Serial Laplink HOWTO

Willem J. Atsma

Revision History

Revision v2.1	2001−07−20	Revised by: wja

How to create a serial connection between two computers? This document details how to setup a "laplink" connection between two computers. Having had a bit of trouble getting the information to get this to work myself, this document should make it easy for you.

Table of Contents

1. Introduction

This very brief document describes how to set up a serial "laplink" connection between two Linux machines or a Linux and a Windows machine. The serial link consists of a "null−modem" cable and a communication protocol: PPP. I use such a setup to send files to and from my laptop; it can also quite nicely be used to install new software from a cd−rom when a laptop doesn't have one. I had more trouble than I should have had when setting up my serial laplink. Hopefully this document will solve your problems in one go (the ones to do with the serial cable at any rate).

Both server and client side of the setup I will describe could be either a Linux machine or a different operating system, as long as it supports the PPP protocol on a null−modem. The MS Windows implementation, as it turns out, is slightly different from standard, requiring some special attention. This document has two main sections. The first describes how to setup the server side. This is the main station with a serial port dedicated to a PPP connection. Alternatively the connection can be started when needed from the command line. The second part describes the configuration of the client that connects to the server to retrieve or upload files.

This documentation was prepared for pppd version 2.4.0, on a version 2.4.2 kernel.

Please let me know if you are having any particular difficulties and suggestions for improvements on this document; my email address is listed at the bottom.

---

2. On the server side

2.1. Configure pppd

The communication will be through a serial port. On the server we will connect through the "/dev/ttyS1" (COM2 in DOS/Windows) device. On your computer this may be different and you should modify the examples below accordingly. I force authentication because I use the configuration to provide a way to access files on a file server for a group of people. After logon they can use the Samba file shares to copy files to and from a laptop. If you are not concerned with security you can comment out the relevant options.

PPP (Point−to−Point Protocol) communication provides TCP/IP across a serial link. In other words: when you want to do internet−based browsing through a modem, you are likely to be using it. In Linux PPP is implemented by the PPP daemon "pppd". Its configuration is done through files in the "/etc/ppp/" directory. We will be using the following files:

/etc/ppp/options contains all general options for PPP connections

/etc/ppp/options.ttyS1

contains PPP options specific to connections through "/dev/ttyS1"

/etc/ppp/pap−secrets

contains authentication information

/etc/inittab

starts initial system services

The "/etc/ppp/options" file should look more or less like this:

```
#/etc/ppp/options
lock
#auth forces authorization from peer
#login makes authentication use the system password file
#NOTE: my pap-secrets allows anyone access, so if this is not specified
#      anyone could connect! If this is a machine on which you dial out
#      as well, then comment auth and login out and move them to
#      /etc/options.ttySn
auth
login
```

The lines starting with a "#" are comments. The "lock" parameter indicates that a lock file will be created to ensure exclusive access to the serial device, the "auth" parameter indicates that the client will need to authenticate itself, and "login" tells pppd to use the system user names and passwords for authentication.

Note that pppd will still check the "/etc/ppp/pap−secrets" file for user name and password information. A special, single line entry makes that pppd will only use the system's user information. The "/etc/ppp/pap−secrets" file looks as follows:

```
#/etc/ppp/pap-secrets
# Secrets for authentication using PAP
# client                 server                 secret         IP addresses
*                        *                      ""             ppp_laplink_client
```

This allows any machine with the IP address "ppp_laplink_client" to connect to the server, without using a password (you can see why it is handy to have the ppp_laplink entries in your /etc/hosts file). The "login" parameter in the "/etc/ppp/options" file, however, makes that the user name and password supplied by the client for authorization have to match the "/etc/ppp/pap−secrets" file *as well as* the system user name and password, so the connection will only succeed after a valid user name and password are provided.

Options specific to the serial line you are connecting with are placed in "/etc/ppp/options.ttySn", where n is the number of the serial device. My server uses "/dev/ttyS1", so the options go into... "/etc/ppp/options.ttyS1".

```
#/etc/ppp/options.ttyS1
asyncmap 0
crtscts
#local indicates that modem lines are not used
local
#silent causes pppd to wait until a connection is made from the other side
silent
#these are entries that exist in the /etc/hosts file
#the link does not work if this is at the end of this file - order matters!
ppp_laplink_server:ppp_laplink_client
#auth forces authorization from peer
#login makes authentication use the system password file
#NOTE: my pap-secrets allows anyone access, so if this is not specified
#         anyone could connect! If this is a server that will never use ppp
#         for dialing out, you should move auth and login to /etc/ppp/options
#auth
#login
#use PAP, not CHAP for authentication
require-pap
115200
```

All these options are well described in the pppd man page; a few of the key ones are explained with comments in the file. The "crtscts" parameter tells pppd to use hardware flow control. This is recommended because it is the fastest. Alternatively you could specify "xonxoff" to use software−based flow control − you would specify this if your null modem cable doesn't connect the RTS/CTS lines (unlikely if you bought your cable in the store). "115200" specifies the data transmission rate − if you have trouble connecting you might want to try with a lower speed. You can find valid speed settings in the termios manual page, although your hardware will limit the baud rate. Choosing an invalid speed setting will elicit an error message from the pppd daemon and apparently it then reverts to some default value.

