Printing with CUPS

Fix Me! this page is a work in progress Fix Me!

CUPS (the *Common UNIX Printing System*) is now the default printing subsystem in Slackware, after years of faithful service by the *lprng* package which is no longer part of Slackware. The commandline *lpr* program that some of you use is in fact part of CUPS, but it emulates the old lpr program that has been known in Linux/UNIX for ages and which was the way to print if you had *lprng* installed in Slackware. But a set of commandline tools is not all that CUPS installs on your system.

CUPS has a browser-based graphical user interface (check out yours at http://localhost:631/) for administering the service, like creating print queues, monitoring, deleting and restarting print jobs and such.
Not everything can be done through the graphical interface though. The basic initial setup of the CUPS service is still a matter of hand-editing certain configuration files.

CUPS can act as a server (when it has a queue configured that prints to a physical device) or a client (when there is no local printer configured but instead your job submissions are sent to a CUPS server on the network). The big plus of networked CUPS services is that they are able to talk to each other, so that printing from one CUPS machine (the client) to another CUPS machine (a CUPS server with attached printer) works automagically - the client machine will pick up the server print queues that are advertised on the network without you needing to configure anything.

Setting up the CUPS service

Slackware comes with a functional CUPS service out of the box. All you have to do is make the file `/etc/rc.d/rc.cups` executable and start the server. After that you can point your browser to http://localhost:631/ for the administratitive interface, and start adding queues for local printers or remote CUPS servers.

```
chmod +x /etc/rc.d/rc.cups
/etc/rc.d/rc.cups start
```

If you need to setup a CUPS network server (other PC's should be able to print to your machine) then there is some extra work ahead.

The way Slackware runs CUPS out of the box is that only access from the `localhost` is allowed. The print queues which you define will not be advertised on your network.
Access to `localhost` is safe - no one else but you on your computer can talk to the `localhost` address (which is tied to the *loopback interface* or "*lo*" interface).

If `localhost` does not work for you when you type the URL in the browser, try the IP address for localhost which is `127.0.0.1`. The URL would then look like this: http://127.0.0.1:631/ . But in any case, this would indicate that you messed up your `/etc/hosts` file which should contain (among others) the line

```
127.0.0.1       localhost
```

If that line is not present, you should add it. It's absence will break quite many network services.

For the sake of this article, I will present several examples containing IP addresses. Our example network will be as follows:

- Network hosts have IP addresses in the range **`192.168.0.1`** to **`192.168.0.254`**, i.e. an address range described by the network/netmask values of **`192.168.0.0/255.255.255.0`** (or the equivalent notation **`192.168.0.0/24`**).
- Our own computer is configured with the IP address **`192.168.0.5`**

- Another computer running a network-accessible CUPS server has the IP address **192.168.0.1**

/etc/cups/cupsd.conf

The `/etc/cups/cupsd.conf` file is the *server configuration file* for CUPS. It determines how the job scheduler will run. The default content of this file (stripped of - very useful - comments and empty lines) looks like this:

```
LogLevel info
Port 631
<Location />
Order Deny,Allow
Deny From All
Allow From 127.0.0.1
</Location>
<Location /admin>
AuthType Basic
AuthClass System
Order Deny,Allow
Deny From All
Allow From 127.0.0.1
</Location>
```

I will briefly go through the most important lines.

- The listen address:

  ```
  Port 631
  ```

  This tells that the CUPS server will listen at TCPIP port 631 for incoming print job submissions, and also runs the administrative Web interface at the port. Port 631 is the *IPP* port, which stands for *Internet Printing Protocol*.

- Print service access control

  ```
  <Location />
  Order Deny,Allow
  Deny From All
  Allow From 127.0.0.1
  </Location>
  ```

  This tells us that access to the root URL (*Location /* i.e. http://localhost:631/) is limited to the `127.0.0.1` IP address exclusively. Remember, `127.0.0.1` is the IP address for the `localhost` interface. Essentially this statement means that no one can access your CUPS print service from the network. If you **do** want to grant access, you will have to add another *Allow* line here. Remember the example network layout from the first section? We will allow all computers in our local network access to the CUPS printing service by adding the line **Allow From 192.168.0.*** so that the previous section will become

  ```
  <Location />
  Order Deny,Allow
  Deny From All
  Allow From 127.0.0.1
  Allow From 192.168.0.*
  </Location>
  ```

- Administration interface access control:

  ```
  <Location /admin>
  AuthType Basic
  AuthClass System
  Order Deny,Allow
  ```

```
Deny From All
Allow From 127.0.0.1
</Location>
```

The location `/admin` (i.e. the URL http://localhost:631/admin) is protected using *Basic Authentication* - the well-known username/password dialog of your browser when you attempt to access this URL for the first time. By default, the only username/password combination that works here is that of the **root** account!

