided by the authentication protocol, and is usually the originating system (user or lpd server) identification.
- AUTHSAMEUSER This item has effect only when checking jobs in a spool queue. The AUTHUSER information from the request is compared to the AUTHUSER information from the request that created a job. If they are identical, the match succeeds.
- AUTHJOB This item has effect only when checking jobs in a spool queue. If the job was transfered using an authentication protocol the match succeeds.

For example, to reject non-authenticated operations, the following line could be put in the permissions file.

```
REJECT NOT AUTH
```

If a remote server has id information FFEDBEEFDEAF, then the following will accept only forwarded jobs from this server.

```
ACCEPT AUTH AUTHFROM=FFEDBEEFDEAF
REJECT AUTH
REJECT NOT AUTH
```

To allow only authenticated users to remove jobs you can use:

```
ACCEPT AUTH SERVICE=R,M,L,P AUTHSAMEUSER
REJECT AUTH
REJECT NOT AUTH
```

17.12. PGP Authentication Support

PGP is a well known encryption and authentication program. For more details see the web site http://www.pgp.net or the ftp site ftp://ftp.pgp.net.

LPRng has greatly simplified the use of PGP for authentication by building in support as follows.

- The `user` and `group` configuration entry (defaults `daemon` and `daemon` respectively) specify the user and group id used by the **lpd** server for file and program execution. PGP uses the current user id of the PGP process to determine the locations of various configuration files and information. In this discussion we will assume that **lpd** runs as uid `daemon`.
- By default, the PGP program expects the public and secret key rings to be in the `$HOME/.pgp/` directory to be readable only by the user. In order to set up PGP authentication, make sure that the `daemon` account has a home directory. The `daemon` user should not allow logins or have its login password disabled.
- Each PGP key has an associated identifier. It is recommended that the **lpd** key be `lpr@hostname`, where hostname is the fully qualified domain name of the server.
- Create the public and private keys for the server. For security reasons the `daemon` account should not have login capabilities.

```
#> su /bin/sh      # start root shell
%> HOME=/tmp
%> export HOME
%> mkdir /tmp/.pgp
%> pgp -kg
      # select 1024 or longer keys
      # set the user id to be lpr@hostname as discussed above
      # set the pass phrase
%> ls /tmp/.pgp
pubring.bak   pubring.pgp  randseed.bin  secring.pgp
%> cd /tmp/.pgp
%> pgp -kxa lpr@hostname serverkey pubring.pgp  # creates serverkey.asc
      # you will want to give serverkey.asc to users to add to their
      # public key ring
%> mkdir ~daemon/.pgp
```

```
%> cp * ~daemon/.pgp
%> chown daemon ~daemon/.pgp ~daemon/.pgp/*
%> chmod 700 ~daemon/.pgp
%> chmod 644 ~daemon/.pgp/*
```

- Next, place the passphrase for the `daemon` user in `~daemon/.pgp/serverkey`, and make sure it has owner `daemon` and `600` permissions (read/write only by `daemon`). This is extremely important. If other users can read this file then security will be severely compromised.
- Next, distribute the `servername.asc` file to users. **LPRng** server. This is usually done by placing the key file in a well known file location or making it available to users by some form of Public Key Distribution system (PKD).
- Users add the `serverkey.asc` key to their public key using:

  ```
  pgp -ka serverkey.asc
  ```

- Finally, the administrator will need to add the users public keys to the `daemon`public key ring file `pubkey.pgp`. This can most easily be done by copying all of the users public keys (in ASCII text format) to a single file (`/tmp/keyfile`)and using:

  ```
  su daemon
  pgp -ka /tmp/keyfile ~daemon/.pgp/pubring.pgp
  ```

- If the **lpd** server is using PGP to forward jobs or requests, the destination server's public key must be put in the originating servers public keyring. For example:

  ```
  su daemon
  pgp -ka /tmp/lpd.keyfile ~daemon/.pgp/pubring.pgp
  ```

17.12.1. Printcap Configuration

Options used:

- `pgp_path`=*path to PGP program*
- `pgp_id`=*destination server key used by clients*
- `pgp_forward_id`=*destination server used by server*
- `pgp_server_key`=*path to server passphrase file*

Example printcap entry:

```
pr:
    :lp=pr@wayoff
    :auth=pgp
    :pgp_id=lpr@wayoff.com
```

```
    :pgp_path=/usr/local/bin/pgp
pr:server
    :lp=pr@faroff
    :auth_forward=pgp
    :pgp_id=lpr@wayoff.com
    :pgp_path=/usr/bin/pgp
    :pgp_forward_id=lpr@faroff.com
```

The `pgp_path` value is the path to the PGP program. The program must be executable by all users.

The `pgp_id` value is the id used by PGP to look extract keys from key rings. When doing a client to server transfer this will be supplied as the id to be used for the destination, and the user's public keyring will be checked for a key corresponding to this id. When a request arrives at the server, the server will use this value as the id of a key in its private key ring. Finally, when a server is forwarding a request to a remote server, it will use this value as the id of the key in its private key ring to be used to sign or encode the destination information.

The `pgp_forward_id` value is used by the **lpd** server as the id to use to find a key for the destination.

The `pgp_server_key` is the path to the file containing the server passphrase. This file will be read by **lpd** to get the passphrase to unlock the server's keyring.

17.12.2. User Files and Environment Variables

Options used:

- `PGPPASSFILE`=*File to read PGP passphrase from*
- `PGPPASSFD`=*File descriptor to read PGP passphrase from*
- `PGPPASS`=*PGP passphrase*

One problem with using PGP is the need to have users input their passphrases. The following methods can be used.

- Put the passphrase in a file, say `$(HOME)/.pgp/.hidden`, and set the PGPPASSFILE environment variable to the file name. This file will be opened and read by PGP to get the passphrase. This file should be owned by the user and have `0600` or read/write only by user permissions.
- A more subtle solution is to use the PGPPASSFD environment variable facility. This causes PGP to read the passphrase from a file descriptor. If the user puts his passphrase in a file, say `$(HOME)/.pgp/.hidden`, then the following shell script can be used:

```
#!/bin/sh
#  /usr/local/bin/pgplpr script - passphrase in $(HOME)/.pgp/.hidden
#
PGPASSFD=3 3<$(HOME)/.pgp/.hidden lpr "$@"
```

- The least desirable method is to put the passphrase in the PGPPASS environment variable. Since the `ps` command can be used to list the environment variables of processes, this is highly undesirable and should not be used under any circumstances.

17.13. Using Kerberos 5 for Authentication

LPRng Kerberos 5 authentication is based on the Kerberos5-1.2.5 release as of 3 June 2002. The distribution was obtained from MIT from the *http://web.mit.edu/kerberos/www/* Website.

The following sections briefly describes how to set up and test the Kerberos software and then how to configure **LPRng** to use Kerberos.

17.13.1. LPRng Configuration

The following `configure` options are used to enable Kerberos support:

```
--enable-kerberos         enable Kerberos V support
--enable-mit_kerberos4    enable MIT Kerberos 4 support
--disable-kerberos_checks disable Kerberos sanity checks
```

The `--enable-kerberos` option will cause `configure` to search for the include files such as `krb5.h` and the `krb5` support libraries. libraries. If it finds these, then Kerberos authentication will be included. The `--enable-mit_kerberos` enable searching for the Kerberos 4 include files and support libraries. If these are found then MIT Kerberos 4 compatibility will be enabled. The `--disable-kerberos_checks` will disable checking for libraries and simply enable the various options.

17.13.2. Kerberos Installation Procedure

1. Get the Kerberos 5 distribution.
2. Compile and install the distribution.
3. Create the `/etc/krb5.conf` and `/usr/local/var/krb5kdc/kdc.conf`, files using templates from the files in the Kerberos distribution's `src/config-files` directory. See the Kerberos Installation Guide and the Kerberos System Administrators Guide for details.
4. Start up the KDC and KADMIN servers - you might want to put the following in your `rc.local` or equivalent system startup files:

```
if [ -f /etc/krb5.conf -a -f /usr/local/var/krb5kdc/kdc.conf  ]; then
    echo -n ' krb5kdc ';    /usr/local/sbin/krb5kdc;
    echo -n ' kadmind ';    /usr/local/sbin/kadmind;
```

```
fi
```

5. Use kadmin (or kadmin.local) to create principals for your users.
6. Use kadmin (or kadmin.local) to create principals for the **lpd** servers. The recommended method is to use `lpr/hostname@REALM` as a template for the principal name, i.e. - `lpr/astart1.private@ASTART.COM` for an example. You should use fully qualified domain names for the principals. Do not assign the principal a password.

   ```
   Example:

   #> kadmin   OR #> kadmin.local
   kadmin: addprinc -randkey lpr/wayoff.private@ASTART.COM
   quit
   ```

7. Extract the keytab for each server:

   ```
   Example:
   #> kadmin   OR #> kadmin.local
   ktadd -k /etc/lpr.wayoff.private  lpr/wayoff.private@ASTART.COM
   quit
   ```

8. The `/etc/lpr.wayoff.private` file contains the keytab information which is the equivalent of a password for a server program. You should create these files and then copy the appropriate `keytab` file to `/etc/lpd.keytab` file on each server. See the warnings about of keytab files in the Kerberos Installation and Kerberos Administration manuals. You should copy the file using an encrypted connection, set the permissions to read only by the owner (`400`), and set the owner to `daemon` or the user that **lpd** will run as.

   ```
   #> chmod 400 lpr.wayoff.com
   #> scp lpr.wayoff.com root@wayoff.com:/etc/lpd.keytab
   #> ssh -l root wayoff.com
   # wayoff > chmod 400 /etc/lpd.keytab
   # wayoff > chown daemon /etc/lpd.keytab
   # wayoff > ls -l /etc/lpd.keytab
   -rw-------  1 daemon  wheel  128 Jan 16 11:06 /etc/lpd.keytab
   ```

9. If you want to have MIT Kerberos4 printing compatibility then you will need to set up Kerberos 4 `servertabs` instead of Kerberos 5 keytabs. Assuming that you have put the Kerberos 5 keytab in `/etc/lpd.keytab`, then you extract the Kerberos 4 srvtab version of the Kerberos 5 keytab using the following commands. You must put the key in the `/etc/srvtab` file in order to be compatible with the Kerberos 4 support.

   ```
   h4: {321} # ktuil
   rkt /etc/lpd.keytab
   wst /etc/srvtab
   ```

17.13.3. LPRng Configuration

The **LPRng** software needs to be configured so that it can find the Kerberos libraries and include files. By default, the include files are installed in `/usr/local/include` and the libraries in `/usr/local/lib`. Use the following steps to configure **LPRng** so that it uses these directories during configuration and installation:

```
cd .../LPRng
rm -f config.cache
CPPFLAGS="-I/usr/local/include -I/usr/include/kerberosIV" \
  LDFLAGS="-L/usr/local/lib -L/usr/lib/kerberosIV" \
  ./configure
make clean all
su
make install
```

17.13.4. Printcap Entries

Options used:

- `auth=kerberos5`*use Kerberos5 authentication*
- `kerberos_id=`*server prinicpal name (for client use)*
- `kerberos_server_principal=`*alias for kerberos_id*
- `kerberos_forward_id=`*destination server used by server*
- `kerberos_forward_principal=`*alias for kerberos_forward_id*
- `kerberos_keytab=`*location of the lpd server keytab file*
- `kerberos_service=`*service to be used*
- `kerberos_life=`*lpd server ticket lifetime*
- `kerberos_renew=`*lpd server ticket renew*

Example printcap entry:

```
pr:client
    :lp=pr@wayoff
    :auth=kerberos5
    :kerberos_id=lpr/wayoff.private@ASTART.COM
pr:server
    :lp=pr@faroff.private
    :auth_forward=kerberos5
    :kerberos_id=lpr/wayoff.private@ASTART.COM
    :kerberos_forward_id=lpr/faroff.private@ASTART.COM
    :kerberos_keytab=/etc/lpd.keytab
```

```
OR If you want to use Kerberos 4 authentication to the server
pr:client
    :lp=pr@wayoff
    :auth=kerberos4
    :kerberos_id=lpr/wayoff.private@ASTART.COM
# support both Kerberos 4 and 5 on server
pr:server
    :lp=pr@faroff.private
    :auth_forward=kerberos5
    :kerberos_id=lpr/wayoff.private@ASTART.COM
    :kerberos_forward_id=lpr/faroff.private@ASTART.COM
    :kerberos_keytab=/etc/lpd.keytab
```

The printcap configuration for Kerberos authentication is very simple.

The `kerberos_id` is the principal name of the lpd server that clients will connect to. For backwards compatibility, `kerberos_server_principal` can also be used. This values is used to obtain a ticket for the **lpd** server, and is the only entry required for client to server authentication.

The other entries are used by the **lpd** server. `kerberos_keytab` entry is the location of the keytab file to be used by the server. This contains the passphrase used by the server to authenticate itself and get a ticket from the ticket server.

