LINUX JOURNAL

(/)

A 10-Minute Guide for Using PPP to Connect Linux to the Internet

HOWTOs (/tag/howtos)

by Terry Dawson on April 1, 1997

Connecting your Linux machine to the Internet with PPP is easy in most situations. In this article I show you how to configure PPP for the most common type of connection. We assume your Linux machine is a stand-alone machine that dials into an Internet Service Provider and performs an automatic login, and the Internet Service Provider allocates the IP address that your machine will use. You can find details of how to configure PPP for other situations in the PPP-HOWTO by Robert Hart. You will need the right software and a couple of pieces of information before you start. Let's get started.

Preparation

First, check that you have the right software. The program that manages PPP for Linux is called **pppd**. The pppd program is linked very tightly with the kernel, so you must run a version of pppd that matches your kernel.

```
Kernel Version    pppd version
1.2.*             2.1.2d
1.3.0 -> 1.3.84   2.1.2d
1.3.84 -> 1.3.99  2.2.0f
2.0.*             2.2.0f
2.1.*             2.2.0f
```

Check the version of pppd and kernel that you have installed with the following commands:

```
$ /usr/sbin/pppd version
$ uname –a
```

The first command is a trick. The pppd command doesn't actually have a version option. However, the version number will appear in the error message pppd returns, since you have supplied it with a bad argument.
If the first command fails, you probably don't have PPP installed. You can obtain the latest version of the source from:

```
ftp://sunsite.unc.edu/pub/Linux/system/Network/serial/ppp/
```

If you have installed from a distribution such as Debian, Red Hat or Slackware, the pppd program is available precompiled within those distributions. You just have to get the package and install it.

Next you must check that your kernel has PPP support. Do this by giving the command:

```
$ dmesg | grep –i ppp
```

You should see the following messages:

```
PPP: version 2.2.0 (dynamic channel allocation)
PPP Dynamic channel allocation code copyright 1995 Caldera, Inc.
PPP line discipline registered.
```

If not, PPP may have been installed as a module. Become root and try:

```
# insmod ppp
```

If that fails, you will have to rebuild your kernel with PPP support. Follow the instructions in /usr/src/linux/README, and when configuring your kernel ensure that you answer “Yes” to:

```
General setup  --->
        [*] Networking support
Network device support  --->
        [*] Network device support
        <*> PPP (point-to-point) support
```

These prompts may be different in non-2.0 kernels.
Next you must note what keystrokes you will send and what prompts you will receive to log in to your ISP. The best way to collect these is to try manually logging into your ISP using a terminal program such as minicom. Be sure to make note of the capitalization of prompts such as the “login:” prompt as this will be important later.

A typical scenario follows:

```
Expect      Send                Comment
------      ----                -------
nothing     AT&F/r              (mode reset)
OK          AT&D2&C1/r          (mode initialization)
OK          AT&D555-9999/r      (modem dialing command)
```

The modem dials, sends CONNECT message and then you enter userid and password as follows:

```
login:  username/r
password: password/r
```

Lastly, you must know the IP address of a nameserver so that you can configure your name resolver and use host names instead of IP addresses. Get this information from your ISP.
Configuring PPP
The pppd program can accept configuration parameters from two places. The first is from the command line, and the second is from “options” files. The arguments supplied are close to identical in either case, but the command line method can be messy. So I will describe how to configure PPP using the options files instead.

The normal location of the options file is:

```
/etc/ppp/options
```

The options file is a simple text file containing parameters pppd will use when it is executed—one parameter per line. The options file must be readable by whoever will execute the pppd program. In most installations this will be root, either directly or by executing pppd from a program like sudo.