Since Basic Authentication sends passwords unencrypted to the server, it makes sense that access to the admin interface is restricted to the local host - this is what the line "*Allow From 127.0.0.1*" is for. If you want access to your CUPS server's administrative interface across the network, it is desirable to protect your password from network sniffers by using *SSL encryption* (i.e.using https:// URLs instead of http://). I will show you how to secure your access in the Securing the admin web interface section.

Now, I told you that CUPS services can talk to each other - they use the *IPP* protocol for that. Print queues on one CUPS server will automatically be available for use on another computer running CUPS. An example: you take your Slackware laptop with you and plug it into the network of your exmployer. If there is a CUPS server on that network, the queues on that server will suddenly pop up on your local CUPS server! Zero config here people. The big question is of course: how does one CUPS program find other CUPS programs in the network?
The answer is, you must make the CUPS server advertise itself on the network. These are the two lines you will add to your `cupsd.conf` file to make that happen:

```
BrowseAddress @LOCAL
BrowseAllow 192.168.0.*
```

Meaning that your CUPS server will use all interfaces that it considers local. Any point-to-point interfaces (ppp, dial-up lines for instance) are *not* considered local. Also it will accept and answer browse requests from computers in our example IP range. A *browse request* is a query from another CUPS program, looking for peers on the network. Our server will now answer those browse requests, thereby making it's own print queues available for use by other CUPS clients in the network. The complete `cupsd.conf` file would look like

```
LogLevel info
Port 631
BrowseAddress @LOCAL
BrowseAllow 192.168.0.*
<Location />
Order Deny,Allow
Deny From All
Allow From 127.0.0.1
Allow From 192.168.0.*
</Location>
<Location /admin>
AuthType Basic
AuthClass System
Order Deny,Allow
Deny From All
Allow From 127.0.0.1
</Location>
```

You'll notice that I also added that line **Allow From 192.168.0.*** otherwise network connections from outside would still be refused. I leave access to the `/admin` URL restricted to `localhost` for now. Later on, we will add some security by means of SSL encryption.

/etc/cups/client.conf

Creating print queues

Securing the admin web interface

In this section I will show you how you can prevent other people on the network from accessing your CUPS server's administrative interface by password protecting it and using encryption. It would be silly if all of a sudden your print queue would be disabled by a joker…

You've probably noticed the line "*AuthClass System*" in the definition of the "`/admin`" location. This line defines the "class" of users (in fact a Linux group taken from the `/etc/group` file) which CUPS considers as being "System users". By default, CUPS will require the authenticating user to be a member of the "`sys`", "`system`", or "`root`" groups (the check is done in that order). That requirement fits only the `root` account, so that explains my statement of one of the previous sections. You can change the name of this group by uncommenting the line **`#SystemGroup sys`** in the `cupsd.conf` file, and changing the work `sys` to some other existing group. Alternatively, you can add your own acccount to the `sys` group of course.

Belonging to the right group is not enough though.
Suppose you have added "**`SystemGroup wheel`**" to `cupsd.conf` so that anyone in the `wheel` group can administer CUPS (I like to use *wheel* for administrative tasks). You will now have to create a password file for CUPS which holds your user account and an associated password. CUPS cannot read your Linux account/password (it needs PAM for that, and Slackware does not use PAM). If the password file does not yet exist, we create it as follows, assuming your user account is "alien":

```
lppasswd -g wheel -a alien
```

If the password file existed, then a line for the *alien* account would have been appended to it.
The contents of that file look like this now:

```
# cat /etc/cups/passwd.md5
    alien:wheel:01234567890abcde1234567890abcde12
```