The `kerberos_id` value is also used by the server during the authentication process to make sure that the correct principal name was used by the request originator. This check has saved many hours of pain in trying to determine why authentication is failing.

The `kerberos_life` and `kerberos_renew` set the lifetime and renewability of the lpd server Kerberos tickets. These values should not be modified unless you are familiar with the Kerberos system. There are extensive notes in the **LPRng** source code concerning these values. The `kerberos_service` value supplies the name of the service to be used when generating a ticket. It is stronly recommended that the `kerberos_id` entry be used instead.

17.13.5. User Environment Variables and Files

In order to use kerberos authentication, the user will need to obtain a ticket from the Kerberos ticket server. This is done using `kinit`.

No other actions are required by the user.

17.14. Using Kerberos 4 for Authentication

LPRng has built-in support for the Project Athena extensions to the RFC1179 protocol. These provide an extremely simple authentication protocol using an initial credential exchange. After the initial exchange the usual RFC1179 protocol is used.

During configuration, if the `krb.h` (Kerberos 4) include file is found, then this is enabled by default.

17.14.1. Printcap Entries

Options used:

- `auth=kerberos4`*use Kerberos4 authentication*
- `kerberos_id=`*destination server key used by clients*
- `kerberos_server_principal=`*alias for kerberos_id*

Example printcap entry:

```
pr:
    :lp=pr@wayoff
    :auth=kerberos4
    :kerberos_id=lpr/wayoff.private@ASTART.COM
```

The configuration information for Kerberos4 and Kerberos5 is identical and differ only in the authentication type. Note that only client to server authentication is supported.

17.15. Using SSL for Authentication

LPRng has built-in support for using SSL as an authentication method. The implementation is based on OpenSSL 0.9.6c and the associated libraries as of of 3 June 2002. The distribution was obtained from the OpenSSL group from the *http://www.openssl.org* Website.

SSL authentication is based a private key/secret key technology, where the various keys are placed in files (or data structures) called *certificates* or `certs`, and the certificates are *signed* by calculating a checksum over the certificate, encypting the checksum and other information using the private key of a *signing* certificate. The top level or *root* certificate is signed by its own key; lower level signing certificates can be created which are signed by the top level or root certificate, and in turn can sign other signing certificates. User certificates can be created and signed by a signing certificate which can be used in the SSL protocol for authentication purposes. The following objects are needed to use SSL encryption.

1. A top level or root certificates and a set of signing certificates. By convention, these are stored in the `/etc/lpd/ssl.ca` directory; the root certificate is usually the `ca.crt` file.
2. Each server has a certificate and private key file which are used to identify the server and sign the SSL messages. The private key file is usually stored in an encrypted form and a password is required unlock the file. By convention, the server files are stored in the `/etc/lpd/ssl.server` directory; the `server.crt` file contains the server certificate and (encrypted) private key; the `server.pwd` file contains the password to decrypt the private key.
3. Each user has a certificate and private key file which are used to identify the user and sign the SSL messages. The private key file is usually stored in an encrypted form and a password is required unlock the file. By convention, the user files are stored in the `${HOME}/.lpr` directory; the

`client.crt` file contains the client certificate and (encrypted) private key; the `client.pwd` file contains the password to decrypt the private key.

4. A utility to create and manage the SSL certificate files.

The locations of the SSL files can be specified by various options to **configure** facility and by values in the the `lpd.conf` file.

17.15.1. Certificate Management

The **lprng_cert** utility is used to set up the various directories and files required for SSL authentication. This code was derived from similar facilities developed for the `mod_ssl` extensions to the **Apache** web server. This interactive utility is very verbose and has extensive comments and assistance.

```
h110: {111} % lprng_certs
lprng_certs -- LPRng SSL Certificate Management
Copyright (c) 2002 Patrick Powell
Based on CCA by Ralf S. Engelschall
(Copyright (c) 1998-2001 Ralf S. Engelschall, All Rights Reserved.)

usage: lprng_certs option
  init              - make directory structure
  newca             - make new root CA and default values for certs
  defaults          - set new default values for certs
  gen               - generate user, server, or signing cert
  verify [cert]     - verify cert file
  index [dir]       - make certificate index files in directory dir
  encrypt keyfile   - set or change password on private key file
```

The `lprng_certs init` option will create the necessary directories for the **LPRng** software on a system. The `lprng_certs newca` option will create the root level certificate and set up a set of defaults for the creation of other certificates. The `lprng_certs defaults` option allows viewing and editting of the various default values. The `lprng_certs gen` option is used to create and sign new certificate files. The OpenSSL software assumes that the file names of the signing certificate files have a special format; the `lprng_certs index` creates links of the required format to the certificate files. Finally, the `lprng_certs verify` and the `lprng_certs encrypt` facilities can be used to verify that the certificate files have the proper format and to change the private key password respectively.

17.15.2. Creating Root Certificate

The `lprng_certs newca` option is used to create a new root signing certificate and to establish defaults.

```
h110: {112} #> lprng_certs newca
lprng_certs -- LPRng SSL Certificate Management
Copyright (c) 2002 Patrick Powell
```

```
Based on CCA by Ralf S. Engelschall
(Copyright (c) 1998-2001 Ralf S. Engelschall, All Rights Reserved.)

INITIALIZATION - SET DEFAULTS
...
```

```
STEP 1: Generating RSA private key for CA (1024 bit)
```

```
STEP 2: Generating X.509 certificate signing request for CA
```

```
STEP 3: Generating X.509 certificate for CA signed by itself
```

```
RESULT:
/etc/lpd/ssl.ca/ca.crt:
/C=US/ST=California/L=San Diego/O=Astart/OU=Certificate Authority/\
 CN=Astart CA/Email=id@astart.com
error 18 at 0 depth lookup:self signed certificate
OK
```

```
STEP 4. Encrypting RSA private key with a pass phrase for security
The contents of the certificate key file (the generated private
key) should be echo kept secret, especially so if it is used to
sign Certificates or for User authentication.  SSL experts strongly
recommend you to encrypt the key file with a Triple-DES cipher and
a Pass Phrase.  When using LPRng, you provide the password via a
file specified by the LPR_SSL_PASSWORD environent variable, or in
the ${HOME}/.lpr/client.pwd file.  The LPD server uses the
ssl_server_password_file option to specify the location of a file
containing the password.  See the LPRng Reference Manual for details, or the
printcap(5) man page.

key file is /etc/lpd/ssl.ca/ca.key
Encrypt the private key now? [Y/n]: y
Fine, you're using an encrypted private key to sign CERTS.
```

17.15.3. Creating Client and Server Certificates

The `lprng_certs gen` option allows the creation of client and server identification certificates. By convention, these are created in a default directory and the system administrator then copies them to the appropriate client or server directory.

```
h110: {112} #> lprng_certs gen
lprng_certs -- LPRng SSL Certificate Management
Copyright (c) 2002 Patrick Powell
```

```
Based on CCA by Ralf S. Engelschall
(Copyright (c) 1998-2001 Ralf S. Engelschall, All Rights Reserved.)

CERTIFICATE GENERATION
What type of certificate? User/Server/Signing Authority/Help? [u/s/a/H]
Create in '/etc/lpd/ssl.certs' [return for yes, or specify directory]
CERT name 'user-10'? [return for yes, or specify name] papowell
CERT name 'papowell'? [return for yes, or specify name]
Creating papowell in /etc/lpd/ssl.certs
Sign with Certificate '/etc/lpd/ssl.ca/ca.crt' \
   [return for yes, ? for list, or specify cert file] ?
Possible CERTS in directory '/etc/lpd/ssl.ca' are:
/etc/lpd/ssl.ca/ca.crt
/etc/lpd/ssl.ca/signer1.crt
/etc/lpd/ssl.ca/tsign.crt
Sign with Certificate '/etc/lpd/ssl.ca/ca.crt'  \
  [return for yes, ? for list, or specify cert file] signer1
Match Found /etc/lpd/ssl.ca/signer1.crt
Sign with Certificate '/etc/lpd/ssl.ca/signer1.crt'  \
  [return for yes, ? for list, or specify cert file]
Private key in /etc/lpd/ssl.ca/signer1.crt

Generating user Certificate [papowell]

STEP 1: Generating RSA private key for user (1024 bit)

STEP 2: Generating X.509 certificate signing request for user
....

STEP 3: Generating X.509 certificate signed by /etc/lpd/ssl.ca/signer1.crt
...

RESULT:
/etc/lpd/ssl.certs/papowell.crt: OK

STEP 4. Enrypting RSA private key /etc/lpd/ssl.certs/papowell.key
  with a pass phrase for security

Encrypt the private key now? [Y/n]: Fine, you're using an encrypted
  private key to sign CERTS.

STEP 5: Combine CERT and KEY file
Generate single CERT and KEY file? [Y/n]

Use the following commands to examine the CERT and KEY files:
   openssl x509 -text -in /etc/lpd/ssl.certs/papowell.crt
   openssl rsa -text -in /etc/lpd/ssl.certs/papowell.crt
```

After the certificate file has been created, then it should be copied to the appropriate location: `/etc/lpd/ssl.server/server.crt` and the password in `/etc/lpd/ssl.server/server.pwd`,

for a server or `${HOME}/.lpr/client.crt` and the password in `${HOME}/.lpr/client.pwd` for a user.

17.15.4. Creating Signing Certificates

Having only one signing certificate, i.e. - the root certificate, may make it difficult to delegate authority for the creation of user certificates and/or server certificates. The `lprng_certs gen` facility can be used to create a certificate that can be used to sign other certificates.

17.15.5. Permissions and Certificate Revocation

The certificate revocation facility is not implemented in **LPRng**, due to various technical and management issues. Instead, the `AUTHUSER` and `AUTHCA` and

17.16. Using MD5 for Authentication

LPRng has built-in support for using MD5 digests as an authentication method. The implementation is provided as an example of how to add user level authentication into the **LPRng** system.

The method used to do authentication is very simple. Each user has a file containing a set of keys that are used to salt an md5 hash. The information being transferred has its md5 checksum calculated using this salt, and is then transferred to the destination, along with the md5 hash result. At the destination the server will get the user id, obtain the salt value from a key file, and then calculate the md5 hash value. If the two are in agreement, authentication is successful.

The keyfile used for md5 authentication contains an id followed by a text string whose binary value is used as a hash key:

```
id1=key
id2=key

Example:

lpr@h2=tadf79asd%^1asdf
lpr@h1=fdfa%$^&^%$
```

17.16.1. Printcap Entries

Options used:

- `auth=md5`*use MD5 authentication*
- `auth_forward=md5`*forward using MD5 authentication*

- `md5_id=`*id for server*
- `md5_forward_id=`*id for server*
- `md5_server_keyfile=`*server keyfile*

Example printcap entry:

```
pr:
    :lp=pr@wayoff
    :auth=md5
    :md5_id=lpr@wayoff.com
pr:server
    :auth_forward=md5
    :md5_id=lpr@wayoff.com
    :md5_server_keyfile
    :md5_forward_id=lpr@faroff.com
```

The `md5_id` value is used by the client to obtain a hash key that is used to salt the md5 calculation for client to server transfers. The `md5_forward_id` value is used by the server to obtain a hash key that is used to salt the md5 calculation for server to server transfers.

The `md5_server_keyfile` contains the keys of users; the id sent as the connection information is used to obtain the key from the file.

To set up md5 authentication, all that is needed is the following.

- For each user generate a key and place it in the server keyfile. This file should have the form:

  ```
  user1@host1=asdfasdfadf
  user2@host2=a8789087asddasdf
  ```

- Assign a key to the server, and set its printcap entry to this key.

  ```
  pr:
      :lp=pr@wayoff
      :auth=md5
      :md5_id=lpr@wayoff.com
  ```

- For each user, create a user key file with the following format:

  ```
  lpr@wayoff = user1@host1 asdfasdfadf
  ```

  The first entry corresponds to the `md5_id` value in the printcap. The second field is the AUTHUSER value supplied to the server and which will be used to look up the key in the servers key file. Finally, the last field is the salt value for the md5 calculation.

17.16.2. User Environment Variables and Files

Options used:

- `MD5KEYFILE=5`*location of user keyfile*

The `MD5KEYFILE` environment variable contains the path to the user keytab file.

17.17. Adding Authentication Support

Additional types of authentication support can be added very easily to **LPRng** by using the following conventions and guidelines.

First, the authentication method can be connection based or transfer based. Connection based authentication involves the **LPRng** client or server opening a connection to the remote server, having the authentication protocol provide authentication information, and then having no further interaction with the system. This is the easiest to implement and understand method. Code needs to be provided to do a simple authentication exchange between the two ends of the connection, after which no other action needs to be taken.