If you don't have an **/etc/ppp** directory, as root create one using the following commands:

```
# mkdir /etc/ppp
# chown root:root /etc/ppp
# chmod 755 /etc/ppp
```

Create an **/etc/ppp/options** file that looks like the following example:

```
debug
/dev/ttyS0
38400
modem
crtscts
lock
connect /etc/ppp/net-connect
asyncmap 0
defaultroute
:
```

This example assumes:

1. You want PPP to give you diagnostic information as it runs.
2. Your modem is connected to serial device /dev/ttyS0.
3. You want the serial port speed to be set at 38400 bps.
4. You want to listen to the Data Carrier Detect signal.
5. You will use hardware (RTS/CTS) handshaking.
6. Your dialer program is /etc/ppp/net-connect.
7. You have a full 8 bit clean connection.
8. By default datagrams should be sent via the PPP link.
9. You want the PPP server that you call to assign the IP address you will use.

These are all fairly typical defaults for an ISP connection. You will have to adjust the serial device to suit where you have your modem connected and, if you are using data compression, you might want to set your serial port speed to something higher. PPP provides a means of *escaping* select characters, so that they do not interfere with your connection. For example, if you were running PPP over a link that would disconnect if it

received a control-D character, you could ask PPP to escape that character, and it would automatically replace it with another and reverse the process at the other end. While the default is safe, it escapes a number of characters that normally don't need escaping and this will decrease the performance of your link. Since most ISPs provide 8 bit clean links you don't need to escape any characters, so we tell pppd not to, using the asyncmap option.

The pppd package includes a program called **chat**. The chat program is a simple program that can be used to automate the dialing procedure. The chat program also accepts arguments from the command line or from a file. Again I'll describe how to configure it from a file as this is the better method.

To make use of the chat program from within pppd, we must ensure that the **connect** option points to a script that calls chat. Create a script called **/etc/ppp/net-connect** that looks like:

```
#!/bin/sh
/usr/sbin/chat –v –t 60 –f /etc/ppp/net–chat
```

This shell script will invoke the chat command with the **-v**, **-t** and **-f** arguments. The **-v** argument is useful when you are configuring pppd, as it sends verbose diagnostic messages to the system log to show you what is happening as the chat program runs. The **-t 60** argument simply tells the chat program to wait 60 seconds for the expected text to arrive before timing out with an error. The **-f** argument tells chat the name of the file it should use to get the expect/send sequences it will use to login.

Make sure the script is readable and executable by whoever will invoke pppd. Assuming again that “whoever” is **root**, use the following commands:

```
# chmod 500 /etc/ppp/net–connect
# chown root:root /etc/ppp/net–connect
```

Create a chat script called **/etc/ppp/net-chat** that will automate the login sequence as described earlier. I will base this script on the details presented in the table.

```
ABORT "BUSY"
ABORT "NO CARRIER"
""              AT&F\r
OK                AT&D2&C1\r
OK                ATD555-9999\r
ogin:
sword:
```

The first two lines are special. The **ABORT** keyword is a special token that allows you to specify strings of characters that will cause the chat program to exit. In the example presented, if the chat program receives either the string **"BUSY"** or the string **"NO CARRIER"** then it will abort immediately. The rest of the file is a simple list of expect/send pairs, based on the information we gathered when we manually logged in. The above example reads in full:

> ABORT the script if we receive **"BUSY"** or **"NO CARRIER"**. Expect nothing, then send **AT&F<carriage-return>** to reset the modem to factory configuration, expect to receive **OK** then send **AT&D2&C1<carriage-return>**, then expect **OK** and send **ATD555-9999<carriage-return>**, then expect **login:** and send **username<carriage-return>**, then expect **sword:** and send **password<carriage-return>**, and then exit normally.

There are a couple of important points to note in this example. First, the modem initialization string I've suggested will, in most modems, ensure that the modem will raise the Data Carrier Detect line when a call is connected, and will hang up the call if the DTR line is lowered. This ensures that the modem is matched with the **modem** option supplied to pppd. Second, I haven't used the full prompt, but only the last few characters. This is generally good practice because under some circumstances the first characters from a line may be dropped. Looking only for the last few characters ensures our login succeeds even if this occurs. Finally, you will notice the <carriage-return> is coded as **\r**. There are a range of other characters may be encoded and sent in this way, if necessary. The chat man page explains what they are should you need to use them.
Finally, we must ensure this script is readable by whoever will invoke pppd. Again assuming that whoever is be **root**, you can use the following commands:

```
# chown root:root /etc/ppp/net-chat
# chmod 600 /etc/ppp/net-chat
```

Configuring the Name Resolver

The name resolver is a small piece of software within the standard Linux library that allows automatic conversion of a host name, e.g., sunsite.unc.edu, into an IP address, e.g., 152.2.254.81.