SSL encryption

TODO

Configuring CUPS clients

TODO

Making CUPS work with Samba

Samba will pick up the CUPS printers and make them available to your Windows computers in the LAN when you have these lines in the `[global]` section of your `/etc/samba/smb.conf` file:

```
   load printers = yes
   printcap name = cups
   printing = cups
   print command = lpr -oraw -r -P'%p' %s
   lpq command = /usr/bin/lpq -P%p
```

You should also have a `[printers]` section in `smb.conf` that would look like this:

```
[printers]
   comment = All Printers
   path = /var/spool/samba
   browseable = no
   # Set public = yes to allow user 'guest account' to print
```

```
   # A synonym for 'public = yes' is: 'guest ok = yes'
   public = yes
   writable = no
   printable = yes
```

The print spool directory which Samba uses (*/var/spool/samba*) must be different from the CUPS spool directory! Check if the directory exists (usually it should) and if it is writable by all:

```
# ls -l /var/spool | grep samba
drwxrwxrwt   2 root   root    4096 2007-12-09 18:26 samba/
```

The *'t'* in `drwxrwxrwt` means that every user can write in this directory but their files cannot be deleted by anyone but themselves. If the directory does not exist, or it's permissions are incorrect, you can run the following three commands to fix that:

```
mkdir -p /var/spool/samba
chown -R root:root /var/spool/samba
chmod 1777 /var/spool/samba
```

When print jobs from the Windows computers never show up in CUPS, you should increase the debug level for your CUPS server:

```
LogLevel debug
```

and restart the CUPS server by running:

```
/etc/rc.d/rc.cups restart
```

You will see the following error messages in `/var/log/cups/error_log`:

```
   E [11/Nov/2004:20:30:25 +0100] print_job: Unsupported format 'application/octet-stream'!
   I [11/Nov/2004:20:30:25 +0100] Hint: Do you have the raw file printing rules enabled?
   D [11/Nov/2004:20:30:25 +0100] Sending error: client-error-document-format-not-supported
```

To resolve this, you need to edit two files (you will find the affected lines at the end) and afterwards restart CUPS:

- In file `/etc/cups/mime.types` uncomment `application/octet-stream`
- In file `/etc/cups/mime.convs` uncomment `application/octet-stream` `application/vnd.cups-raw 0 -`

Sometimes you will find that Samba does not pick up changes to your CUPS configuration, especially when you edited the `smb.conf` file. If you have computers connected to Samba shares and do not want to disconnect them, you can force the `smbd` processes on the Linux server to re-read their configuration while running. The following command will do that by sending a SIGHUP or *hangup signal* to these running processes:

```
killall -HUP smbd
```

Even though it uses the command `killall` this does not mean that any process gets killed!

A PDF printer for CUPS

Usually, the CUPS print queues will print to paper of course. But there are cases when you want an electronic image of some document, or web page, that you do not need on paper. The PDF file format is the ideal candidate for this, since this format is supported on virtually all operating systems and architectures.
So, we need to add a print queue to CUPS which does not print our submitted jobs to paper sheets but instead generates a PDF file from our input, and makes this file available to us in some way.
This section describes exactly how to set this up, leaving the choice up to you whether you want your PDF files delivered to you via email, or have them dumped into a directory of your choice.
This setup uses no proprietary software to create the PDF files - all you need is already present on your Slackware machine. For those with several computers in a network, having a CUPS server to generate the PDF's has the additional advantage that it does not matter what the client computer is. Your Linux and Windows workstation, or your Mac, will be equally able to use this pdfprinter as long as your CUPS service allows job submissions from the network.

If you are running a network server with Samba, your CUPS pdfprinter will be integrated into the Samba configuration if you tell it to use CUPS for printing support (which is the default behaviour anyway). Windows clients will be able to generate PDF's by using a Samba print queue.

I came across an implementation of a PDF printer that supports CUPS and Samba. It can be found at http://www.linux-als-server.de/html/special-pdfprinter.php . My recipe below is based on the instructions found at that URL. Since the original text is in german, my article will help you non-german speaking persons setting this up I hope

Required software

You will need to meet the following software requirements on your CUPS machine:

- CUPS installed and configured (naturally)
- GhostScript installed

Only if you want to deliver PDF files by email, you'll need these too:

- Sendmail configured to deliver emails
- Perl installed, with the additional module MIME::Entity. If you don't have MIME::Entity installed, you can compile it yourself (it is part of MIME::tools, which in turn requires the IO::stringy and Mail perl modules), or use the program cpan2tgz to create Slackware packages for it yourself, or download my precompiled Slackware packages (in my repository).

