An LPD for the 90s

Mark Fletcher
SAS Institute Inc.

An LPD for the 90s

Mark Fletcher – SAS Institute Inc.

ABSTRACT

lpdx was designed as an enhancement to LPD to meet the demands for seamless inter-platform printing and to simplify administration efforts relating to a large diverse printing environment. With the tremendous increase in the number of printers in the last few years, it is important to have an easy to use, network based administration tool that eliminates rlogins to servers and special accounts. As network traffic increased we noticed numerous timeout and hanging problems with the rlpdaemon/lpsched print servers we were running on our HP-UX systems. lpdx's features were brought about by the increase in the number of printers, the downsizing of many printing duties from the mainframe, and the demands an ever increasing PC base presented. Some of these features include form and job class selection, network-wide job control, printer forwarding, and better control over administration access.

Introduction

I have been studying and developing Unix printing for the last few years. In that time it has amazed me that Unix printing has seen such little growth since the original development of LPR/LPD. Perhaps this is because we keep expecting the ''paperless office'' to materialize; thinking that the printed page will go the way of the typewriter. The fact is, printed output has grown steadily every year and there does not appear to be an end in sight.

Most vendors supply some sort of modified version of LPR/LPD with their systems. Our experience has been that these print systems do not adequately address the demands of large, multi-platform, printer diverse environments. Some of the problems we have discovered are:

Lack of seamlessness
: Users are required to know unreasonable amounts of detail about print queue locations, commands, options, etc.

Lack of drivers
: Most vendor printing systems support only a simple and narrow set of printer types and capabilities.

Lack of scalability
: What seems reasonable for administering five or six printers becomes an administrative nightmare as the printer population grows into the hundreds or more. Administrators are forced to log in on multiple servers and run administrative commands. In addition to keeping track of queue locations, administrators must have login accounts on every server that contains a print queue. The problem quickly becomes unmanageable.

Lack of compatibility and simplicity
: Most ''out of the box'' solutions tend to address specific hardware (e.g., An HP-UX system talking to an HP LaserJet 4SI). Very often vendors complicate their systems with ''features'' administrators do not need or want.

Lack of capabilities
: Many features available on other systems are just not there or are poorly implemented on most Unix print systems. Because most of these systems still depend on underlying programs developed in the 70s and 80s, it's no small wonder that users are greatly disappointed when they learn of the immaturity in Unix printing.

In 1992 we developed an LPR client program called nlp (see proceedings of LISA VI conference, October 19-23, 1992). With this client, many of the above mentioned problems were solved. nlp gave us a printer database that mapped logical printer names to a <hostname, queuename>. This hid the details of server locations and queues from the user. However, since the server was vendor supplied, we still were limited by the capabilities of the server.

As our network grew we found ourselves facing many new printing challenges. I organize these into three categories.

1. SAS has been steadily downsizing mainframe resources. As we migrated printers from the mainframe to Unix, we soon discovered inadequacies that could not be resolved with the current system. The most important of these was the lack of form support. Mainframe users were accustomed to submitting jobs to specific forms criteria (type of paper, etc.) which was then released by an operator to the printer after the proper form had been loaded on the printer. Since many of these tasks were mission critical (payroll, contracts, etc.) this was a very serious problem.
2. Due to the sharp increase in the size of our printer base (we currently have over 350 printers on 150 servers) we found administration time sky rocketing. Administrators still had to log in on the servers to manage print queues

which meant that they needed accounts on all the servers. In addition, they had to determine at each occurrence which server to log into.
3. As our computing environment has matured and diversified across the globe, the demand for a true client-server printing model has become essential.

Design Strategies

We started with the "time tested" lpd source and enhanced it to conform to the client-server model we needed. We had three basic goals in mind as we developed lpdx.

1. An easy to use, client-server based administration tool that eliminates rlogins to servers and the need for special accounts.
2. A robust, down-right relentless print scheduler that won't ever give up trying to print the jobs in its queues. As the size and breadth of our network increased we noticed numerous time-out and hanging problems with the rlpdaemon/lpsched print servers we were running on our HP-UX systems.
3. Form and job class selection, network-wide job control at the user level, and printer forwarding.

These are needs that emerged due to the increase in the number of printers, downsizing of many printing duties from the mainframe, and the demands presented by an ever increasing PC base.