Transfer based authentication is more complex, but allows encrypted transfers of information between the two systems. A connection is established between client and server (or server and server), and an initial protocol exchange is performed. Then the authentication module transfers the command or job information to the destination, where is it unpacked and/or decrypted. The internal **lpd** server facilities are then invoked by the authentication module, which also provides a destination for any error message or information destined for the client. The authentication module will encrypt or encode this information and then send it to the client program. This type of authentication is more complex, but provides a higher degree of security and reliability than the simple connection based system.

17.17.1. Printcap Support

By convention, printcap entries `auth=XXX` and `auth_forward=XXX` specifies that authentication protocol XXX is to be used for client to server and for server to server transfers respectively.

Similarly, the server receiving an authentication request must have a `XXX_id=name` entry in the printcap or configuration information. This allows several different authentication protocols to be accepted by a server.

By convention, printcap and configuration entries of the form `XXX_key` contain configuration information for the XXX authentication protocol. As part of the authentication support process the `XXX_key` values are extracted from the printcap and configuration files and placed in a simple database for the authentication support module.

If you are using a routing filter, then you can also place `XXX_key` information in the routing entry for each file, and this will be used for sending the job to the specified destination.

17.17.2. Code Support

The LPRng/src/common/sendauth.c file has the following entries at the end.

```
#define SENDING
#include "user_auth.stub"

struct security SendSecuritySupported[] = {
  /* name,       config_tag, connect,    send,   receive */
  { "kerberos4", "kerberos", Send_krb4_auth, 0, 0 },
  { "kerberos*", "kerberos", 0,           Krb5_send },
  { "pgp",       "pgp",      0,           Pgp_send },
#if defined(USER_SEND)
 USER_SEND
#endif
  {0}
};
```

This is an example of how to add user level authentication support. The user_auth.stub file contains the source code for the various modules authentication modules. You can replace this file with your own version if desired. The following fields are used.

name

The authentication name. The auth=XXX printcap or configuration value will cause the name fields to be searched using a glob match.

config_tag

When a match is found, the config_tag value is used to search the printcap and configuration entries for information. If the config_tag field has value XXX, then entries with keys XXX_key will be extracted for use by the authentication code.

connect

Routine to call to support connection level authentication. This routine is responsible for connection establishment and protocol handshake. If the value is 0, then the send field value will be used.

send

Routine to call to support transfer level authentication. The send routine is provided a file and a connection to the remote server, and is responsible for the transferring files.

The LPRng/src/common/lpd_secure.c file has the following information at the end:

```
#define RECEIVE 1
#include "user_auth.stub"

 struct security ReceiveSecuritySupported[] = {
    /* name, config_tag, connect, send, receive */
#if defined(HAVE_KRB_H) && defined(MIT_KERBEROS4)
```

```
    { "kerberos4", "kerberos",  0, 0, 0 },
#endif
#if defined(HAVE_KRB5_H)
    { "kerberos*", "kerberos",   0, 0, Krb5_receive },
#endif
    { "pgp",       "pgp",   0, 0, Pgp_receive, },
#if defined(USER_RECEIVE)
/* this should have the form of the entries above */
 USER_RECEIVE
#endif
    {0}
};
```

This information matches the same information in the `sendauth.c` file. When the authentication request arrives at the server, the `name` field values are searched for a match, and then the `config_tag` value is used to get extract configuration information from the database for the protocol.

The `receive` routine is then called and is expected to handle the remaining steps of the authentication protocol. If the routine exits with a 0 value then the lpd server expects `connection` level authentication has been done and proceeds to simply transfer information using the standard RFC1179 protocol steps. A non-zero return value indicates an error and an error is reported to the other end of the connection.

If the `receive` module is to perform `transfer` level authentication, then the module carries out the necessary steps to transfer the command and/or job information. It then calls the necessary internal **LPRng** routine to implement the desired services. After finishing the requested work, these routines return to the calling authentication module, which then will transfer data, close the connection to the remote system, and return to the calling system. The combination of 0 return value and closed connection indicates successful transfer level authentication to the server.

The `user_auth.stub` file contains the following code that sets the `USER_SEND` variable:

```
#if defined(SENDING)
extern int md5_send();
#  define USER_SEND \
  { "md5", "md5", md5_send, 0, md5_receive },
#endif
```

If the SENDING value has been defined, this causes the prototype for `md5_send()` to be place in the file and the `USER_SEND` value to be defined. This will cause the `md5` authentication information to be placed in the correct table.

17.17.3. Connection and Transfer Authentication

Rather than go into a detailed description of the code, the `user_auth.stub` file contains extremely detailed examples as well as several working versions of authentication information. It is recommended that the user start with one of these and then modify it to suit themselves.

Chapter 18. Accounting

In Academic institutions, avoiding printing accounting has been regarded as a challenge, an ongoing game of fat cat and poor starving mouse, between the brutal and corrupt Administration and the poor, downtrodden, over-charged student. We will disregard the fact that if most students put as much effort into their studies as in finding ways to avoid accounting procedures then they would be Rhodes Scholar material, but I digress...

There are two approaches to printing accounting: use software determine the number of pages that should be printed or use a hardware pagecounter facility on the printer to accurately determine the number of pages used. While the software method works well in a relatively error and security compromise free environment and where print jobs do not jam, for accurate account a hardware level pagecounter or some other method must be used. LPRng provides facilities to use either method to determine page counts.

There are also two methods available to store the actual accounting information: the simple *save to file* method and the more complex *save to program* method. LPRng provides support for both methods.

18.1. Accounting Printcap Options

The accounting facilities are controlled and enabled by the following entries in the printcap file.

Tag	Default Value	Purpose
af	NULL	accounting file name
max_accounting_file_size	100	max accounting file size (in Kbytes)
min_accounting_file_size	10	min accounting file size (in Kbytes)
as	"jobstart $H $n $P $k $b $t"	accounting info for job start
ae	"jobend $H $n $P $k $b $t"	accounting info for job end
achk	FALSE	
la	TRUE	do accounting for 'local' printer
ar	FALSE	do accounting for 'remote' transfers

18.2. Accounting Information

The `:as=...` and `:ae=...` options specify the the start of job and end of job accounting information or a set of programs to be run to record the start and end of job accounting information. The option values are expanded using the same methods as for the filter options. For example:

```
:as=jobstart $H $n $P $k $b $t
  jobstart '-Hh4.private' '-nroot' '-Pps' '-kcfA938h4.private' \
    '-b1093' '-tNov  5 19:39:25'
```

```
:ae=jobend $H $n $P $k $b $t
  jobend '-Hh4.private' '-nroot' '-Pps' '-kcfA938h4.private' \
    '-b1093' '-tNov  5 19:39:59'
```

If the options have the form `:as=|/path ...`, then the specified program is run to record the information. By convention the accounting information is passed as command line values to the program. The programs are run in the same manner as a print filter. When the `:as` or `:ae` value specifies a program then logging using the `:af` option values in the next section is not performed. For example:

```
:as=|/usr/local/libexec/jobstart $H $n $P $k $b $t
  /usr/local/libexec/jobstart '-Hh4.private' '-nroot' \
  '-Pps' '-kcfA938h4.private' '-b1093' '-tNov  5 19:39:25'
:ae=jobend $H $n $P $k $b $t
  jobend '-Hh4.private' '-nroot' '-Pps' '-kcfA938h4.private' \
    '-b1093' '-tNov  5 19:39:59'
```

18.3. Accounting File

The Accounting File (`:af=`) option value specifies the destination of accounting information. If the format of the `:af` option is `:as=| ...` , then the value is assumed to be a program to be run to record start and end of job information. The program is run in the same manner as a print filter. The values of the `:as` and `:ae` options are written to the program's `STDIN` and the output from the program's `STDOUT` is used as described below for authorization.

If the `:af=` option has the format `host%port` then a TCP/IP connection is opened to the specified port on the indicated host. The values of the `:as` and `:ae` options are written to the remote host. The port that the connection originates from will be in the range set by the configuration or printcap *originate_port* option.

Finally, if the `:af=` has neither of these formats then it will be treated as a pathname to a file. If the file exists or the `create_files` option is true, then the file will be opened and the values of the `:as` and `:ae` options are written to the file. The accounting file should be periodically truncated.

By convention the `:af=` value is passed to filters as a command line option. LPRng will pass the option value only if it is specifies a file or network destination. This implies that accounting information can be written to the accounting file or network destinations by the print spooler, `:of` filters, or print file filters. The filters are responsible for opening the accounting file or network connection.

The following is an example of information written to the accounting file:

```
jobstart '-Hh4.private' '-nroot' '-Pps' '-kcfA938h4.private' \
'-b1093' '-tNov  5 19:39:25'
start '-p12942' '-kcfA938h4.private' '-nroot' '-hh4.private' '-Pps' \
'-c0' '-Fo' '-tSun Nov  5 19:39:25 1995'
filestart '-p12944' '-kcfA938h4.private' '-nroot' '-hh4.private' '-Pps' \
'-c0' '-Ff' '-tSun Nov  5 19:39:27 1995'
fileend '-p12944' '-kcfA938h4.private' '-nroot' '-hh4.private' '-Pps' \
'-b3' '-c0' '-Ff' '-tSun Nov  5 19:39:58 1995'
end '-p12942' '-kcfA938h4.private' '-nroot' '-hh4.private' '-Pps' \
```

```
'-b2' '-c0' '-Fo' '-tSun Nov  5 19:39:59 1995'
jobend '-Hh4.private' '-nroot' '-Pps' '-kcfA938h4.private' \
'-b1093' '-tNov  5 19:39:59'
```

The `jobstart` and `jobend` lines are written by **lpd** and are the expanded `:as` and `:ae` values. The `start` and `end` line are added by the `:of` filter. This filter usually queries the printer and gets printer dependent accounting information such as the pagecounter value. The `:of` filter is then suspended and the job is processed by the various format dependent filters. The the `filestart` and `fileend` lines are produced by the other filters.

The `max_accounting_file_size` and `min_accounting_file_size` are used by LPRng to control the accounting file size. When the accounting file size exceeds the `max_accounting_file_size` (in Kbytes), it is truncated to `min_accounting_file_size` (in Kbytes). If `max_accounting_file_size` is 0 (zero), then the file is allowed to grow without limit.

18.4. Authorization and Quotas

In addition to simply recording accounting information the accounting procedures can be used to check print quotas or update databases. This is done by using the Accounting Check `:achk` flag and the `:as`, `:ae`, and `:af` network connection or program capabilities.

If the `:achk` flag is set and the `:as=` option specifies a program to be run, or the `:af=` option specifies a program to be run or a network connection then output of the program or information read from the network connection is used to control the handling of the job. If the `:as=` option specifies a program to be run then the program is run and the exit code and output is saved. If the `:as=` option specifies a string and the `:af=` option specifies a program to be run or a remote host to be contacted then the `:as=` value is written to the program STDIN or remote host. The program STDOUT or network connection is read and saved and the program exit code is saved.

If the information was read from a program, then the exit code of the program is checked:

```
Exit Code    Action
JSUCC (0)    process data read
JFAIL        retry with JFAIL status
JHOLD        hold job
JREMOVE      remove job
JABORT       abort processing jobs
other        abort processing jobs
```

If the information was read from a network connection or the program exited with JSUCC (0) then the start of the first line of the information read is used. If this line starts with the following case insensitive words then the following actions are taken:

```
Word         Action
(blank)      process job
ACCEPT       process job
FAIL         retry with JFAIL status
```

```
HOLD          hold job
REMOVE        remove job
(other)       abort processing jobs
```

These facilities can be used to implement a wide variety of quota mechanisms. The most simple method is to create a script or program that can be run as the `:as=` program. This would connect to a database server or check a database to see if user quotas had been exceeded. If they had, then it would return a `REMOVE` or `HOLD` status as appropriate.

18.5. Accessing Printer Hardware Pagecounters

The following is from Hewlett-Packard documentation, http://www.hp.com/cposupport/printers/support_doc/bpl02119.html

All HP LaserJet 4/5/6 family printers have a page count feature built into the firmware. However, this feature works differently depending on which HP LaserJet printer is being used. The following is a description of how the page count feature works for each printer within the HP LaserJet 4/5/6 printer families.

```
HP LaserJet 4/4M printers
HP LaserJet 4 Plus/4M Plus printers
HP LaserJet 4P/4MP printers
HP LaserJet 4Si/4Si MX printers
HP LaserJet 4ML printers
HP LaserJet 5P/5MP printers
HP LaserJet 6P/6MP printers
```

All of the above printers use the same method for keeping track of the number of copies. There are really two different page count values: Primary and Secondary values. Every time a page is printed, whether it is an internal job (such as a self-test) or a standard print job, the Secondary page count increases by one. This value is stored in standard RAM. Once the Secondary page count value reaches 10, the Primary page count will increase by 10. The Primary page count value is stored in a type of memory called NVRAM (Non-Volatile RAM). This is important, since NVRAM is not cleared when the printer is powered off. Standard RAM, on the other hand, is cleared when the printer is turned off or reset. Thus, the Primary page count only increases in increments of 10.