Installing a new CUPS backend

CUPS uses so-called *backends* to support output to printing devices. Examples of these backends are *ipp*, *lpd* and *socket*; they show up in the CUPS URI that you define when creating a new CUPS print queue, for instance printing via IPP: "`ipp://192.168.0.1/print/hp6`".
We are going to create a new backend, which supports printing to PDF files and delivering these files to the correct end location (mailbox or directory path). The backend is going to be called **pdf** since that name is not yet being used.

- First off, let's start with creating the necessary directories (as root). The PDF files are going to be output to a temporary location (the *spool* directory) before being emailed to their final destination. We'll be using a new directory "`/var/spool/pdf`" for that, and make sure that everyone has write access there. If you do not want to use email delivery, this spool directory must be accessible by everyone who wants to use your pdfprinter. That means if you offer the pdfprinter as a network service to other computers in your network, you will have to export this spool directory using Samba or NFS. It is not within the scope of this article to describe how you would do that.

  The executables (in the form of Shell and Perl scripts) for the `pdf` backend are going to be stored in another new directory called `/usr/lib/cups/pdf`:

```
mkdir -p /var/spool/pdf
chmod 1777  /var/spool/pdf
mkdir /usr/lib/cups/pdf
chmod 775 /usr/lib/cups/pdf
```

The `chmod 1777 /var/spool/pdf` makes `/var/spool/pdf` writable for everyone, plus sets the *sticky bit* which means that other users will be unable to delete files that they do not own themselves.

- Next, we are going the create the file that provides or "pdf backend" functionality. Defining a new backend is really no more than creating an executable file called in CUPS's backends directory `/usr/lib/cups/backend` - this file's name will become the name of the backend. So, for the "pdf" backend we create the file `/usr/lib/cups/backend/pdf` and make this a shellscript, the contents of which you can find in the section below - PDF Printer Scripts. Do not forget to make this new "pdf" script executable:

```
chmod 755 /usr/lib/cups/backend/pdf
```

This backend will in turn call two other scripts which we shall place in the library directory `/usr/lib/cups/pdf` - see next bullet point.
The reason the script did not get world-executable permissions, is that the backend program needs to be run as the *root* user - CUPS will only do that if the program is not world-executable. Otherwise, the backend would be run using the unprivileged user account, typically "lp". *The script needs to run as root in order to be able to change ownership of the resulting PDF file to the user who printed it.*

- Now that we have the `pdf` backend file in place, the next step is to create the two helper scripts that our `pdf` script uses. One script does the conversion of PostScript to PDF, while the second does the actual email delivery. Install these two files called "`ps2pdf.cups`" and "`sendpdf`" into the cups library directory which we just created: `/usr/lib/cups/pdf`. You can find these scripts in the section below called PDF Printer Scripts. Do not forget to make these files executable:

```
chmod 755 /usr/lib/cups/pdf/ps2pdf.cups
chmod 755 /usr/lib/cups/pdf/sendpdf
```

The first script, `ps2pdf.cups` is similar to the `ps2pdf` script that comes with GhostScript. It's function is to take the PostScript data that CUPS generated from the submitted print job, and convert it into PDF. The `sendpdf` script takes this PDF file and delivers it via email.
The script as found in PDF Printer Scripts uses the `sendmail` program to deliver the PDF file to your inbox. It uses your logon name to determine the destination email address.