System V Compatibility

With SAS being primarily a System V shop, many of the features of the System V LPSCHED system were added to lpdx; mostly to take advantage of pre-existing software. The most notable is the printer interface program. Unlike the filters used by LPD, lpdx calls an administrator written program (the interface program) which handles the actual printing of the job in exactly the same manner as LPSCHED does. The lpadmin(1M) man page provides a description of the interface program. lpdx passes "well defined" parameters to the interface program (e.g., job number, login name, title) and it is the responsibility of the interface program to produce a banner page and a complete data stream suitable for the particular printer type for which it is written. Since most of our printers are network attached using network print servers, the interface program serves as a TCP client to the network print server to transfer the data. This eliminates the need for lpdx to handle output devices which

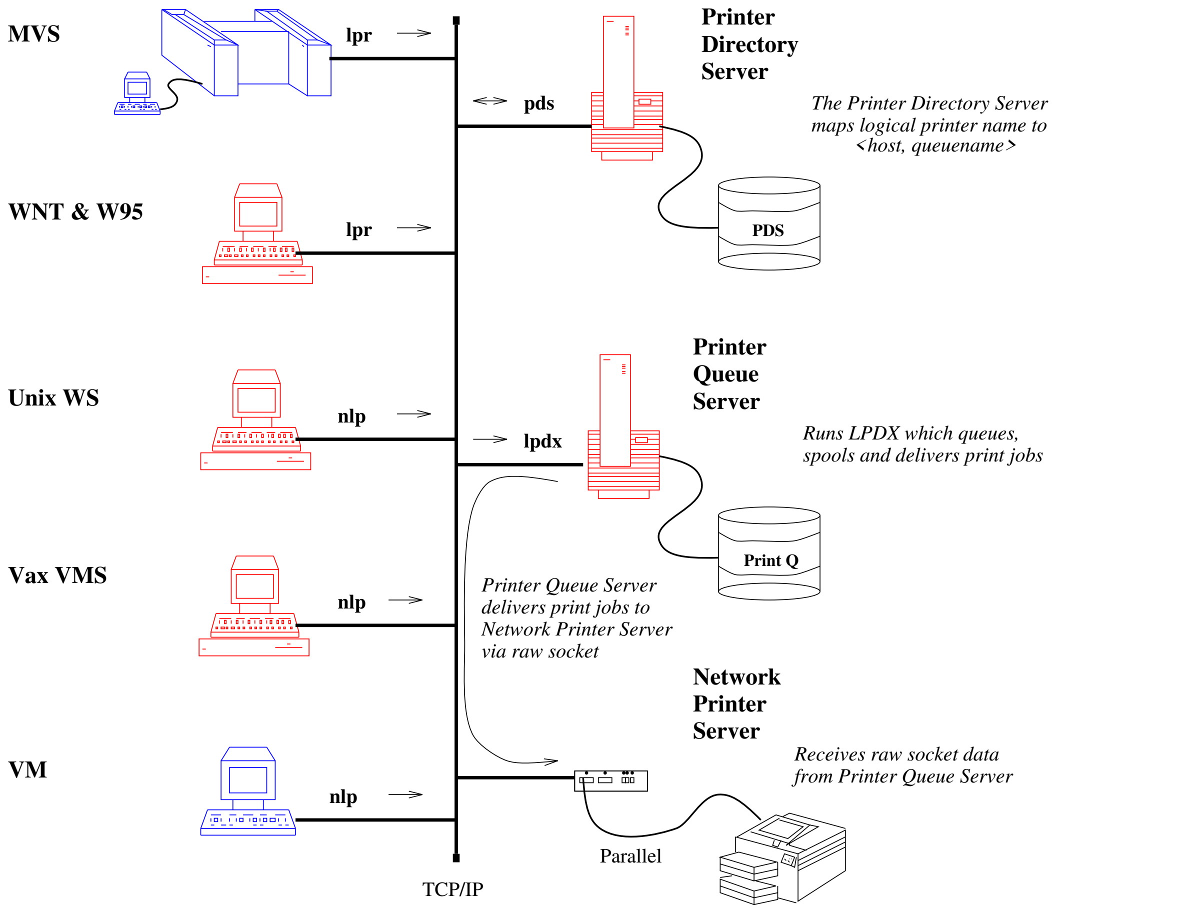

Figure 1: Typical SAS printer configuration

further simplifies the interface program as well as lpdx itself. See Figure 1.