Example

You have a brand new HP LaserJet 6P printer and you print a self-test page. When you look on the test page for the Page Count value, you will see that it says 1. Next, you decide to print a two page letter and, after that, another self-test. The page count value now says 4. Internally, the printers Secondary page count (stored in RAM) has the value of 4 while the Primary page count (stored in NVRAM) still has the value of 0. Now, you turn the printer off, then back on, and print another self-test. The page count value again says 1 since the previous value of 4, stored in RAM, was cleared when the printer was powered off. Finally, print a ten page document and then turn the printer off. Upon turning the printer back on and printing out another self test, you see that the page count value is 11. Internally, the Secondary page

count value is back at 1 while the Primary page count value (stored in NVRAM) is 10. Added together, you end up with the resulting value seen on the self-test page.

The HP LaserJet 4L/5L/6L printers differ from that of the other printers in that they do not have any NVRAM available for storing page count values. Thus, no way exists for the printer to retain a page count value once the printer is powered off. The HP LaserJet 4L/5L/6L printers have only a single page count value that increases in increments of one until the printer is powered off.

18.6. Reliable Accounting

In order to do reliable accounting, the printer must be queueried for the current value of the pagecounter at the start and end of jobs and this information stored in the accounting file. The **ifhp** filter can be configured to obtain the pagecounter values and to record them at the start and end of each part of a print job. We can use the `:of` filter to read the pagecounter value at the start and end of a job, and have the other file files record information as well. The `ifhp` filter will record the pagecounter information with the `-p` option. We need simply take the difference of the starting and ending pagecounter values to find the number of pages used by a job.

```
Example:
 lpd generates:
  jobstart  - from the lpd.conf 'as=' option
  jobend    - from the lpd.conf 'ae=' option
    -H   - host name
    -n   - user name
    -P   - printer
    -k   - control file name
    -b   - byte count of job/file
    -t   - current printing time

 ifhp filter generates:
   start/end         - of filter, for entire job
   filestart/fileend - if or other filter, for each file
   (options above are same)
    -A  - identifier information
    -q  - process id of filter
    -p  - current value of page counter, 0 indicates no
          page counter on printer or it is not readable

jobstart '-Hh110.private' '-nroot' '-Plp' \
   '-kcfA129h110.private' '-b48780' '-t2001-10-19-09:36:36.000'
                           ^^^ bytes in file

start '-q26130' '-p105340' '-t2001-10-19-09:36:38.330' \
                 ^^^^^^^  starting page counter value for job
    '-Aroot@h110+129' '-nroot' '-Plp'
filestart '-q26132' '-p105340' '-t2001-10-19-09:36:38.350' \
                    ^^^^^^^  starting page counter value for file
    '-Aroot@h110+129' '-nroot' '-Plp'
fileend '-b19' '-T435' '-q26132' '-p105359' '-t2001-10-19-09:43:51.504'
                                 ^^^^^^^  ending page countvalue for file
```

```
         ^^^ number of pages printed for this file
    '-Aroot@h110+129' '-nroot' '-Plp'
end '-b19' '-T435' '-q26130' '-p105359' '-t2001-10-19-09:43:51.504'
                                ^^^^^^^  ending page countvalue for job
     ^^^ number of pages printed for this job
    '-Aroot@h110+129' '-nroot' '-Plp'
jobend '-Hh110.private' '-nroot' '-Plp' \
    '-kcfA129h110.private' '-b48780' '-t2001-10-19-09:43:51.000'
                           ^^^ bytes in file
```

If for some reason the job is killed or terminates due to error conditions, the `:of` filter may not get to record the ending value for the job. This can lead to accounting files with the following entries:

```
start '-p100' '-q20005' '-Fo' '-kcfA100taco' '-uuser' '-hhost' '-R...
filestart '-p101' '-q20005' '-Ff' '-kcfA100taco' '-uuser' '-hhost' '-R...
start '-p110' '-q20005' '-Fo' '-kcfA101taco' '-uuser' '-hhost' '-R...
filestart '-p112' '-q20010' '-Fo' '-kcfA101taco' '-uuser' '-hhost' '-R...
end '-p112' '-q20010' '-Fo' '-kcfA101taco' '-uuser' '-hhost' '-R...
```

The missing `end` is a clear indication that the user's job has been terminated. We simply use the pagecounter value determined at the start of the next job to find the numbers of pages used for this job.

18.7. LPRng accounting.pl Utility

The **LPRng** `accounting.pl` utility provides the basic framework for using the `:as=|`, `:ae=|`, and pagecounter information written to the accounting file to do reliable accounting, and may be found in the **LPRng** distribution `UTILS` directory. Usually this is modified according to local site needs and installed in the filter directory.

The utility maintains the accounting file by inserting a `START` record at the start of a job and an `END` record at the end of the job. It is assumed that the last `END` record in the file marks the last place that accounting was completed.

The following shows the printcap entry for using the `accounting.pl` utility. The `start` and `end` options are used to specify that the utility is being called at the job start or end.

```
printer
 :af=acct
 :as=|/usr/local/libexec/filters/accounting.pl start
 :ae=|/usr/local/libexec/filters/accounting.pl end
```

At the start of each job the utilty writes a `START` record into the accounting file. This record can contain information suitable for use by local site. The exit code and information written to the utility STDOUT is used by the `lpd` server to determine if the job is to be printed. This allows job quotas to be implemented

in a simple way by having the `accounting.pl` utility query a database with the user quotas and reject the job if the user's quota is exceeded.

At the end of the job, the utilitity will read the accounting file and use the recorded information to update the accounting information. In order to make this reliable, the following steps are taken.

1. The accounting file is read and scanned for the last `END` record. If there is none, then the next step starts at the beginning of the accounting file.
2. The file is scanned for `START` lines and `pagecounter` information determined at the start of a job.
3. If the last line in the accounting file does not indicate successful completetion of the job and contain pagecounting information, then the accounting procedure is abandoned until the next job completes successfully.
4. If the last line in the accounting file indicates successful completion, then its pagecounter value is used as the last page counter value.
5. Job information is updated by finding the start and end pagecounter values for each job. It is possible that a job will not have a pagecounter value recorded at its start; in this case the page usage will be 0, as it did not even get initialized.
6. After determining the accounting information, the procedure will then update and databases and the accounting file. During this update, interrupts should be disabled and the amount of time taken to update the accounting information and/or file should be minimized.

Administrators can use this script as a starting point for more advanced accounting. For example, rather than just recording the information, at the job start the script can query either a local database or a remote server to see if the user has permissions to access the printer. At the end of the job or when an END line is written to the accounting file, the local database or remote accounting server can be updated.

Chapter 19. RFC 1179 - Line Printer Daemon Protocol

RFC1179 can be obtained from the **LPRng** distribution, in the LPRng_DOC/rfc1179 directory, or from one of many sites which mirror the RFCs.

This RFC is an *informational* RFC, which means that the information in it is meant as a guide to users, and not as a fixed standard. In addition, the RFC tried to document the behavior of the BSD **lpd** print server, and left out many details dealing with error recover, error messages, extensions to the protocol, etc.

In this section, I will try to explain what RFC1179 specifies as a protocol, and many of the problems encountered in trying to use it.

19.1. Ports and Connections

Options used:

- `lpd_port`=*Port for **lpd** connections*
- `lpd_listen_port`=*Port for **lpd** to accept connection*
- `originate_port`=*Ports to originate connections on*
- `reuse_addr` FLAG *Set SO_REUSEADDR flag on connection*
- `retry_econnrefused` FLAG *Retry on connect ECONNREFUSED error*
- `retry_nolink` FLAG *Retry on device open or connection ffailure*
- `unix_socket_path` PATH *UNIX FIFO pathname for local connections*
- `socket_linger`=*socket linger timeout*

RFC1179 requires that the **lpd** server listen for TCP/IP connections on port 515. This port is registered with the Internet Naming Authority, and the `/etc/services` file or TCP/IP services database usually has an entry:

```
printer     515/tcp     spooler     # line printer spooler
```

RFC1179 explicitly states that all connections to port 515 must originate from ports 721-731. The reason for this restriction is due to the UNIX concept of *reserved* and *privileged* ports. By convention, ports in the range 1-1023 can only *bound* by processes whose Effective User ID (EUID) is 0 (root). This, ordinary users could not originate a connection from the reserved or privileged port range.

In a UNIX environment, this means that the user programs **lpr**, **lpq**, **lprm**, and **lpc** would have to be SETUID root.

As experience has shown, for security purposes, the fewer programs that need to have privileged status, the better. **LPRng** uses the `lpd_port=printer` configuration option to set the port for the connections to a **lpd** server. By default, this is port 515, but can be set to other values. This port value is used to make connections to a remote **lpd** server. The `lpd_listen_port=printer` configuration option can be used to specify a port for the **lpd** to listen for incoming requests. If no `lpd_listen_port` value is specified the `lpd_port` value will be used as the **lpd** listening port.

The `unix_socket_path` option specifies the pathname of a UNIX FIFO or socket that can be used for connections the **lpd** server if the client and server are on the same host. The use of a local FIFO restricts connections from outside hosts. The UNIX FIFO path should be to a node in a directory that is writable by by the **lpd** server and not other non-privileged processes.

The restriction of originating ports to 721-731 causes another set of problems. Part of the TCP/IP protocol is concerned with avoiding communications problems resulting from the arrival of old or *stale* packets. When a connection between `sourcehost, sourceport` and `desthost, destport` is made, a set of sequence numbers is established and used for sending and acknowledgement of data. When the connection terminates, the TCP/IP protocol restricts the establishment of a new connection between `sourcehost, sourceport` and `desthost, destport` for a period long enough for all *stale* packets to be removed from the system. This is approximately 10 minutes long.

In order to simplify assignments of ports, timing out connections, and other matters, many TCP/IP packages do keep track of explicit connections *originating* from a port, but simply prevent the port from being reused for either origination or reception of a connection. They do, however, keep track of the active connections *to* a port, and perform timeouts on these. This is usually much simpler to implement, as it can be done with a list attached to the port.

This implementation method creates some problems when a large number of connections must be originated from a relatively small number of port numbers. Observe what happens when host 1 tries to send a large number of jobs to a server 2. The following connections are established and terminated: `host 1, port 721` and `host 2, port 515` `host 1, port 722` and `host 2, port 515` `host 1, port 723` and `host 2, port 515` `host 1, port 724` and `host 2, port 515` `host 1, port 725` and `host 2, port 515` `host 1, port 726` and `host 2, port 515` `host 1, port 727` and `host 2, port 515` `host 1, port 728` and `host 2, port 515` `host 1, port 729` and `host 2, port 515` `host 1, port 730` and `host 2, port 515` `host 1, port 731` and `host 2, port 515`

Now according to the RFC1179 rules and the TCP/IP protocol, we will have to wait until one of these connections terminates before we can make another. On the originating system, if the TCP/IP implementation does timeouts on the originating port, we will have to wait for the timeout to elapse before we can make a new connection. Unfortunately, there is no way to find out what the status of the port is, so we will have to try them each in turn until we get a successful connection.

The **LPRng** code has tried to provide several methods to deal with these problems. Firstly, the `originate_port=512 1023` option specifies the range of ports used to originate connections when the software is running either as ROOT or SETUID root. By strict RFC1179 rules, this should be `originate_port=721 731`, but it turns out that most BSD **lpd** based implementations only check for a *reserved* originating port. By using 512 ports we get a greatly reduced rate of errors due to lack of ports due to pending timeouts.

However, on some systems which are acting as servers for a large number of printers even increasing this port range is insufficient, and steps need to be taken use the originating port numbers more efficiently. The Berkeley TCP/IP implementation `getsockopt()` and `setsockopt()` allows the user to

manipulate some of the underlying timeouts and options of the TCP/IP network. When a TCP/IP connection is established, the `setsockopt()` facility can be used to set the `SO_REUSEADDR` flag on the connection. This flag effectively sets the timeout value on the ports and connections to 0, allowing immediate reuse of the ports. When done on an originating end of a connection, this will allow the originating port number to be reused immediately.

It would appear that by setting `SO_REUSEADDR` on the originating end that we have solved our problems. However, unless the destination end of the connection sets its `SO_REUSEADDR` flag on the connection, it will still do a timeout. Thus when we try to make a connection from a port that was active within a short period of time to the same host, then it will reject the connection until the timeout is over.

The `reuse_addr` flag (default off) forces the **LPRng** software to set the `SO_REUSEADDR` flag on originating connections. As indicated, this will allow ports to be reused immediately for outgoing connections, rather than waiting for a timeout.

While the `reuse_addr` flag usually allows us to reuse ports, there is still the problem of dealing with connections failing due to the remote site rejecting the connection due to a pending timeout from a previous connection. A careful study of the original BSD TCP/IP network code and of some others indicates that when a connection fails due to a pending timeout, an ECONNREFUSED error code is returned to a `connect()` system call. If this happens and we suspect that the remote site is rejecting the connection due to a timeout problem, then we should retry making the connection but from a new port, and continue retrying until all possible ports are used.