If you have not setup your sendmail for email delivery, or don't want email delivery of your PDF's, you can change the behaviour of the `/usr/lib/cups/backend/pdf` script. Edit the last part of that script, commenting out the email delivery part and *uncommenting* the directory part.
That piece of script would then look like this:

```
## Normally the PDF file will be emailed to the creating user.
## Alternatively, you can decide not to email it,
## but leave the file on the server and restrict access by others:
#if [ "$2" != "" ]; then
#  $MAILBIN $2 $OUTPUTFILENAME
#  rm -f $OUTPUTFILENAME
#fi
if [ "$2" != "" ]; then
       chown $2 $OUTPUTFILENAME
       chmod 700 $OUTPUTFILENAME
fi
exit 0
```

In case you wonder where the PDF files will now show up after printing to the "pdfprinter" queue: the output directory for the PDF files is defined when you create your queue, for instance using the `lpadmin` command (see the last bullet point for the exact `lpadmin` commandline parameters), or by using the browser-based administrative front-end to CUPS.
Whatever output directory you choose to use, remember to make it writeable for everyone and also set the *sticky bit* just like I did for my example directory `/var/spool/pdf` (see above).

Reminder: If you don't use the email delivery feature, you do not need to install the `sendpdf` script, and you don't have to deal with the Perl software requirements!

- CUPS uses *PPD* files (PostScript Printer Definition files) for the conversion of a submitted print job into PostScript print data which is correctly formatted for the printer backend. Usually the backend is a physical device (a printer) and the PPD file describes the device's capabilities. For our PDF backend which is not a physical device at all, we just need generic PPD file that can output color information in PostScript format (if you prefer grey-scale PDF files, you should of course get an appropriate non-colour PPD file).
  Adobe wrote a PPD file `distiller.ppd` which you can freely download at their web site . The Adobe site is often inaccessible, but fortunately there are many mirrors of that PPD file. Here is one: distiller.ppd.gz . A copy of the file is available in this Wiki as well: distiller.ppd.gz

  Another (really free) PPD file can be found on the cups-pdf web site, where you can find another implementation of a CUPS PDF print solution which I have not yet tried. The URL to this PPD file is PostscriptColor.ppd

  Save either of those files in the CUPS ppd-collection directory, under the name of `/usr/share/cups/model/pdfcolor.ppd` - the name does not really matter, you can keep the original filename if you wish, but since I am using that name `pdfcolor.ppd` in the rest of this example, I refer to it here as well to avoid confusion. CUPS PPD files are gzipped - probably to save space. It is not a requirement to do so for our own PPD file, but we will do it anyway inncase you did not download the "*.gz*" file:

  ```
  gzip /usr/share/cups/model/pdfcolor.ppd
  ```

  This creates the file `/usr/share/cups/model/pdfcolor.ppd.gz` which we will use in the `lpadmin` command further down.

- Finally, we need to restart the CUPS service to activate our new `pdf` backend, so we can create our pdfprinter queue .

  ```
  /etc/rc.d/rc.cups restart
  ```

- Create a new print queue using the commandline tool `lpadmin`.
  For our PDF backend, the command will look like this:

  ```
  lpadmin -p pdfprinter -v pdf:/var/spool/pdf/ -D "Generate PDF files" -E -P /usr/share/cups/model/
  ```

  This creates the print queue called **pdfprinter** using the **pdf** backend, with the directory **/var/spool/pdf** to generate PDF files in, and it will use **/usr/share/cups/model/pdfcolor.ppd.gz** as the PPD file for this printer definition.

  If you want to add the printer queue through the CUPS web interface, you should pick "*PDF Creator*" as the device and "*Adobe* | *Adobe Distiller*" as the Model/Type of the printer.

We're ready! The PDF printer setup is complete.

Using the PDF printer

Client computers (or you on your local computer) can create a PDF file of any printable data, by just printing to the CUPS print queue "*pdfprinter*". Linux client computers on the network which have CUPS configured as a client, will pick up the CUPS server automatically (CUPS services talk among each other on the network) so this will all work completely transparently.

If you have a network with Windows computers, the best thing is to setup a Samba server on the CUPS server and let Samba use CUPS for printing. That way, our "pdfprinter" will automatically become available to these Windows computers as a network printer. Windows machines should install this network printer queue using a *Generic PostScript* driver.

