NAME

ppp - Point to Point Protocol (a.k.a. user-ppp)

SYNOPSIS

ppp [-*mode*] [**-nat**] [**-quiet**] [**-unit***N*] [*system* ...]

DESCRIPTION

This is a user process *PPP* software package. Sometimes, *PPP* is implemented as a part of the kernel (e.g., as managed by **pppd**) and it is thus somewhat hard to debug and/or modify its behaviour. However, in this implementation *PPP* is done as a user process with the help of the tunnel device driver (tun).

The **-nat** flag does the equivalent of a "nat enable yes", enabling **ppp**'s network address translation features. This allows **ppp** to act as a NAT or masquerading engine for all machines on an internal LAN. Refer to libalias(3) for details on the technical side of the NAT engine. Refer to the *NETWORK ADDRESS TRANSLATION (PACKET ALIASING)* section of this manual page for details on how to configure NAT in **ppp**.

The **-quiet** flag tells **ppp** to be silent at startup rather than displaying the mode and interface to standard output.

The **-unit** flag tells **ppp** to only attempt to open */dev/tunN*. Normally, **ppp** will start with a value of 0 for *N*, and keep trying to open a tunnel device by incrementing the value of *N* by one each time until it succeeds. If it fails three times in a row because the device file is missing, it gives up.

The following *mode*s are understood by **ppp**:

-auto **ppp** opens the tun interface, configures it then goes into the background. The link is not brought up until outgoing data is detected on the tun interface at which point **ppp** attempts to bring up the link. Packets received (including the first one) while **ppp** is trying to bring the link up will remain queued for a default of 2 minutes. See the "set choked" command below.

In **-auto** mode, at least one "system" must be given on the command line (see below) and a "set ifaddr" must be done in the system profile that specifies a peer IP address to use when configuring the interface. Something like "10.0.0.1/0" is usually appropriate. See the "pmdemand" system in */usr/share/examples/ppp/ppp.conf.sample* for an example.

-background

Here, **ppp** attempts to establish a connection with the peer immediately. If it succeeds, **ppp**

goes into the background and the parent process returns an exit code of 0. If it fails, **ppp** exits with a non-zero result.

-foreground
In foreground mode, **ppp** attempts to establish a connection with the peer immediately, but never becomes a daemon. The link is created in background mode. This is useful if you wish to control **ppp**'s invocation from another process.

-direct
This is used for communicating over an already established connection, usually when receiving incoming connections accepted by getty(8). **ppp** ignores the "set device" line and uses descriptor 0 as the link. **ppp** will also ignore any configured chat scripts unless the "force-scripts" option has been enabled.

If callback is configured, **ppp** will use the "set device" information when dialing back.

When run in **-direct** mode, **ppp** will behave slightly differently if descriptor 0 was created by pipe(2). As pipes are not bi-directional, ppp will redirect all writes to descriptor 1 (standard output), leaving only reads acting on descriptor 0. No special action is taken if descriptor 0 was created by socketpair(2).

-dedicated
This option is designed for machines connected with a dedicated wire. **ppp** will always keep the device open and will ignore any configured chat scripts unless the "force-scripts" option has been enabled.

-ddial
This mode is equivalent to **-auto** mode except that **ppp** will bring the link back up any time it is dropped for any reason.

-interactive
This is a no-op, and gives the same behaviour as if none of the above modes have been specified. **ppp** loads any sections specified on the command line then provides an interactive prompt.

One or more configuration entries or systems (as specified in */etc/ppp/ppp.conf*) may also be specified on the command line. **ppp** will read the "default" system from */etc/ppp/ppp.conf* at startup, followed by each of the systems specified on the command line.

Major Features

Provides an interactive user interface. Using its command mode, the user can easily enter commands to establish the connection with the remote end, check the status of connection and close the connection. All functions can also be optionally password protected for security.

Supports both manual and automatic dialing. Interactive mode has a "term" command which enables you to talk to the device directly. When you are connected to the remote peer and it starts to talk *PPP*, **ppp** detects it and switches to packet mode automatically. Once you have determined the proper sequence for connecting with the remote host, you can write a chat script to {define} the necessary dialing and login procedure for later convenience.

Supports on-demand dialup capability. By using **-auto** mode, **ppp** will act as a daemon and wait for a packet to be sent over the *PPP* link. When this happens, the daemon automatically dials and establishes the connection. In almost the same manner **-ddial** mode (direct-dial mode) also automatically dials and establishes the connection. However, it differs in that it will dial the remote site any time it detects the link is down, even if there are no packets to be sent. This mode is useful for full-time connections where we worry less about line charges and more about being connected full time. A third **-dedicated** mode is also available. This mode is targeted at a dedicated link between two machines. **ppp** will never voluntarily quit from dedicated mode - you must send it the "quit all" command via its diagnostic socket. A SIGHUP will force an LCP renegotiation, and a SIGTERM will force it to exit.

Supports client callback. **ppp** can use either the standard LCP callback protocol or the Microsoft CallBack Control Protocol (https://winprotocoldoc.blob.core.windows.net/productionwindowsarchives/MS-CBCP/[MS-CBCP].pdf).

Supports NAT or packet aliasing. Packet aliasing (a.k.a. IP masquerading) allows computers on a private, unregistered network to access the Internet. The *PPP* host acts as a masquerading gateway. IP addresses as well as TCP and UDP port numbers are NAT'd for outgoing packets and de-NAT'd for returning packets.