The lpdx Directory Tree

The directory structure of lpdx is a hybrid of BSD's LPD and System V's LPSCHED. We took the features we liked from each and combined them with the new ideas we had to come up with the structure shown in Figure 2. To simplify system administration, all the files and directories relating to lpdx are maintained under a single directory tree. Additionally, we made sure the actual spool directory (request) could be replaced with a symbolic link so that the spool could be easily moved to accommodate disk space considerations. Also, because of the client-server administration tool, there is no longer any need for administrators to be ‘‘cd'ing’’ and ‘‘ls'ing’’ their way around the spool directories. This enabled us to tighten down security on the spool directories (771 or even 711) which eased the minds of those printing confidential information like payroll data and employee records.

The printcap File

The concept of a printcap file is maintained with lpdx. The file itself is moved from /etc to lpdx's *etc* directory. Typically, we only use the **sd** capability but lpdx will honor any of the valid capabilities of LPD including everything that would be needed to create a physical device directly on the server such as the parallel or serial ports.

Use of the rhosts File

The *rhosts* file can be used to restrict specific machines and/or users from accessing the print queues on a particular server. This feature allows individual groups to create ‘‘personal’’ printers that only certain users can access. The syntax of this file is the same as *~/.rhosts*.

RFC 1179 & Extensions

Care was taken to assure that lpdx remain compatible with RFC 1179 which defines the protocol for LPR/LPD. This means that any LPR client will work as well with lpdx as it does with LPD. Figure 3 shows the standard RFC 1179 protocols and the lpdx extensions. A detailed description of the extensions will follow.

A Tricky Start

Starting lpdx is a bit tricky. Figure 3 shows that you send a ‘\007’ to start lpdx. To what do you send this byte since lpdx is not running? We handled this problem with **inetd**. First, we chose a reserved port that was not being used. Amazingly, 516 was available. This was great since it is right next door to 515 (the line printer port). Therefore, we added this line to */etc/services*:

startlpd 516/tcp

The start command will then open port 516 on the server with which we wish to start lpdx on. The last detail is to add a line to the */etc/inetd.conf* file:

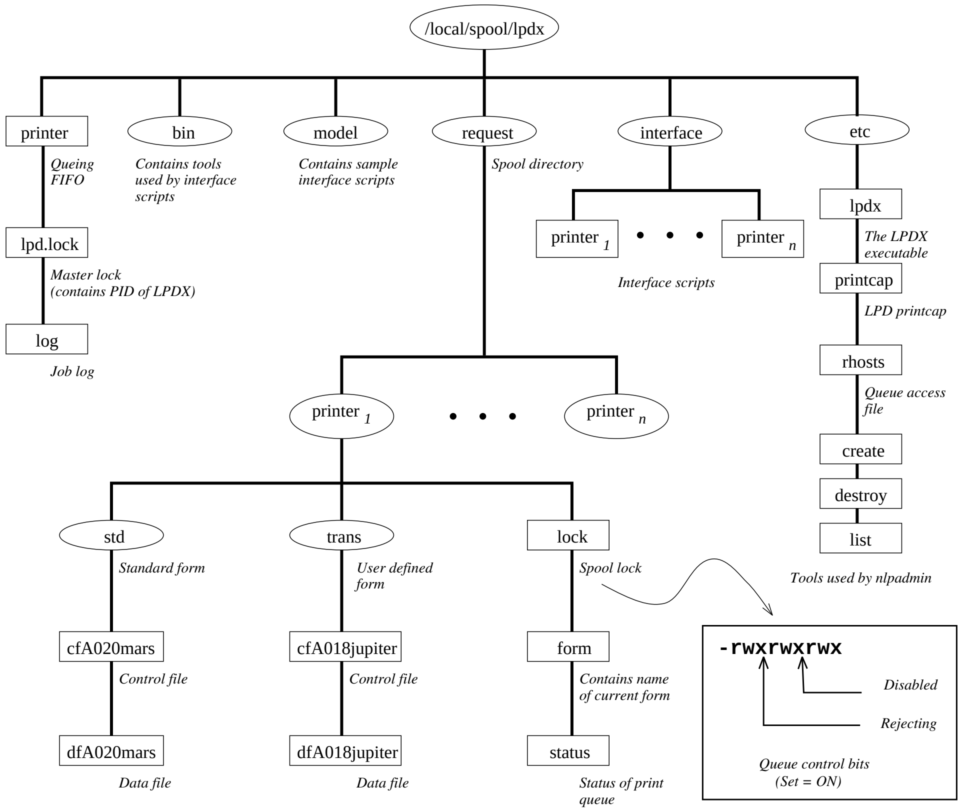

Figure 2: Directory structure of lpdx

startlpd stream tcp nowait root ...
/local/spool/lpd/etc/lpd lpd

With this, lpdx can be started from any machine on the network without having to log in on the server, or even knowing which server it is running on! Stopping lpdx is a simple manner of sending a '\010' to lpdx which causes it to **exit(0)**.