PDF Printer Scripts

- Script `/usr/lib/cups/backend/pdf`

```
#!/bin/sh
# ----------------------------------------------------------------------
# "/usr/lib/cups/backend/pdf":
# ----------------------------------------------------------------------
#
PDFBIN=/usr/lib/cups/pdf/ps2pdf.cups
MAILBIN=/usr/lib/cups/pdf/sendpdf
FILENAME=
# filename of the PDF File
PRINTTIME=`date +%Y-%m-%d_%H.%M.%S`
# no argument, prints available URIs
if [ $# -eq 0 ]; then
  if [ ! -x "$PDFBIN" ]; then
    exit 0
  fi
  echo "direct pdf \"Unknown\" \"PDF Creator\""
  exit 0
fi
# case of wrong number of arguments
if [ $# -ne 5 -a $# -ne 6 ]; then
  echo "Usage: pdf job-id user title copies options [file]"
  exit 1
fi
# get PDF directory from device URI, and check write status
PDFDIR=${DEVICE_URI#pdf:}
if [ ! -d "$PDFDIR" -o ! -w "$PDFDIR" ]; then
  echo "ERROR: directory $PDFDIR not writable"
  exit 1
fi
# generate output filename
OUTPUTFILENAME=
if [ "$3" = "" ]; then
  OUTPUTFILENAME="$PDFDIR/unknown.pdf"
else
  if [ "$2" != "" ]; then
    OUTPUTFILENAME="$PDFDIR/$2-$PRINTTIME.pdf"
  else
    OUTPUTFILENAME="$PDFDIR/$PRINTTIME.pdf"
  fi
fi
# run ghostscript
if [ $# -eq 6 ]; then
  $PDFBIN $6 $OUTPUTFILENAME >& /dev/null
else
  $PDFBIN - $OUTPUTFILENAME >& /dev/null
fi

# Make the file visible (but read-only except for owner);
# This is only needed when the username ($2) is not set,
# for instance when printing a test page from the web interface.
chmod 644 $OUTPUTFILENAME

## Normally the PDF file will be emailed to the creating user.
## Alternatively, you can decide not to email it,
## but leave the file on the server and restrict access by others:
if [ "$2" != "" ]; then
  $MAILBIN $2 $OUTPUTFILENAME
  rm -f $OUTPUTFILENAME
fi
#if [ "$2" != "" ]; then
#       chown $2 $OUTPUTFILENAME
#       chmod 700 $OUTPUTFILENAME
#fi
exit 0
```

```
# EOF
# ----------------------------------------------------------------
```

- Script `/usr/lib/cups/pdf/ps2pdf.cups`

```
#!/bin/sh
# ----------------------------------------------------------------
# "/usr/lib/cups/pdf/ps2pdf.cups":
# Convert PostScript to PDF.
# ----------------------------------------------------------------

OPTIONS=""
while true
do
        case "$1" in
        -*) OPTIONS="$OPTIONS $1" ;;
        *)  break ;;

        esac
        shift
done

if [ $# -lt 1 -o $# -gt 2 ]; then
        echo "Usage: `basename $0` [options...] input.ps [output.pdf]" 1>&2
        exit 1
fi

infile=$1;

if [ $# -eq 1 ]
then
        outfile=$1
else
        outfile=$2
fi

# Doing an initial 'save' helps keep fonts from being flushed between pages.
exec gs -q -dNOPAUSE -dBATCH -sDEVICE=pdfwrite \
        -sOutputFile=$outfile $OPTIONS -c save pop -f $infile

# EOF
# ----------------------------------------------------------------
```

- Script `/usr/lib/cups/pdf/sendpdf`

```
#!/usr/bin/perl
# ----------------------------------------------------------------
# "/usr/lib/cups/pdf/sendpdf":
# Parameter: Username, PDF File
# ----------------------------------------------------------------
use MIME::Entity;
# Get the ARGS
$to = $ARGV[0];
$pdffile = $ARGV[1];
# Set some variables
$mailpipe = '| /usr/sbin/sendmail -t -oi';
$from = "pdfprinter";
$subject ="PDF File";
$content = "Your PDF print";
# create the mail
my $mail = MIME::Entity->build( Type    => 'text/plain',
                                From    => $from,
                                To      => $to,
                                Subject => $subject,
                                Data    => $content
                              );
# Attach the pdf file
```

```
$mail->attach( Path     => $pdffile,
               Type     => 'application/pdf',
               Encoding => 'base64'
             );
# Open mailpipe and send the mail
open MAIL, "$mailpipe" or die "Could not open mailpipe \"$mailpipe\" !\n";
$mail->print(\*MAIL);
close MAIL;

# EOF
# ---------------------------------------------------------------------
```