Configuration of the name resolver is easy; there is only one file to change. You will almost certainly already have this file on your machine, but you will need to configure the correct address for the nameserver. Assuming your ISP supplied you with a nameserver address of **128.78.64.10** then your **/etc/resolv.conf** file should have a line that says:

```
nameserver 128.78.64.10
```

Starting the Link

To start the PPP link, all you need to do is execute the following command as root:

```
# /usr/sbin/pppd
```

The pppd program will start and will search for its options in the standard locations. It will find our options file at /etc/ppp/options and read each line. When it has finished processing all available options, it will open the specified serial device, create a lock file to prevent other programs from trying to use it, and then attempt to run the connect program and to execute the /etc/ppp/net-connect script. The net-connect script will execute the chat program telling it that it should take its parameters from the /etc/ppp/net-chat file. The chat program starts, reads each of the lines from the net-chat file, waits for the strings, and sends the responses it has been given. Provided the chat program did not **ABORT** then control is passed back to the pppd program, which will then switch the line into PPP mode and create a PPP network device. The pppd program will automatically begin negotiation of some configuration details with the PPP program at the other end of the link. The most important of these details is the IP address you will use. The pppd program will create a ppp network device **ppp0** and then configure it with the details it obtained from the other program. Finally, the pppd program will configure your routing table with a route that tells your Linux machine it should send datagrams to the PPP link, if it doesn't have anywhere better to send them. The pppd program will then sit happily in the background until either the line fails, the remote end closes the connection or you terminate it locally.

Okay, that sounds complicated, so a summary:

1. pppd starts.
2. pppd reads /etc/ppp/options.
3. pppd executes /etc/ppp/net-connect.
4. chat reads data from /etc/ppp/net-chat.

5. pppd obtains IP address details from server.
6. pppd creates ppp0 device and configures it.
7. pppd creates default route.
8. pppd runs in background.

Testing the Connection

To test the connection, do each of the following steps in turn.

Step 1:

```
run /sbin/ifconfig
```

The ifconfig program is used to set or display network interface configurations. Here you are interested in displaying only.

Step 2:

```
# /sbin/ifconfig
```

The output should look like Listing 1.

Listing 1 (/files/linuxjournal.com/linuxjournal/articles/021/2109/2109l1.html)

The **inet addr** field is the IP address you have been allocated. The **P-t-P** field is the address of the PPP machine at the other end of the link. This means your PPP network connection has been successfully established.

If you don't see a **ppp0** device, check your system log file, i.e., /var/adm/messages, to ensure that your chat script worked successfully. Correct any possible errors. If you see any nasty looking error messages, double check that you are using the correct version of PPP for your kernel.

Step 3: ping the PPP Remote Host. The ping command sends specially formatted datagrams to a host that that host will send replies to. This allows us to check that we have a working route to that host. Listing 2 shows our case. Those "**64 bytes from ...**" lines in the listing mean we are talking successfully to the machine at the other end of the link. This is good, since it means the link is working.

Listing 2 (/files/linuxjournal.com/linuxjournal/articles/021/2109/2109l2.html)

If you don't see any of the "**64 bytes from ...**" lines, this means you are not properly talking to the remote machine. Double check your chat script and the system log file.

Step 4: ping your nameserver. This is an important test to be sure the default route pppd put in place is working. To do this, ping the nameserver address configured into the /etc/resolv.conf file. In our case:

```
# ping 128.78.64.10
```

Output will be similar to what you observed when you pinged the PPP server.