Disable and Reject

The *disable* and *reject* features of LPDSCHED was something we wanted to be sure and carry forward with lpdx. LPD, of course, already supports *disable* so we just had to enhance it slightly to support *reject* which is used to prevent users from submitting jobs (in the event that you don't want jobs piling up on a printer queue that is going to be down for some time). LPD uses the "group" **x** bit of the *lock* file to disable the queue for the printer. Since that was the only bit manipulation done to this file, we simply followed suit and used the "owner" **x** bit for our *reject* function (refer to the box in Figure 2). Disabling/Rejecting or Enabling/Accepting are now easily accomplished by toggling the appropriate bits in the printer's *lock* file.

Job Forwarding

Forwarding jobs was another very easy function to add. Since we kept the *printcap* facility completely intact, we simply utilized the **rm** capability to cause anything submitted to the printer to get redirected to a "remote" printer. Administrators simply issue the "forward" command to the printer admin tool which sends a '\022' to lpdx causing lpdx to place a **rm** line in the *printcap* file. To stop forwarding, the line is simply removed.

Forms Control

The implementation of forms constituted the most drastic modification to the LPD code. We decided to use directories to hold all jobs of a common form type and to create a form file at the base of the queue directory which contains the name of the current form. This scheme simplifies the printing side of lpdx in that it simply looks in the form file then goes to the appropriate directory and prints the jobs it finds there. However, this complicates the receiver side of lpdx a bit. When a user specifies a particular form, the form name is placed on the "class" line of the control file. As you can see from the example in Figure 4, this user has specified the form "TRANS" which we will assume means that the user intends for the job to wait until transparencies are mounted on the printer before the job is printed.

```
Hntsys.sas.com
Pmark
JWindows Print Job
CFORM=TRANS
Lmark
TSample print job
fdfA016ntsys.sas.com
UdfA016ntsys.sas.com
NC:\DOC\SAMPLE.PS
```

Figure 4: The control file from a typical LPR client

RFC-1179 protocol

\001*queue*\n	Print any waiting jobs
\003*queue list*\n	Send queue state (short)
\004*queue list*\n	Send queue state (long)
\005*queue user-name list*\n	Remove jobs
\002*queue*\n	Enter receive printer job mode
\001\n	Abort job
\002*count cfname*\n	Receive control file
\003*count dfname*\n	Receive data file

Extensions to RFC-1179 used by lpdx

\007\n	Start scheduler
\010\n	Stop scheduler
\011*queue reason*\n	Disable queue
\016*queue*\n	Enable queue
\013*src-queue dest-host dest-qname list*\n	Move jobs
\006*queue device model*\n	Create queue
\014*queue*\n	Delete queue
\015*queue form*\n	Change current form
\017*logname* \n	List log
\020*queue*\n	Accept queue
\021*queue reason*\n	Reject queue
\022*src-queue dest-host dest-queue*\n	Forward queue

Figure 3: RFC-1179 control lines and lpdx extensions

When lpdx receives jobs and spools them, it can not know which form directory to put the job in because it must open the control file to get that information (Figure 2). We decided to receive all jobs in the ‘‘std’’ directory (the default) and then move them appropriately after fetching the form name from the control file. This way, overall performance is not penalized for an exception. Since most jobs are destined for the ‘‘std’’ directory, they are already in the right place. Only if a job is destined for another form is it moved. The performance hit is not serious since all the directories are on the same file system and the effect is basically a file rename. Also, we decided to exploit the short and long forms of the ‘‘job list’’ command defined in the RFC. When lpdx receives a ‘‘short job list’’ command, only jobs in the current form are listed. A ‘‘long’’ command will display all jobs in all forms for that printer. lpdx tracks the current form using the *form* file kept at the base of the printer queue directory. The contents of this file simply contains the name of the current form. Administrators can change the current form by sending a ‘\015’ to lpdx followed by the name of the desired form and lpdx writes this name to the file.