The `retry_econnrefused` (default on) flag is used to specify that we retry connections in this manner. When this is set, a `connection refused` error causes the connection to be retried using a new port. This will be repeated until all available ports have been tried.

When printing a job and the **lpd** server connection to a remote site or device open fails, the `retry_nolink` (default on) will cause the attempt to be retried indefinitely. The combination of `retry_econnrefused` and `retry_nolink` will provide robust connection attempts to remote systems.

While the above problems cause difficulties when making connections, there are also problems when terminating connections. After closing a socket, the TCP/IP software will try to flush any pending data to the destination. Unfortunately, on some systems it will only do this while the process is active. This has caused problems on systems which terminate a process it has received an abnormal (signal caused) termination.

The `setsockopt()` SO_LINGER option allows the user to specify that when a socket is closed normally, that the process should block until pending data is flushed or for the `socket_linger` period. If `socket_linger` is 0, then no SO_LINGER operation is done.

In summary, if you experience problems with connection failures due to port exhaustion, first try setting the `reuse_port` flag, and you should see a reduction. Check to ensure that the `retry_econnrefused` and `retry_nolink` flags are set, and the error code in the log and status files. If the failures continue, then the problem is caused by the remote end having timeout limitations and there is little you can do except to set a very long `connect_retry` interval, say `connect_retry=120` (2 minutes).

19.2. Protocol Requests and Replies

Options used:

- `remote_support`=*Remote operations supported*

After a connection has been established, a request can be sent to the **lpd** server. The request consists of a single octet indicating the request type, followed by the printer (or print queue) name, followed by a set of options for the request, followed by a LF (line feed) character.

Table 19-1. RFC1179 Commands

NNN	RFC1179	Operation	program
1	yes	start print	**lpc**
2	yes	transfer a printer job	**lpr**
3	yes	print short form of queue status	**lpr**
4	yes	print long form of queue status	**lpr**
5	yes	remove jobs	**lprm**
6	**LPRng**	do control operation	**lpc**
7	**LPRng**	transfer a block format print job	**lpr**
8	**LPRng**	secure command transfer	**lpc**
9	**LPRng**	verbose status information	**lpr**

After the request has been sent, then a reply will be returned. In general the reply has the following form:

```
\000\n    Success
\NNN\n    Failure (NNN is error code)
text\n    Text or status information
```

As can be seen, this protocol is extremely simple, but there are a set of problems due to the loosely written language of RFC1179.

1. Firstly, while RFC1179 sets limits on the lengths of commands, it does not strictly set limits on the characters set used in the commands. This can result in problems when trying to print status information, headers on banners, and other details.
2. The original RFC1179 protocol did not provide any way to do remote control of queues or **lpd** servers. This has been added to the protocol. As a side effect, if you try to use **lpc** to control a

non-**LPRng** printer, it will not work.

3. You can specify that a network printer is non-**LPRng** by using the `remote_support=RQVMC` option and specify the operations supported by the printer. The letters R, Q, M, and C stand for **lpr**, **lpq**, **lprm**, and **lpc** operations respectively, and indicate that these are supported. If `remote_support` does not allow a particular operation, then the **LPRng** software will not send a corresponding request to the printer. For example, `remote_support=R` would restrict operations to spooling jobs only, and the **LPRng** software would not query the printer for status.

19.3. Job Transfer

Options used:

- `longnumber` FLAG *Long job number (6 digits)*
- `send_data_first` FLAG *Send data files first*
- `use_shorthost` *Use short hostname*

A job transfer operation starts with a job transfer request, followed by several file transfer operations. At the end of the file transfers, the connection should be closed.

A file transfer request has the form:

Command	Purpose
\001\n	abort
\002nnnn cfname	control file transfer
\003nnnn dfname	data file transfer

The abort operation is used to terminate job transfer and indicate that the job should not be processed for printing. The connection will be closed and the partly transferred job will be discarded.

The control file and data file transfer commands have a length (in bytes) of the file and the name of the file to be transferred. When the command is received, the server will reply with a status line:

Status	Purpose
\000	Accepted, proceed
\nnn	Rejected with error code

The reply is only a single octet. Some defective implementations of RFC1179 send a LF after the octet, which makes life very difficult. **LPRng** makes an effort to detect these non-conforming RFC1179 systems and will accept jobs from them. However, it will not send jobs to them.

If **LPRng** sends a reject code, as an extension to RFC1179 it also sends an error message. Note that the values for error codes are not defined, nor are their causes. **LPRng** uses the following values for error codes, which appear to be compatible with many, but not all, of the BSD **lpd** based systems:

Code	Error
\000	Accepted, proceed
\001	Queue not accepting jobs
\002	Queue temporarily full, retry later
\003	Bad job format, do not retry

When the sender gets the reply indicating success, it sends the `nnnn` bytes of the control or data file, followed by a `\000` octet. The receiver will then reply as above; a single `\000` octet indicating success.

The above procedure is carried out until all data files and the control file of a job are transferred.

RFC1179 is silent on the following issues:

1. When sending a job, do you send the control file first, followed by the data file(s), or the data files first?
2. When sending multiple jobs, can you send them on a single connection, or do you have to establish a new connection for each job?

LPRng will *accept* jobs whether they are sent control or data files first. By default, it sends the control file first, followed by the data file. If the destination system requires that the data files be sent first, the `send_data_first` printcap option can be used to force data files to be sent first.

RFC1179 states that:

> The name of the control file ... should start with ASCII "cfA", followed by a three digit job number, followed by the host name which has constructed the control file.

The *should* in this wording indicates that this is simply a guideline, and that other formats are possible. Some of the major problems with this format are as follows:

1. The restriction to 3 digits means that at most 1000 jobs can be in a queue. Strangely, some systems generate far more than 1000 jobs a day, and need to archive them on a regular basis. The `longnumber` option will allow **LPRng** to use a 6 digit job number for files in the print queue.
2. The host name format is not specified. Some implementations consider that this is the short host name, while others think it is the fully qualified domain name (FQDN). **LPRng**, by default, will use the FQDN host name. However, the `use_shorthost` option will force it to use short host names in control and data files.
3. The `cfA` control file name was modified to allow the job priority to be used as the A letter of the control file. By default, this is A (lowest, i.e. `cfA`) and but can range to Z (highest, i.e. `cfZ`). All known spoolers except **LPRng** seem to ignore the actual value of the letter.

19.4. Data File Transfer

As mentioned before a data file is transferred using the command below.

Command	Purpose
\003nnnn dfname	data file transfer

From RFC1179:

> The data file may contain any 8 bit values at all. The total number of bytes in the stream may be sent as the first operand, otherwise the field should be cleared to 0. The name of the data file should start with ASCII "dfA". This should be followed by a three digit job number. The job number should be followed by the host name which has constructed the data file. Interpretation of the contents of the data file is determined by the contents of the corresponding control file.

There are several surprises in RFC1179.

1. Apparently a job should only consist of a single data file. This is a severe limitation, and in fact the BSD **lpr** and other print spoolers process jobs with multiple data files. By convention, these data files have names of the form `dfA`, `dfB`, ... `dfZ`, `dfa`, `dfz`.
2. The RFC does not specify that the control file and data file job numbers must be identical. Most implementations follow this convention, which simplifies life tremendously.
3. The RFC does not specify that the control file and data file job host names must be identical. Most implementations follow this convention, which simplifies life tremendously.
4. A zero length data file does not cause a data transfer to take place. **LPRng** modifies this action to be slightly different. When a zero length data file transfer is indicated, all of the input until the connection is closed is used as the contents of the data file.

   When *piping* into the **lpr** program, this can be very useful as it eliminates the need to create temporary files on the local host. Note that some print spoolers do not use this interpretation, and this option should be used carefully.

19.5. Control File Contents

The control file consists of a set of lines which either provide printing information or specify data files to be printed. The information lines start with upper case letters or digits, while the data files lines start with lower case letters. Here is a sample control file:

```
Hh4.private
J(stdin)
CA
Lpapowell
Apapowell@h4+955
Ppapowell
```

```
fdfA955h4.private
N(stdin)
UdfA955h4.private
```

The following are the letters and their meanings in the control file.

Table 19-2. Control File Lines and Purpose

Letter	Defined	Purpose
A	**LPRng**	Identifier for job
C	RFC1179	Class for banner page
H	RFC1179	Host name
I	RFC1179	Indent Printing
J	RFC1179	Job name for banner page
L	RFC1179	Print banner page
M	RFC1179	Mail When Printed
N	RFC1179	Name of source file
P	RFC1179	User identification
Q	**LPRng**	Queue name
R	**LPRng**	Accounting info
S	RFC1179	Symbolic link data
T	RFC1179	Title for pr
U	RFC1179	Unlink data file
W	RFC1179	Width of output
Z	**LPRng**	Filter options
1	RFC1179	troff R font
2	RFC1179	troff I font
3	RFC1179	troff B font
4	RFC1179	troff S font
c	RFC1179	Plot CIF file
d	RFC1179	Print DVI file
f	RFC1179	Print formatted file
g	RFC1179	Plot file
k	RFC1179	Reserved for use by Kerberized **LPRng** clients and servers.
l	RFC1179	Print file leaving control characters
n	RFC1179	Print ditroff output file
o	RFC1179	Print Postscript output file
p	RFC1179	Print file with ’pr’ format

Letter	Defined	Purpose
t	RFC1179	Print troff output file
v	RFC1179	Print raster file
z	RFC1179	Reserved for future use with the Palladium print system.

The `A` (Identifier) line was introduced to record a unique system wide job identifier for **LPRng** submitted jobs. This is basically formed from the user name, job number, and host at the time of submission. For example: `papowell@h4+955` is job number 995 submitted by papowell from host h4.

The `C` (Class) line is set by the `lpr -C class` option, and the value can be used to control printing. For example, the `lpc class zone` command would restrict job printing to only jobs with class `zone`.

The `H` (hostname), `P` (username), and `J` (jobname) fields are used to identify the host and user which sent the job, and to provide information to be displayed by **lpq** when reporting job status.

The `L` (print banner page) field is one that has caused many problems for users. RFC1179 indicates that its presence causes the banner page to be printed, and its absence suppresses banner pages. The `lpr -h` option suppresses putting this line into the control file. Usually the `L` field is a duplicate of the `P` field.

The `M` (mail information) field supplies a mail address for **LPRng** to send mail to when a job is completed. See Job Completion Notification Requested for more details.

The `N` (file name) field is usually provided to identify the file name corresponding to the data file. This can be used to print names on page separators, etc. **LPRng** largely ignores this line.

The `I` (indent) and `W` (width) fields are supposed to specify a page indent and width for printing. These fields are passed to filters if they are present.

The `Q` (queue name) field is an **LPRng** extension, and contains the name of the print queue the job was originally sent to. See qq printcap option for details.

The `R` (accounting info) field was added by **LPRng** to allow a specified account to be billed for job printing. The `lpr -Rname` option can be used to specify the accounting name.

The `S` (symbolic link) and `U` (unlink after printing) lines were used by the original BSD **lpd** print system to control how it passed files to the print server. **LPRng** ignores these lines. In fact, it will remove `S` lines and force the `U` lines to refer only to job data files. This closes a nasty security loophole on non-**LPRng** print spoolers.

The `T` (pr job title) is used with the `lpr -p` operation to supply a banner to the `pr` program.

The `Z` (filter options) value is specified with `lpr -Zoption` and is passed to the data file filters during the printing operation. See Filters for details on how the this is used during the printing process.

All of the lower case letters are reserved for format specifications for data files. In the control file, these are followed by the name of the data file to which they correspond. While in principle different data files in the control file can have different formats, this has not been implemented in any known spooling system. See Filters for details on how the data file formats are used during the printing process.

19.6. lpq Requests

The RFC1179 protocol specifies that **lpq** print status requests can be sent to the **lpd** server. The lpq

requests have the format:

```
\003printer [id]* \n    short
\004printer [id]* \n    long
\011printer [id]* \n    LPRng extension- verbose
```

The **lpd** print server will then return queue status and close the data connection.

RFC1179 does not state in any manner what the format of the queue status should be. Thus, implementors have been free to augment or change the status as they like. Even the BSD **lpq** status format has been changed from different versions.

See **lpq** - Status Monitoring Program for information on the formats returned.

The `id` values are used to select the jobs to be displayed. **LPRng** displays any job whose ID, hostname, or user name information from the control file `A`, `H`, or `P` fields match any of the id values.

Note that since there is no identification of the information requestor, then restriction of information is almost impossible.

19.7. lprm Requests

The RFC1179 protocol specifies that **lprm** job removal requests can be sent to the **lpd** server. The lpq requests have the format:

```
\005printer user [id]* \n
```

The **lpd** print server will search the specified print queue and remove any job whose ID, hostname, or user name information from the control file `A`, `H`, or `P` fields match any of the id values and for which the user has permission to perform a removal operation. See Permissions and Authentication for details.