Note that "auth" and "login" options are commented out here, because they were specified in the general options file. If you also use your computer to dial into an ISP, you will want to specify them here rather than in "/etc/ppp/options", or you will be asking your ISP to authorize itself when you dial in and that probably won't succeed. The reason why they are not specified in this file by default is because if you have other incoming PPP connections now or in the future, you want to make sure they are always authenticated. Remember that the "pap−secrets" as presented here gives zero protection.

Finally, the "ppp_laplink_server:ppp_laplink_client" entry specifies the local and remote IP address after the link is up. You can use actual IP numbers here (e.g. 192.168.0.1:192.168.1.1), or entries from the "/etc/hosts" file, like I have done. The nice thing of doing the latter is that you can use the names to refer to these links later. I also recommend you use IP numbers like the ones I used (192.168.0.1:192.168.1.1). These addresses are set aside for local networks and don't exist on the internet, so you are avoiding possible conflicts. After the link is up, the client can refer to the server with the IP address of ppp_laplink_server (192.168.0.1) and the server refers to the client with ppp_laplink_client (192.168.1.1).

You could use different entries in "/etc/ppp/pap−secrets" to only allow select users access. I am using the PAP protocol for authentication; you could use CHAP if you'd like − the setup is much the same, using the "chap−secrets" file. For these and other options you can consult the man pages and the documentation mentioned at the bottom.

2.2. A getty−like installation of pppd

You can have the PPP daemon (pppd) start when you boot the system and have it monitor the serial line of your choice. An elegant way of achieving this is to edit the "/etc/inittab" file. This file contains information for initializing the system. Add the following to this file:

```
# Start pppd for the serial laplink.
pd:2345:respawn:/usr/sbin/pppd /dev/ttyS1 nodetach
```

This reads as follows: for runlevels 2, 3, 4 and 5 start "/usr/sbin/pppd /dev/ttyS1 nodetach" and if it dies (at the end of a connection) respawn (start a new one). The "nodetach" option makes that pppd stays connected to the terminal that started it, rather than forking and exiting. This option is necessary because the "init" process would respawn a new one immediately otherwise. Other entries in the inittab file specify getty processes to run on serial terminals (tty's); their initialization looks a lot like this one.

To activate this new configuration type:

```
[root@griis /root]# /sbin/init q
```

2.3. Start the server when needed

If it is only occasionally that you want to connect to your server, you might prefer to start the connection manually. All the settings remain the same; you can start the server by simply typing:

```
/usr/sbin/pppd /dev/ttyS1 nodetach
```

at the command line. The "nodetach" option is not really necessary, but it makes it easy to kill the connection by pressing "ctrl−c".

2.4. Serving MS Windows clients

Unfortunately the MS Windows implementation is not quite standard. Before initiating the PPP connection it requires the exchange of the text strings "CLIENT" (from the client) and "CLIENTSERVER" (from the server). To accommodate a Windows client the following line has to be added to the "/etc/ppp/options.ttyS1" file:

```
connect 'chat -v -f /etc/ppp/scripts/winclient.chat'
```

Then create the scripts directory and the chat file "/etc/ppp/scripts/winclient.chat":

```
TIMEOUT 3600
CLIENT CLIENTSERVER\c
```

The connect option allows you to specify a program to deal with the string exchange before the connection. Usually the "chat" program is used for this; check the manual for more details. The given script deals with the Windows connection issue. You don't need it when connecting a Linux box.

3. On the client side

After having configured the server, the client is easy. The "/etc/ppp/options" file has a single entry:

```
#/etc/ppp/options
lock
```

The serial port on my client is "/dev/ttyS0" (COM1), so I also have to create a "/etc/ppp/options.ttyS0" file:

```
#/etc/ppp/options.ttyS0
115200
crtscts
local
user zaphod
noauth
```

Only a few new options in this file. Note that the speed on server and client has to be the same (here it is 115200). With "user zaphod" the client user name is specified. This user name has to correspond to an entry in the "/etc/ppp/pap−secrets" file of the client and has to be a valid user name on the server. The "noauth" option specifies that the peer (server) does not have to authenticate itself to the client.