Deleting and Moving Jobs

lpdx’s ability to move and delete jobs has been significantly improved over what either LPD or LPSCHED provide.

We decided to relax LPD’s job delete (cancel) rules a bit to allow users with the same username on more than one system to cancel their jobs from any machine. lpdx just checks to make sure the user id matches the user id of the job being canceled. It does not pay attention to the name of the client machine like LPD does. We find this works out well at our site but may be too permissive for others.

We went all out on job moving. If you look ahead to Figure 7 you see a very flexible job moving capability. We provide two tools to accomplish job moving. The user command, **nlpmv** allows a user to perform all the various types of moves to his/her jobs. The **nlpadmin** move allows any and all jobs to be freely moved anywhere throughout the network. **nlpadmin** will be described in a later section.

Job Logging

lpdx generates two types of log entries. First, a print job log is kept in lpdx’s home directory and contains entries for all the printers defined on that particular server. The detail contained in this job log is dependent on the printer’s *interface script*. In addition to the job log, lpdx also writes system type messages and error messages to */usr/adm/syslog*. Typical messages might include full disk errors or network dropouts. Between these two logs, we can get a pretty good idea about problems that occur.

Listing Jobs

LPD’s job listing output was dramatically changed. We decided to make the ‘‘long’’ listing show the contents of all forms on the printer and the ‘‘short’’ listing just the current form. Figure 5 is a listing of the ‘‘long’’ command from an LPR client.

LPR Clients

lpdx is fully compatible with standard LPR clients however the use of the ‘‘class’’ field has been changed to allow print job options to be passed to the printer’s interface script. Example options are things like job orientation, banner page suppression and forms selection. Here is an example of how a form designation can be specified with an lpr command:

lpr -Phplj2 -Cform=trans foil1.ps

As mentioned earlier, nlp is a special LPR client that accesses a printer database to translate logical printer names to <host, queuename>. We have had great success mixing LPR clients from lots of different systems with lpdx. Our largest growing community is currently Windows NT.

Robustness

The printing side of lpdx was beefed up a bit to ensure that every effort is made to find a job that shows up in the queue. To this end, a call gets made to

```
Lpd v1.1       hplj2,midnight       Current form:std    Model:lj4
-------------------------------------------------------------------------
std:
Rank   Owner   Job          Files                    Time       Total Size
1st    bud     477bigiron   B30S/VPSTCPIP/BUD.S      10:09:24   1301
2nd    jim     157panther   fiftyways.txt            10:10:03   1348

trans:
Rank   Owner   Job          Files                    Time       Total Size
1st    mark    157panther   a, b, c                  15May96    1295
2nd    martha  19myhomepc   C:/SPOOL/~LP00105.TMP    10:15:15   6768
3rd    sally   201bigiron   B20S/VPSTCPIP/SALLYBJT   10:18:32   1301
4th    mark    182panther   fiftyways.txt            10:25:22   1348
5th    mark    183panther   fiftyways.pcl ...        10:32:12   8561
```

Figure 5: Listing from an LPR ‘‘long’’ display command

the "printjob" routine following every request lpdx receives. If a "list job" request is received, for example, lpdx will "bump" the queue after the request is processed. The performance impact to do this is low. If there are no jobs in the queue "printjob" exits after just a few commands. We have not seen a single occurrence of a stalled queue with lpdx though we experienced many problems with our old spooler in this regard. Additionally, each time a job arrives, lpdx performs a series of "sanity" checks to make sure everything looks healthy. It syslogs any problems it encounters. This has taken a lot of mystery out of our printing environment.

nlpadmin – The Administrative Tool

lpdx is entirely managed by a network client called nlpadmin. At no time is it necessary for an administrator to log into the server or even be concerned with which server is in question. nlpadmin is a menu driven tool which issues commands to lpdx using extensions to the RFC 1179 protocol for LPR/LPD previously discussed from Figure 3. Figure 6 shows the nlpadmin main menu screen.

Among its capabilities, nlpadmin can start up lpdx or shut it down. Queue management includes disable/enable and reject/accept borrowed from the System V lpsched. A very powerful job move command allows jobs to be moved between forms on the same server as well as between servers. Administrators can view job logs and system logs as well as forward jobs from one print queue to another. New print queues can be easily built or deleted using the create and destroy commands. Create and destroy are actually shell scripts in lpdx's *etc* directory. These commands can be modified by administrators at any time to meet specific system requirements or special purposes.