Most RFC1179 compatible spoolers use the user information in the request as the name of the user which spooled the job. However, in a network environment this is extremely easy to fabricate, and is at best a weak type of authentication.

19.8. LPC Requests

LPRng has extended the RFC1179 protocol to allow queue and printer control commands to be sent to the **lpd** server. The format of these commands are:

```
\006printer user key [options]
```

The following commands are supported.

Table 19-3. LPC Commands

Command	Operation
Command	Operation
`active [printer[@host]]`	check to see if server accepting connections
`abort (printer[@host] | all)`	terminate server process printing job
`disable (printer[@host] | all)`	disable queueing
`debug (printer[@host] | all) debugparms`	set debug level for printer
`enable (printer[@host] | all)`	enable queueing
`hold (printer[@host] | all) (name[@host] | job | all)*`	hold job
`holdall (printer[@host] | all)`	hold all jobs on
`kill (printer[@host] | all)`	stop and restart server
`lpd [printer[@host]]`	get **lpd** PID for server
`lpq (printer[@host] | all) (name[@host] | job | all)*`	invoke **lpq**
`lprm (printer[@host] | all) (name[@host]|host|job| all)*`	invoke **lprm**
`move printer (user|jobid)* target`	move jobs to new queue
`noholdall (printer[@host] | all)`	hold all jobs off
`printcap (printer[@host] | all)`	report printcap values
`quit`	exit LPC
`redirect (printer[@host] | all) (printer@host | off )*`	redirect jobs
`release (printer[@host] | all) (name[@host] | job | all)*`	release job
`reread [printer[@host]]`	**lpd** reread database information
`start (printer[@host] | all)`	start printing
`status (printer[@host] | all)`	status of printers
`stop (printer[@host] | all)`	stop printing
`topq (printer[@host] | all) (name[@host] | job | all)*`	reorder job
`defaultq`	default queue for **lpd** server
`local (printer | all)`	client printcap and configuration information
`server (printer | all)`	server printcap and configuration information

Many of these commands support extremely specialized operations for print queue management, However, the following are the most commonly used and are supported by the BSD **lpd** print spooling system as well:

- `start, stop, enable, disable` Start and stop will start and stop printing for a specified

queue. Enable and disable enable and disable sending and/or accepting jobs for the queue.

- `abort, kill` Abort will cause the process doing the actual job printing to be terminated. Kill does an abort, and then restarts the printing process. These commands are used to restart a queue printing after some disaster.
- `topq` Places selected jobs at the top of the print queue.
- `status` Shows a status display of the print spools on the server.

The following commands are extensions to the basic set provided by the BSD **lpd** system.

- `lpq, lprm` Invokes the lpq or lprm program from lpc. Useful when in the interactive mode.
- `hold, holdall, release` The hold command will cause the selected jobs to be held until released. The holdall jobs sets all jobs submitted to the queue to be held until released. The release command releases jobs for printing. If a job has had an error and is in the error state, the release command will cause it to be reprinted.
- `move, redirect` The move command will move selected jobs to the specified spool queue. The redirect command sends all jobs submitted to the queue to be sent to the specified queue.
- `active, lpd, reread` The active command will connect to the server for the printer. This is used to check to see if non-**LPRng** print servers are active. The lpd command will connect to the server and get the process id (PID) of the **lpd** server. The reread command causes a SIGHUP signal to be sent to the lpd process, causing it to reread the `lpd.conf`, `printcap`, and `lpd.perms` files. This is done when configuration information has been modified and the administrator wants to have the server use the new information.
- `debug` This is a desperation facility for developers that allows dynamic enabling of debug information generation. Not normally used in general operation.
- `local, server` These commands will print out the configuration information in the local `lpd.conf` file, as well as the printcap information for the specified printers; `client` prints what the **LPRng** clients (`lpr, lpq, ...`) would use while `server` prints what the **LPRng** server (**lpd**) would use if running on this host. This is an extremely useful diagnostic tool for administrators. Not normally used in general operation.

19.9. Block Job Transfer

Options used:

- `send_block_format` FLAG *Transfer job as a block*

In normal job transfer operations, the sender and receiver have a handshake interaction in order to transfer a print job. Each file is sent individually. The `send_block_format` option forces a Block Job

Transfer operation. This causes the sender to transfer a single file containing all the job printing information, including control file and data files.

The transfer command line has the form:

```
\007printer size\n
```

The receiver will return any acknowledgement of a single 0 octet, and then the size bytes of the job will be transferred by the sender. At the end of the transfer a single 0 octet is added, and the receiver will indicate success by returning a single 0 octet. Any other value returned by the receiver indicates an error condition.

The file transferred by the sender is simply the command lines that it would have normally sent for job transfer, followed by the control or data file values.

19.10. Authenticated Transfer

RFC1179 does not provide any authentication or encryption mechanism for the transfer of jobs or commands to the **lpd** print server. The Authenticated Transfer operation was added to allow an encrypted or authenticated transfer of print jobs or commands.

Since there are various restrictions on the incorporation of authentication facilities into programs, **LPRng** supports authentication by providing a simple interface to encryption programs.

The idea is that when authentication is required when sending a job, **LPRng** will generate a block transfer job as described for the Block Job Transfer operation, and then invoke a set of programs to encryt and transfer the file, and encrypt and transfer the returned status.

Similarly, when sending a command, the command information will be placed in a file and the encrypted file will be transferred.

This technique means that the programs and support to do encryption are external to **LPRng**, and can use any type of method that they choose to implement the secure and/or authenticated transfer.

See Authentication and Encryption for details on the authentication interface.

Chapter 20. The Most Frequently Asked Questions

In this section, the Most Frequently Asked Questions have been placed, together with their answers. You may notice that some questions have the same answer, but the symptoms appear differently.

Some of these answers will reference other material in this FAQ, or the **LPRng** man pages.

20.1. Why do I get malformed from address errors?

This is the number one question asked by most **LPRng** users who try to use **LPRng** with network printers or other systems supporting RFC1179 printing. For details about **LPRng** and RFC1179, see RFC1179 and **LPRng**.

The `malformed from address` error is usually reported when trying to send a print job from **LPRng** to other BSD **lpr** or RFC1179 **lpr** implementations, or with network connected printers that have a built in **lpd** server. This is due to the following RFC1179 rule:

> Servers originate a connection from ports in the range 721-731.

WHY? These are a subset of the 'reserved' ports in UNIX, and normal users cannot open connections from them. This provides a small amount of security from UNIX users on the host 'spoofing' a server.

IMPLICATION: in order to do use a reserved port, the program must have root privileges. This means the LPR, **lpd**, **lpq**, etc., programs must be installed SUID root. This can open up a can of worms with regard to security, but **LPRng** has been designed to take as much paranoid care as possible to avoid problems.

WHAT TO DO: When installing **LPRng** you will need to install the executables SUID root. In the `src/Makefile`, you can remove the comment from the line

```
PERMS=SUID_ROOT_PERMS
```

and then do `make install`. This will install the executables SUID, and owned by root.

20.2. It was working normally, then I get connection refused errors

This message usually appears when you have been sending a large number of jobs to a network printer or a remote system. The reason for this is a combination the above port 721-731 restriction and the TCP/IP timeouts. For details, see RFC1179 and **LPRng**, but here is a quick explanation.

A TCP/IP connection is usually specified as between `srchost:srcport, desthost:destport`, although in practice the order of source (src) and destination (dest) is not important.

When a connection is established, each end of the connection exchanges the necessary flow control and error control information. When a connection is terminated, each end of the connection will not accept

another connection from the same `host:port` that was previously active for a specified timeout period, usually 10 minutes.

Some TCP/IP implementations go further: they will not allow ANY connection to be *originated* (via the `bind()` system call or API) from a port that was active, or accepted on a port that was active for this timeout period.

Now let us see what happens when we have a client program, which must originate a connection on port 721-731, connect to the server, which waits for a connection on port 515. We first try to make a connection from host:port `1.1.1.1:721` to `1.1.1.2:515`. The first time that we make the connection (or the first connection) we succeed. We can transfer a file, etc., and then close the connection. When we try to reconnect from `1.1.1.1:721` to `1.1.1.2:515` we get an error such as "address already in use" or "connection refused".

Luckily, we can use port 722 to originate a connection, and we can connect from `1.1.1.1:722` to `1.1.1.2:515`. We continue on, until we come to port 731, and then we need to wait for our timeouts.

SOLUTION:

It appears that most RFC1179 implementations do not check for the exact port range 721-731, but only that the connection originates from a reserved port, i.e. - in the range 1-1023. You can extend the range of ports used by **LPRng** by changing the

```
originate_port=721 731
```

value in the defaults (`LPRng/src/common/defaults.c`) file or in the `lpd.conf` file. I recommend the following:

```
originate_port=512 1022
```

This is, in fact, now the default in **LPRng** software. If you get the infamous `malformed from address` error message from your spooler, then you will have to set originate_port=721 731, and live with a delayed throughput.

20.3. Job is not in print queue, but it gets printed!

In the original BSD **lpd** implementation, the **lpr** program copied users files to a special spool queue directory, and then caused the **lpd** server to peek in the directory and print the files.

This type of operation required spool directory space, special SETUID programs, and a slew of headaches in system security and management.

The LPR, **lpq**, and other user programs in the **LPRng** suite use TCP/IP connections and transfer jobs directly to a **lpd** server running on a remote host, or even the local host if appropriate. Note that this type of operation does not require a **lpd** server to run on each local machine. In fact, you can have a single host system performing all of your printing. This type of operation is very similar to a central mail server versus individual systems, each having their own mail server and queues.

However, some users require or want their jobs to be spooled on the local host system, and then transferred to the remote printer. This is usually the case when some type of processing (filtering) is needed in order to print the job correctly. There are several methods that can be used to force this.

Method 1: Explicit Printer Address

You can force a job to be sent directly to the `pr` serviced by the **lpd** server on `host` by using the form:

```
lpr -Ppr@host file
```

You can also set the PRINTER environment variable to a similar form, and get the same effect:

```
PRINTER=pr@host; export PRINTER;
lpr file
```

Method 2: User and Server Printcap Entries

If you want to have the benefits of a printcap file, i.e. - you can use aliases or abbreviations for the names of printers, then here is a couple of hints. First, the **LPRng** software scans the `printcap` file for printcap entries, combining information for the same printer into a single entry. Information found later in the printcap file will override earlier information. In addition, you can tag entries as either being used for all utilities or just for the **lpd** server. Here are a couple of examples:

```
# for all utilities
pr:lp=pr@host
# just for lpd
pr:server
  :lp=/dev/lp
# more information
pr:check_for_nonprintable@
# --- final result for LPR
pr:lp=pr@host:check_for_nonprintable@
# --- final result for lpd
pr:lp=/dev/lp:check_for_nonprintable@
```

As you can see, the `server` keyword indicates that the printcap entry is only for the server. The **lpr** utility will send the job to the host, while the **lpd** server will print it on `/dev/lp`.

Note that the `lp=...` information overrides the `:rp:` (remote printer) and `:rm:` (remote machine) fields if they are present.

Method 3: Force sending to server on `localhost`

The `force_localhost` printcap or configuration flag forces non-**lpd** applications to send all requests and print jobs to the server running on the local host.

This method is similar to the previous one, but has the benefit that it can be configured as a global (i.e. - applies to all printers) rather than printer specific. You can put this in the `lpd.conf` file for general application, or have a printcap entry of the following form:

```
# for all utilities
pr:lp=pr@host:force_localhost
```

The **lpd** server will ignore the `force_localhost` flag, and send jobs to the `pr` queue on the `host` machine. However, the LPR, **lpq**, etc., utilities will send their requests to the server running on the local host.

20.4. Job disappears and is never printed, but lpr works

This is a rather disconcerting problem, and usually occurs when sending jobs to either a network printer or a nonconforming RFC1179 print spooler. For details about **LPRng** and RFC1179, see RFC1179 and **LPRng**, but here is a quick explanation.

An **lpd** job consists of a control file, which contains information about the job, and one or more data files. RFC1179 is silent on the order that jobs are sent; however some implementations REQUIRE that the data files be sent first, followed by the control file.

SOLUTION:

Set the `send_data_first` flag in the printcap for the particular printer, or in the `lpd.conf` configuration file. This is:

```
:send_data_first:   (printcap)
send_data_first     (lpd.conf)
```

Note that some printers/servers INSIST on the control file first; You can clear the flag using `send_job_first@` if you need to.

20.5. I get messages about bad control file format

RFC1179 describes a set of fields that MAY appear in the control file. It is silent if other ones can appear as well. Unfortunately, some implementations will reject jobs unless they contain ONLY fields from a very small set. In addition, RFC1179 is silent about the ORDER the fields can appear.

LPRng quite happily will accept jobs from poor or nonconforming RFC1179 spooler programs, and fix them up to be conformant.