Listing 3 (/files/linuxjournal.com/linuxjournal/articles/021/2109/2109l3.html)

If this test fails, it could mean your default route hasn't been added properly. To double check, run the route command as shown in Listing 3. The route command displays the contents of the IP routing table. The **-n** option tells it not to try and convert IP addresses into host names. The line starting with **0.0.0.0** is the default route. If you don't see a line like this, double check that you have included the **defaultroute** option in the /etc/ppp/options file. If you have a line like this but it doesn't point to ppp0, check that your system isn't already creating a default route to another device. If it is, find which **rc** file is doing it and comment out this entry.

Step 5: ping a remote host. This is the real and simple test. Try either:

```
# ping sunsite.unc.edu
```

or:

```
# ftp ftp.funet.fi
```

If this works, you are connected properly to the Internet. Enjoy.
If the command just sits there and, after a minute or so, gives you an error message about being unable to resolve the host name, check that you have modified your /etc/resolv.conf file correctly, and that the IP address you have configured there is the correct IP address for your ISP's nameserver.
Dropping a Connection
To drop a connection you just need to kill pppd. When it exits, it will hang up the line, if you've configured the modem as I've suggested.

On most distributions this will be as simple as:

```
# killall –HUP pppd
```

Making PPP Automatically Redial

If you are lucky enough to have a semi-permanent connection to your ISP, i.e., one where you can stay connected for as long as you like, you may want to have your Linux automatically redial if the telephone call drops out for some reason. Here is a simple way of doing this that assumes you have configured your PPP link to be activated by root.

The first very important step is to add this line to your /etc/ppp/options file:

```
–detach
```

This line tells pppd **not** to go into the background after it has successfully connected. The next step is to add a line to your /etc/inittab file that looks like this:

```
pd:23:respawn:/usr/sbin/pppd
```

Put this line down with the other lines that are similiar to it—the ones that run the login program.
This line simply tells the init program that it should automatically start the /usr/sbin/pppd program and that it should automatically restart it if it dies. Provided you have your modem configured to raise Data Carrier Detect and you have configured pppd as I have described, **init** will ensure the pppd program is always running and re-run it if it terminates.

A word of warning—this is simple, but provides no safeguards against problems that might cause the telephone call to be successfully made and then hang up. If you experience this problem, the init program will quite happily keep re-running the pppd program until you tell it to stop. You could run up quite a telephone bill if something nasty goes wrong.
Conclusion
This article describes a basic PPP configuration. There are many excellent documents that provide more detailed and comprehensive information on the subject. This article should be sufficient to get you connected to the Internet in a typical configuration. If you have any problems you cannot diagnose, I strongly recommend you read the PPP-HOWTO by Robert Hart at:

```
http://sunsite.unc.edu/LDP/HOWTO/PPP–HOWTO.html
```

Robert has done an excellent job in rewriting the HOWTO, and it should be of assistance to you.

Terry Dawson is the author of a number of Linux HOWTO documents including the AX25-HOWTO, IPX-HOWTO and the NET-2-HOWTO. Terry has been an advocate of Linux from the moment he booted Linux 0.12 and saw the potential for Linux to significantly enhance experimentation in networking protocols in Amateur Radio. He can be reached via e-mail at terry@perf.no.itg.telecom.com.au.

You May Like

Running GNOME in a Container (/content/running-gnome-container)

Adam Verslype (/users/adam-verslype)

(/content/running-gnome-container)

Digital Will, Part I: Requirements (/content/digital-will-part-i-requirements)

Kyle Rankin (/users/kyle-rankin)

(/content/digital-will-part-i-requirements)

Build Your Own Internet Radio Receiver (/content/build-your-own-internet-radio-receiver)

Nick Tufillaro (/users/nick-tufillaro)

(/content/build-your-own-internet-radio-receiver)

Testing Models (/content/testing-models)

Reuven M. Lerner (/users/reuven-m-lerner)

(/content/testing-models)

Connect With Us

(https://youtube.com/linuxjournalonline)

(https://www.facebook.com/linuxjournal/) (https://twitter.com/linuxjournal)

Linux Journal, representing 25+ years of publication, is the original magazine of the global Open Source community.