Because nlpadmin is just another client to lpdx, many instances of nlpadmin can be on an administrator's screen at any given time. Printers anywhere on the network, whether in the same city or even the same country, can be administered simply by changing printer names within nlpadmin.

A summary of the nlpadmin commands is listed in Figure 7.

Securing nlpadmin

Security is handled via a lookup table kept in a disk file. nlpadmin runs setuid root and upon invocation it looks up the real user name in the lookup table and checks the user's authorization before allowing the user to proceed. We keep our nlpadmin authorization file in AFS so that it can be accessed by nlpadmin on most of our workstations. We use AFS ACLs to restrict access to this file.

Here is an example of the nlpadmin authorization file:

```
## Administrators of ALL printers
mark     *
jon      *
jim      *
helpdesk *

## Publications
julie    :hppubs1:hppubs2:
sam      :hppubs1:hppubs2:

## QA
testy    :hpqa1:hpqa2:
picky    :hpqa1:hpqa2:
```

This method of authentication allows us to easily track all our printer administrators. Also, "changes" made to the system by administrators is logged so that a history of all maintenance on a print queue is tracked.

Roll-out

lpdx was placed into production at SAS Institute in April 1996. Since that time, our help desk is reporting far less problems with hung jobs and hung queues. We attribute this to the robustness of the lpdx algorithms. Administrators have found nlpadmin to cut their diagnostic efforts to a fraction of what it was. They have been able to diagnose problems by reading the logs and have made repairs that they would normally pass on to second level support. Many security holes have been plugged because administrators no longer have to log in to other machines to administer their printers. With the implementation of forms support, many more mainframe tasks have been successfully moved to TCP/IP as well. Plans are underway to convert our European offices to lpdx later this year.

```
NLPADMIN Commands:
        start    stop     enable   disable  accept   reject
        form     forward  move     list     create   destroy
        nlpstat  deljob   printer  adduser  help     quit

        Enter help 'command' for syntax.

Current printer:hplj2  Current server:<server2.sas.com,hplj2>
Comm:   Net:mpx,lp20.net,p1

nlpadmin(hplj2)#
```

Figure 6: Main menu of nlpadmin

Futures

Future plans for lpdx enhancements include more integration of SNMP traps into lpdx to monitor printer status and alert users/administrators of stagnant queues, hung printers, etc. We plan to design GUI interfaces to nlpadmin as well as other user interfaces. We also have a project underway to port lpdx to Windows NT for support of dial-in users with locally attached printers. We should take another look at the nlpadmin authorization file as it was placed into AFS out of mostly laziness. The PDS should probably administer this file and be controlled via client-server.

Author Information

Mark Fletcher has been administering and developing system level tools for Unix systems for the last 10 years. He has been interested in printer system implementations for the last several years and is currently working on printer integration techniques for Windows NT and Unix. Mark can be reached via, U.S. Mail at SAS Institute Inc.; SAS Campus Drive; Cary, NC 27513. Reach him electronically at mark@unx.sas.com .

```
start:   Starts the LPD print spooler on the server for the current printer.
stop:    Stops the LPD print spooler on the server for the current printer.
enable:  Enables the printing of jobs in the printer's queue.
disable: Stops jobs on the printer from printing.
accept:  Causes the print spooler to resume accepting jobs.
reject:  Causes the print queue to stop accepting jobs.
form:    Change the active (current) form on the printer.
             Syntax: form=new_form
forward: Causes any jobs queued to printer to be forwarded to another printer.
move:    Moves job(s) to other forms or printers
             Syntax:
                 move Destination Source
                         Destination:
                                 form=formname
                                 destination-printer
                                 destination-printer:formname
                         Source:
                                 form=formname
                                 job(s)
                                 user(s)
                                 -all
             Examples:
                 move form=std 207bullet 118daisy
                 move hplj1:let sasabc
list:    Lists the printer's job log or system error log.
create:  Creates a new print queue on the server for the printer.
destroy: Deletes a print queue from the server.
nlpstat: Displays print queue
deljob:  Cancel jobs.
printer: Changes the printer nlpadmin is acting on.
adduser: Edits the nlpadmin user authorization file.
help:    Displays help on a specified command.
quit:    Quits nlpadmin
```

Figure 7: nlpadmin command summary