If you are sending jobs to one of a non-conforming spooler, you can force **LPRng** to send jobs with only the fields described in RFC1179 by setting the the `:bk:` (BacKwards compatible) flag in the printcap for your printer.

20.6. What is RFC 1179, the Line Printer Daemon Protocol?

RFC1179 defines a standard method by which print jobs can be transferred using the TCP/IP protocol between hosts. The standard was developed by simply detailing the way that a version of the BSD **lpd**

software did its job.

From the RFC Introduction:

> RFC 1179 describes a print server protocol widely used on the Internet for communicating between line printer daemons (both clients and servers). RFC1179 is for informational purposes only, and does not specify an Internet standard.

Having said this, the RFC then goes on to describe the protocol used by a particular implementation of **lpd**. The problem was that the RFC did not provide any way to put extensions to the operations into the system, and failed to specify such interesting details as the order in which print jobs and their components could be transferred.

Comment by Patrick Powell `<papowell@lprng.com>` :

> Since 1988, there have been a large number of print spooling systems developed which claim RFC1179 conformance, but which are mutually incompatible.
>
> Rather than live with the limited capabilities of the RFC1179 standard, **LPRng** has extended them by adding capabilities to perform remote control of print spoolers, encrypted and authenticated data transfers, and other operations missing from the RFC1179 specification. However, great effort was made to be backwards compatible with older and other **lpd** based systems.
>
> **LPRng** was developed in order to be able to both accept and provide interactions with these systems. It does so by allowing various options to be used to *tune* how print jobs would be exchanged. Currently, **LPRng** can be configured to send and receive print jobs between a vast number of the existing spooling systems. It is flexible enough to act as a gateway between non-compatible systems, and has provisions to transform jobs from one format to another in a dynamic manner.

For a detailed explanation about **LPRng** and RFC1179, see RFC1179 and **LPRng**.

20.7. I want to replace lp, lpstat, etc, but my programs need them

LPRng was designed as a replacement the BSD printing system. As such, it inherited its command names and options from the latter. As you might know, SystemV uses a totally different set of commands, incompatible with the BSD ones.

The good news is that the **LPRng** binaries include an emulation for the SystemV commands. (See lp Simulation for details. Briefly, you create links to the appropriate programs, and invoke them by the link names. *Actually, these links are installed by default in recent versions.*

```
ln -s lpr lp
ln -s lpq lpstat
ln -s lprm cancel
```

If you make these links, calling `lp`, `lpstat` and `cancel` will give you a (partial) SVR4 emulation. They have their own man pages, which you should read if you need the emulation.

Since it is a *partial* emulation, you shouldn't expect everything to work. In particular, I would guess that any script which relies on the output format of one of your system binaries will break. Again, see lp Simulation for more details or additional suggestions.

Chapter 21. Remote Logger Operation

Several sites have wanted a way to provide central logging of job status and/or information. In order to do this, the following functionality is implemented in **LPRng**.

21.1. Logger Network Communication

Options used:

- `logger_destination`=*logger information destination*
- `logger_pathname`=*pathname of temp file for log information*
- `logger_max_size`=*max size in K of temp file for log information*
- `logger_timeout`=*time between connection attempts*

The printcap/configuration variable `logger_destination` specifies a destination in the standard `host%port` notation used by **LPRng**. Host is the destination host, and can be a name or IP address. Port is the port on the destination host. A TCP/IP connection is made to the indicated port.

Log information is save in a temporary file specified by `logger_path`, and up to `logger_max_size` K bytes of data will be saved.

If a connection cannot be made to the `logger_destination`, then every `logger_timeout` seconds a new connection attempt will be made. If `logger_timeout` is 0, then a connection attempt will be made every time new data arrives to be logged.

21.2. Logger Messages

Log messages consist of a single line terminated with a newline (`\n`) character.

Each log message reports a system event or status change of the LPD server. When the connection is first established, a complete dump of the status of the LPD server is sent. After this, only status update messages are sent. The remote monitor can force a status dump by simply closing and reopening the connection.

21.3. Message Format

Each message is encoded as a URI escaped string. That is, non-alphanumeric characters are encoded as the 3 character sequence `%xx`, where `xx` is the hexadecimal value of the character. The message has the format `key=value`, where `key` indicates the message type. For example:

```
dump=host=h4.private%0aprinter=t1%0aprocess=1613%0a
```

```
update_time=1999-03-23-20:32:17.148%0a\
value=queue=holdall 0%25250aprinting_aborted=0x0%25250a\
printing_disabled=0x0%25250aspooling_disabled=\
0x0%25250a%250a%0a
```

The following keys are used:

1. dump A status dump of the current contents of a print queue.

Each message has a set of headers and a value. For example, the decoded dump message from the previous section would be:

```
host=h4.private
printer=t1
process=1613
update_time=1999-03-23-20:32:17.148
value=queue=holdall 0%250aprinting_aborted=0x0%250a\
printing_disabled=0x0%250aspooling_disabled=0x0%250a%0a
```

Each line consists of a key and a value. The `host` key indicates the host name, `printer` is the print queue, `process` is the process which generated the report or action, `update_time` is the time at which the report was generated, and `value` is the value of the report.

The decoded `value` of the above report is:

```
queue='holdall 0%0aprinting_aborted=0x0%0aprinting_disabled=0x0%0a\
spooling_disabled=0x0%0a
```

The `queue` key provides the current value of the queue control file.

21.4. Dump Messages

Dump messages are generated at the start of operations, and consist of a list of queue status messages.

21.5. LPD Messages

These are used to indicate LPD startup or change in operation.

```
Decode: lpd=host=h4.private%0aprocess=1672%0aupdate_time=1999-03-23-20:5
1:10.507%0avalue=Starting%0a
host=h4.private
process=1672
```

```
update_time=1999-03-23-20:51:10.507
value=Starting
lpd: 'Starting'
```

21.6. Job Status Messages - UPDATE

Update messages are used to report changes in the queue contents, such as job arrival.

```
Decode: update=host=h4.private%0aidentifier=papowell@h4+676%0anumber=
 ...
host=h4.private
identifier=papowell@h4+676
number=676
printer=t1
process=1677
update_time=1999-03-23-20:51:17.197
value=bnrname=papowell%0acf_esc_image=Apapowell@h4+676%250aCA%250aD1999-03-
  ...
```

This update message reports the arrival of a new job at the queue. The `value` field reports the control file contents:

```
class=A
date=1999-03-23-20:51:17.151
file_hostname=h4.private
fromhost=h4.private
held=0x0
hf_name=/var/tmp/LPD/t1/hfA676
hold_class=0x0
hold_time=0x0
identifier=papowell@h4+676
job_time=0x36f86f45
jobname=/tmp/hi
logname=papowell
number=676
priority=A
queuename=t1
size=3
transfername=cfA676h4.private
update_time=1999-03-23-20:51:17.187
```

The `update_time` field in the section above is the time that the job information was last updated. The `cf_esc_image` value is the URL escaped control file information.

21.7. Printer Status Messages - PRSTATUS

These messages report printing or other activity related to a job.

```
Decode: prstatus=host=h4.private%0aidentifier=papowell@h4+676%0anumber=
676%0aprinter=t1%0aprocess=1692%0aupdate_time=1999-03-23-21:02:04.855%0avalue=
finished 'papowell@h4+676'%252c status 'JSUCC'%0a

host=h4.private
identifier=papowell@h4+676
number=676
printer=t1
process=1692
update_time=1999-03-23-21:02:04.855
value=finished 'papowell@h4+676'%2c status 'JSUCC'
PRSTATUS: 'finished 'papowell@h4+676', status 'JSUCC"
```

Appendix A. Index To All The Configuration and Printcap Options

Table A-1. LPRng Options

Option	Purpose or Value
ab	always print banner, ignore lpr -h option
accounting_namefixup	update accounting name information
achk	query accounting server when connected
ae	accounting at end (see also af, la, ar, as)
af	name of accounting file (see also la, ar)
ah	automatically hold all jobs
allow_getenv	Allow use of LPD_CONF
allow_user_logging	allow users to request logging info using lpr -mhost%port
allow_user_setting	allow privileged user to impersonate other users
ar	enable remote transfer accounting (if af is set)
as	accounting at start (see also af, la, ar)
use_auth	authentication type to use
auth_forward	authentication type for forwarding
be	Banner at End Generation Program
bk	Berkeley **lpd** job file format
bk_filter_options	Berkeley **lpd** filter options
bk_of_filter_options	Berkeley **lpd** OF filter options
bkf	backwards-compatible filters: use simple parameters
bl	short banner line sent to banner printer
bp	Banner Generation Program (see bs, be)
bq_format	Format of bounce queue output
br	Serial port bit rate (see ty)
bs	Banner at Start Generation Program
check_for_nonprintable	**lpr** checks for nonprintable file
class_in_status	Show job class name in **lpq** status information
client	Mark printcap entry for client programs only
cm	comment identifying printer (**lpq**)
config_file	configuration file
connect_grace	connection control for remote printers
connect_interval	connection control for remote printers
connect_timeout	connection control for remote printers

Option	Purpose or Value
connect_try	connection control for remote printers
create_files	create spool queue files
control_filter	control file filter
db	debug options for queue
default_format	default job format
default_permission	default permission for files
default_printer	default printer
default_priority	default job priority
default_remote_host	default remote host
default_tmp_dir	default directory for temp files
destinations	printers that a route filter may return and we should query
done_jobs	save status for last N jobs
done_jobs_max_age	remove status older than N seconds
fifo	enforce FIFO job ordering
ff	string to send for a form feed
filter_ld_path	filter LD_LIBRARY_PATH value
filter_options	filter options
filter_path	filter PATH environment variable
fo	send form feed when device is opened
force_fqdn_hostname	force FQDN hostname value in control file
force_localhost	force clients to send all requests to localhost
force_queuename	force use of this queuename if none provided
fq	send form feed when device is closed
full_time	use extended time format
generate_banner	generate banner page for forwarded jobs
group	Effective Group ID (EGID) for SUID ROOT programs
half_close	Use shutdown(fd,1) when sending job to remote printer instead of close(fd)
hl	Header (banner) last, at end of job
ignore_requested_user_priority	Ignore requested user priority
incoming_control_filter	incoming job control filter
if	default (f, l) filter program
ipv6	using IPV6 conventions
kerberos_keytab	kerberos keytab file location
kerberos_life	kerberos key lifetime
kerberos_renew	kerberos key renewal time
kerberos_forward_principal	kerberos remote principle name for forwarding

Option	Purpose or Value
kerberos_server_principal	kerberos remote server principle name
kerberos_service	kerberos default service
la	enable local printer accounting (if af is set)
ld	leader string sent on printer open
lf	error log file for spool queue
lk	lock the IO device
lockfile	lpd lock file
logger_destination	destination for logging information
logger_timeout	intervals between connection attempts
logger_pathname	temp file for log information
logger_max_size	max size in Kbytes of temp file for log information
longnumber	use long job number when a job is submitted
lp	printer device name or specification
lpd_bounce	force lpd to filter job before forwarding
lpd_force_poll	force lpd to poll idle printers
lpd_poll_time	interval between lpd printer polls
lpd_port	lpd listening port
lpd_printcap_path	lpd printcap path
lpr_bounce	lpr does filtering as in bounce queue
lpr_bsd	lpr does filtering as in bounce queue
mail_from	mail user from user name
mail_operator_on_error	mail to this operator on error
max_accounting_file_size	maximum size (in K) of accounting file
max_connect_interval	maximum time between connection attempts
max_log_file_size	maximum size (in K) of spool queue log file
max_servers_active	maximum number of lpd queue servers that can be active
max_status_line	maximum length of status line
max_status_size	maximum size (in K) of status file
mc	maximum copies allowed
min_accounting_file_size	minimum size (in K) of accounting file
min_log_file_size	minimum size (in K) of spool queue log file
min_status_size	minimum size to reduce status file to
minfree	minimum amount of free space needed
ml	minimum number of printable characters for printable check
ms_time_resolution	millisecond time resolution
mx	maximum job size (1Kb blocks, 0 = unlimited)

Option	Purpose or Value
nb	use nonblocking device open
network_connect_grace	pause between transferring jobs to remote printer
nline_after_file	N line after file name
of	banner output filter
of_filter_options	OF filter options
oh	Printcap entry valid only on these hosts
originate_port	originate connections from these ports
pass_env	clients pass these environment variables to filters
perms_path	lpd.perms files
pl	page length (in lines)
pr	pr program for p format
printcap_path	printcap file
ps	printer status file name
pw	page width (in characters)
px	page width in pixels (horizontal)
py	page length in pixels (vertical)
queue_lock_file	queue lock file name
queue_control_file	queue control file name
queue_status_file	queue status file name
qq	put queue name in control file
remote_support	operations allowed to remote host
report_server_as	server name for status reports
retry_econnrefused	Retry on connect ECONNREFUSED errors
retry_nolink	Retry device open or connect failures
return_short_status	return short lpq status when request arrives from specified host
reuse_addr	set SO_REUSEADDR on outgoing ports
reverse_lpq_format	reverse lpq format when request arrives from specified host
rg	clients allow only users in this group access to printer
rm	remote machine (hostname) (with rp)
router	routing filter, returns destinations
rp	remote printer name (with rm)
rw	open printer for reading and writing
safe_chars	additional safe characters in control file lines
save_on_error	save job when an error
save_when_done	save job when done
sb	short banner (one line only)