The last bit: setting the client name (user name) and its secret (password) in the "/etc/ppp/pap−secrets" file:

```
#/etc/ppp/pap-secrets
# Secrets for authentication using PAP
# client                server                  secret              IP addresses
zaphod                  *                       gargleBlaster
```

So user "zaphod" uses a password "gargleBlaster", which should get him into the system. Note that these files contain sensitive information, make sure you set the permissions on them correctly.

3.1. Start the client

On the command line, type:

```
/usr/sbin/pppd /dev/ttyS0 nodetach
```

3.2. Connecting to an MS Windows server

What if you want to make a connection to a Windows server? Again we'll need the "connect" option and a chat script. Add the following line to your "/etc/ppp/options.ttyS0" file:

```
connect chat −v −f /etc/ppp/scripts/winserver.chat
```

Also create the chat script "/etc/ppp/script/winserver.chat":

```
TIMEOUT 10
'' CLIENT\c
```

This makes the client send the "CLIENT" string before trying to start the PPP connection.

3.3. Setting up an MS Windows (95) client

To connect to the server using windows, you use the Direct Cable Connection program, and set it up as "guest". In order for the windows client to connect at full speed (or at all), you will have to change the communication speed of direct cable connection. To do this, go to Control Panel−>System−>Device Manager−>Modems and select the serial cable. Proceed to the 'modem' option and select a maximum speed of 115200 or whatever your linux computer is set to. Now, it is important that you reboot at this stage, or it WILL NOT WORK (at least with Microsoft windows 95 version A), I am not sure about others.

3.4. Oddly enough...

Oddly enough I created a Linux server that has to connect to both Linux and Windows computers, so I found myself with the odd situation of having to setup both Linux systems such that they would connect to Windows servers and clients!

4. Connecting...

The moment of truth: connect the computers with the null modem cable, first start the server side (if you are using the "/etc/inittab" approach it should be up already), then start the client side. You should see something like this on the server:

```
[root@griis /root]# pppd /dev/ttyS1 nodetach
Using interface ppp0
Connect: ppp0 <--> /dev/ttyS1
user zaphod logged in
Deflate (15) compression enabled
local  IP address 192.168.0.1
remote IP address 192.168.1.1
```

Note that if you started pppd from the "/etc/inittab" file these messages will appear in "/var/log/messages". On the client side you should see something like this:

```
[root@wylg /root]# pppd /dev/ttyS0 nodetach
Using interface ppp0
Connect: ppp0 <--> /dev/ttyS0
Remote message: Success
Deflate (15) compression enabled
local  IP address 192.168.1.1
remote IP address 192.168.0.1
```

On either side you can ping the other computer to see if the connection is active:

```
[zaphod@wylg zaphod]$ ping 192.168.0.1
PING 192.168.0.1 (192.168.0.1) from 192.168.1.1 : 56(84) bytes of data.
64 bytes from 192.168.0.1: icmp_seq=0 ttl=255 time=31.7 ms
64 bytes from 192.168.0.1: icmp_seq=1 ttl=255 time=20.3 ms
64 bytes from 192.168.0.1: icmp_seq=2 ttl=255 time=19.2 ms
64 bytes from 192.168.0.1: icmp_seq=3 ttl=255 time=20.3 ms

--- 192.168.0.1 ping statistics ---
4 packets transmitted, 4 packets received, 0% packet loss
round-trip min/avg/max = 19.2/22.8/31.7 ms
```

You can now access the server through any TCP/IP services it might have available, such as FTP, HTTP, TELNET and SAMBA.

5. Internet through the serial port

Suppose you have a laptop and you want to access the internet occasionally. Also suppose you have a machine that has an ethernet connection and a serial port. You can set the laptop up as a client, the other machine as a server, and use *IP masquerading* to connect to the internet through the null modem cable. Check the IP−Masquerading−Howto for help on how to do this.

6. Other useful documents

- The pppd and chat manual pages
- PPP−HOWTO
- How to hook up PPP (http://axion.physics.ubc.ca/ppp−linux.html)
- Modem−HOWTO
- Serial−HOWTO
- IP−Masquerade−HOWTO

7. License

This document is covered by the Gnu Free Document License (http://www.gnu.org/copyleft/fdl.html). Please freely copy and distribute (sell or give away) this document in any format. It's requested that corrections and/or comments be fowarded to the document maintainer. You may create a derivative work and distribute it provided that you:

- Send your derivative work (in the most suitable format such as sgml) to the LDP (Linux Documentation Project) or the like for posting on the Internet.
- License the derivative work with this same license or use GPL. Include a copyright notice and at least a pointer to the license used.
- Give due credit to previous authors and major contributors.

If you're considering making a derived work other than a translation, it's requested that you discuss your plans with the current maintainer.

8. Credits

I have received a nice amount of helpful feedback, specifically from J.M. and B.S., and some of it was incorporated in this new version, thanks! author: W.J. Atsma email: watsma@mech.ubc.ca revision: 29jul2001