Option	Purpose or Value
sd	spool directory pathname
send_block_format	send block of data, rather than individual files
send_data_first	send data files first in job transfer
send_failure_action	failure action to take after send_try attempts failed
send_job_rw_timeout	print job read/write timeout
send_query_rw_timeout	status query operation read/write timeout
send_try	maximum number of times to try sending job
sendmail	sendmail program
server	Mark printcap entry for lpd server program only
server_tmp_dir	server temporary file directory
sf	suppress form feeds separating data files in job
sh	suppress header (banner) pages
short_status_date	short (hh:mm) timestamp format for status
short_status_length	short lpq status length in lines
socket_linger	set the SO_LINGER socket option
spool_dir_perms	spool directory permissions
spool_file_perms	spool file permissions
ss	name of queue that server serves (with sv)
stalled_time	time after which to report active job stalled
stop_on_abort	stop processing queue on filter abort
stty	stty commands to set output line characteristics
sv	names of servers for queue (with ss)
syslog_device	name of syslog device
tc	Include indicated printcap entries in current entry
tr	trailer string to send before closing printer
translate_format	translate data format in control file
unix_socket_path	FIFO path for local process connections
use_info_cache	read and cache information
use_shorthost	Use short hostname for lpr control and data file names
user	Effective User ID (EUID) for SUID ROOT programs
wait_for_eof	Wait for EOF on connection

Appendix B. License

LPRng, IFHP, and LPRngTool LICENSE
GNU GPL and Artistic License

(Version 5, 28 Aug 2003)

You may use "LPRng" or "IFHP" under either the terms of the GNU GPL License or the Artistc License. These licenses are included below. The licenses were obtained from the http://www.opensource.org web site on 28 Aug 2003.

These Licenses apply to the computer software packages known as "LPRng", "IFHP", and associated files. The "Package" or "Program" below refers to the programs, files, and associated software which are distributed as the package.

The "LPRng" Software Package is a copyrighted work whose copyright is held by Patrick Powell.

The "IFHP" Software Package is a copyrighted work whose copyright is held by Patrick Powell.

The "LPRngTool" Software Package is a copyrighted work whose copyright is held by Patrick Powell.

BY MODIFYING OR DISTRIBUTING THE PROGRAM (OR ANY WORK BASED ON THE PROGRAM), YOU INDICATE YOUR ACCEPTANCE OF THIS LICENSE TO DO SO, AND ALL ITS TERMS AND CONDITIONS FOR COPYING, DISTRIBUTING OR MODIFYING THE PROGRAM OR WORKS BASED ON IT. NOTHING OTHER THAN THIS LICENSE GRANTS YOU PERMISSION TO MODIFY OR DISTRIBUTE THE PROGRAM OR ITS DERIVATIVE WORKS. THESE ACTIONS ARE PROHIBITED BY LAW. IF YOU DO NOT ACCEPT THESE TERMS AND CONDITIONS, DO NOT MODIFY OR DISTRIBUTE THE PROGRAM.

Addendum Fri Jun 21 17:06:33 PDT 2002

If you wish to distribute the LPRng source code or binaries of any of the programs in the LPRng packages under terms of the GNU license, then when any package or portion of the LPRng is configured to use any facility or utility of the OpenSSL distribution, the additional following clause will be applied, as recommended in the OpenSSL 0.9.6c release FAQ:

"This program is released under the GPL with the additional exemption that compiling, linking, and/or using OpenSSL is allowed."

GNU GENERAL PUBLIC LICENSE
Version 2, June 1991

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software--to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software

patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

GNU GENERAL PUBLIC LICENSE
TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

c) If the modified program normally reads commands interactively

when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source

code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES

PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

Appendix: How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

<one line to give the program's name and a brief idea of what it does.>
Copyright (C) 19yy <name of author>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

Gnomovision version 69, Copyright (C) 19yy name of author

```
    Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type ‘show w’.
    This is free software, and you are welcome to redistribute it
    under certain conditions; type ‘show c’ for details.

The hypothetical commands ‘show w’ and ‘show c’ should show the appropriate
parts of the General Public License.  Of course, the commands you use may
be called something other than ‘show w’ and ‘show c’; they could even be
mouse-clicks or menu items--whatever suits your program.

You should also get your employer (if you work as a programmer) or your
school, if any, to sign a "copyright disclaimer" for the program, if
necessary.  Here is a sample; alter the names:

  Yoyodyne, Inc., hereby disclaims all copyright interest in the program
  ‘Gnomovision’ (which makes passes at compilers) written by James Hacker.

  <signature of Ty Coon>, 1 April 1989
  Ty Coon, President of Vice

This General Public License does not permit incorporating your program into
proprietary programs.  If your program is a subroutine library, you may
consider it more useful to permit linking proprietary applications with the
library.  If this is what you want to do, use the GNU Library General
Public License instead of this License.

-----------------------------------------------------------------------------------------------

From http://www.opensource.org  - The Artistic License
Version as of 28 Aug, 2003

                        The Artistic License

Preamble

The intent of this document is to state the conditions under which
a Package may be copied, such that the Copyright Holder maintains
some semblance of artistic control over the development of the
package, while giving the users of the package the right to use and
distribute the Package in a more-or-less customary fashion, plus
the right to make reasonable modifications.

Definitions:

"Package" refers to the collection of files distributed by the
Copyright Holder, and derivatives of that collection of files created
through textual modification.

"Standard Version" refers to such a Package if it has not been
modified, or has been modified in accordance with the wishes of the
Copyright Holder.

"Copyright Holder" is whoever is named in the copyright or copyrights
for the package.
```

"You" is you, if you're thinking about copying or distributing this Package.

"Reasonable copying fee" is whatever you can justify on the basis of media cost, duplication charges, time of people involved, and so on. (You will not be required to justify it to the Copyright Holder, but only to the computing community at large as a market that must bear the fee.)

"Freely Available" means that no fee is charged for the item itself, though there may be fees involved in handling the item. It also means that recipients of the item may redistribute it under the same conditions they received it.

1. You may make and give away verbatim copies of the source form of the Standard Version of this Package without restriction, provided that you duplicate all of the original copyright notices and associated disclaimers.

2. You may apply bug fixes, portability fixes and other modifications derived from the Public Domain or from the Copyright Holder. A Package modified in such a way shall still be considered the Standard Version.

3. You may otherwise modify your copy of this Package in any way, provided that you insert a prominent notice in each changed file stating how and when you changed that file, and provided that you do at least ONE of the following:

a) place your modifications in the Public Domain or otherwise make them Freely Available, such as by posting said modifications to Usenet or an equivalent medium, or placing the modifications on a major archive site such as ftp.uu.net, or by allowing the Copyright Holder to include your modifications in the Standard Version of the Package.

b) use the modified Package only within your corporation or organization.

c) rename any non-standard executables so the names do not conflict with standard executables, which must also be provided, and provide a separate manual page for each non-standard executable that clearly documents how it differs from the Standard Version.

d) make other distribution arrangements with the Copyright Holder.

4. You may distribute the programs of this Package in object code or executable form, provided that you do at least ONE of the following:

a) distribute a Standard Version of the executables and library files, together with instructions (in the manual page or equivalent)

on where to get the Standard Version.

b) accompany the distribution with the machine-readable source of
the Package with your modifications.

c) accompany any non-standard executables with their corresponding
Standard Version executables, giving the non-standard executables
non-standard names, and clearly documenting the differences in
manual pages (or equivalent), together with instructions on where
to get the Standard Version.

d) make other distribution arrangements with the Copyright Holder.

5. You may charge a reasonable copying fee for any distribution of
this Package. You may charge any fee you choose for support of this
Package. You may not charge a fee for this Package itself. However,
you may distribute this Package in aggregate with other (possibly
commercial) programs as part of a larger (possibly commercial)
software distribution provided that you do not advertise this Package
as a product of your own.

6. The scripts and library files supplied as input to or produced
as output from the programs of this Package do not automatically
fall under the copyright of this Package, but belong to whomever
generated them, and may be sold commercially, and may be aggregated
with this Package.

7. C or perl subroutines supplied by you and linked into this Package
shall not be considered part of this Package.

8. The name of the Copyright Holder may not be used to endorse or
promote products derived from this software without specific prior
written permission.

9. THIS PACKAGE IS PROVIDED "AS IS" AND WITHOUT ANY EXPRESS OR
IMPLIED WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE IMPLIED
WARRANTIES OF MERCHANTIBILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Appendix C. Testing and Diagnostic Facilities

The **LPRng** code has the ability to run as non-setuid software, and to use the non-default TCP/IP ports for communication. This facility allows a *Test Version* to be run in parallel with the normal **LPRng** software.

To simplify testing and portability issues, a simple test version of the spool queues and jobs has been supplied with the **LPRng** distribution. These queues can be placed in a suitable location (`/tmp` is common) and the **LPRng** software tested.

The test version of the software will use the `LPD_CONF` environment variable to specify the location of the configuration file. It will read this configuration file on startup and use the values to override the normal defaults. Since a user could maliciously set up their own configuration files with values that could compromise system security, it is strongly recommended that the test version is not made SETUID root. In fact, the **LPRng** code will chatter messages when the LPD_CONF ability is enabled and it is run as root.

C.1. Compiling the Test Version

Edit `src/Makefile`, and uncomment the indicated line. Then run `make` to regenerate the distribution.

```
#### ****** TESTING AND SECURITY LOOPHOLE ******************************
# Define GETENV to allow the LPD_CONFIG environment
#  variable to be used as the name of a configuration file.  In non-testing
#  systems,  this is a security loophole.
#CF := $(CF) -DGETENV
```

C.2. Setting Up The Test Version Spool Queues

The **LPRng** TESTSUPPORT directory contains a set of shell scripts and files that need to be installed in the appropriate directory. The following steps are used.

1. First, you need to set up your HOST environment variable to the fully qualified domain name of your host and your USER environment variable to your user name. This is done in order to get values to put into the Test Version configuration files.
2. In the TESTSUPPORT directory, edit the `Makefile`, and specify the location of the `Test Version` spool queues. The default location is `/tmp`; since on most systems these files are deleted or are available to everybody, a more secure location should most likely be used. *DO NOT USE THE RAW TESTFILE DIRECTORY*. These files need to be copied and placed in another directory.
3. The `LPD_CONF` environment variable should be set to the location of the installed `lpd.conf` file.
4. In the TESTSUPPORT directory, run `make`. This will copy and install the necessary files.

Example:

```
CSH:
  setenv HOST {fully qualified domain name};
  setenv USER `whoami`
  setenv LPD_CONF /tmp/LPD/lpd.conf
  set path=( /tmp/LPD $path )
  unsetenv PRINTER
 Example:
    setenv HOST h4.private
    setenv USER papowell
    setenv LPD_CONF /tmp/LPD/lpd.conf
    set path=( /tmp/LPD $path )
    unsetenv PRINTER
Bourne Shell:
  HOST={fully qualified domain name}; export HOST;
  USER='whoami'; export USER
  LPD_CONF=/tmp/LPD/lpd.conf.$HOST; export LPD_CONF
  PATH=/tmp/LPD:$PATH; export PATH
  PRINTER=; export PRINTER
 Example:
    HOST=h4.private; export HOST
    USER=papowell; export USER
    LPD_CONF=/tmp/LPD/lpd.conf.$HOST; export LPD_CONF
    PATH=/tmp/LPD:$PATH; export PATH
    PRINTER=; export PRINTER
cd TESTSUPPORT
make
```

C.3. Running the Test Version Software

Set your current directory to the location of the compiled `Test Version` executables. Execute the various executables using `./cmd`, or set `.` *as the first entry in the PATH* . If it is not the first entry, then the standard system executables will be used.

1. Run `./checkpc`. this will print out the various values for the spool queues in the `Test Version` setup. If the `t1`, `t2`,... spool queues are not displayed, make sure that the LPD_CONF environment variable is set correctly and that you are using the `Test Version` executable.
2. Run `./checkpc -f`. This will fix up the (deliberately introduced) problems in the spool queues.
3. Next, run `./lpd -F` in one window, and then run `./lpq -a` in another window. This will check that the server is working.
4. You can now amuse yourself by sending jobs, setting up permissions checking, and other chores.

5. When everything appears to be working correctly, you can then remove the `Test Version` flag from the `src/Makefile`, recompile, and install the **LPRng** software.

Index