Supports background PPP connections. In background mode, if **ppp** successfully establishes the connection, it will become a daemon. Otherwise, it will exit with an error. This allows the setup of scripts that wish to execute certain commands only if the connection is successfully established.

Supports server-side PPP connections. In direct mode, **ppp** acts as server which accepts incoming *PPP* connections on stdin/stdout.

Supports PAP and CHAP (rfc 1994, 2433 and 2759) authentication. With PAP or CHAP, it is possible to skip the Unix style login(1) procedure, and use the *PPP* protocol for authentication instead. If the peer requests Microsoft CHAP authentication and **ppp** is compiled with DES support, an appropriate

MD4/DES response will be made.

Supports RADIUS (rfc 2138 & 2548) authentication. An extension to PAP and CHAP, *R*emote *A*ccess *D*ial *I*n *U*ser *S*ervice allows authentication information to be stored in a central or distributed database along with various per-user framed connection characteristics. If libradius(3) is available at compile time, **ppp** will use it to make *RADIUS* requests when configured to do so.

Supports Proxy Arp. ppp can be configured to make one or more proxy arp entries on behalf of the peer. This allows routing from the peer to the LAN without configuring each machine on that LAN.

Supports packet filtering. User can {define} four kinds of filters: the *in* filter for incoming packets, the *out* filter for outgoing packets, the *dial* filter to {define} a dialing trigger packet and the *alive* filter for keeping a connection alive with the trigger packet.

Tunnel driver supports bpf. The user can use tcpdump(1) to check the packet flow over the *PPP* link.

Supports PPP over TCP and PPP over UDP. If a device name is specified as *host*:*port*[/tcp|udp], **ppp** will open a TCP or UDP connection for transporting data rather than using a conventional serial device. UDP connections force **ppp** into synchronous mode.

Supports PPP over Ethernet (rfc 2516). If **ppp** is given a device specification of the format PPPoE:*iface*[:*provider*]and if netgraph(4) is available, **ppp** will attempt talk *PPP* over Ethernet to *provider* using the *iface* network interface.

On systems that do not support netgraph(4), an external program such as pppoed(8) may be used.

Supports IETF draft Predictor-1 (rfc 1978) and DEFLATE (rfc 1979) compression. ppp supports not only VJ-compression but also Predictor-1 and DEFLATE compression. Normally, a modem has built-in compression (e.g., v42.bis) and the system may receive higher data rates from it as a result of such compression. While this is generally a good thing in most other situations, this higher speed data imposes a penalty on the system by increasing the number of serial interrupts the system has to process in talking to the modem and also increases latency. Unlike VJ-compression, Predictor-1 and DEFLATE compression pre-compresses *all* network traffic flowing through the link, thus reducing overheads to a minimum.

Supports Microsoft's IPCP extensions (rfc 1877). Name Server Addresses and NetBIOS Name Server Addresses can be negotiated with clients using the Microsoft *PPP* stack (i.e., Win95, WinNT)

Supports Multi-link PPP (rfc 1990) It is possible to configure **ppp** to open more than one physical connection to the peer, combining the bandwidth of all links for better throughput.

Supports MPPE (draft-ietf-pppext-mppe) MPPE is Microsoft Point to Point Encryption scheme. It is possible to configure **ppp** to participate in Microsoft's Windows VPN. For now, **ppp** can only get encryption keys from CHAP 81 authentication. **ppp** must be compiled with DES for MPPE to operate.

Supports IPV6CP (rfc 2023). An IPv6 connection can be made in addition to or instead of the normal IPv4 connection.

PERMISSIONS

ppp is installed as user root and group network, with permissions 04554. By default, **ppp** will not run if the invoking user id is not zero. This may be overridden by using the "allow users" command in */etc/ppp/ppp.conf*. When running as a normal user, **ppp** switches to user id 0 in order to alter the system routing table, set up system lock files and read the ppp configuration files. All external commands (executed via the "shell" or "!bg" commands) are executed as the user id that invoked **ppp**. Refer to the 'ID0' logging facility if you are interested in what exactly is done as user id zero.

GETTING STARTED

When you first run **ppp** you may need to deal with some initial configuration details.

- Make sure that your system has a group named "network" in the */etc/group* file and that the group contains the names of all users expected to use **ppp**. Refer to the group(5) manual page for details. Each of these users must also be given access using the "allow users" command in */etc/ppp/ppp.conf*.

- Create a log file. **ppp** uses syslog(3) to log information. A common log file name is */var/log/ppp.log*. To make output go to this file, put the following lines in the */etc/syslog.conf* file:

```
!ppp
*.*<TAB>/var/log/ppp.log
```

It is possible to have more than one *PPP* log file by creating a link to the **ppp** executable:

```
# cd /usr/sbin
# ln ppp ppp0
```

and using

```
!ppp0
*.*<TAB>/var/log/ppp0.log
```

in */etc/syslog.conf*. Do not forget to send a HUP signal to syslogd(8) after altering */etc/syslog.conf*.

- Although not strictly relevant to **ppp**'s operation, you should configure your resolver so that it works correctly. This can be done by configuring a local DNS resolver or by adding the correct 'nameserver' lines to the file */etc/resolv.conf*. Refer to the resolv.conf(5) manual page for details.

  Alternatively, if the peer supports it, **ppp** can be configured to ask the peer for the nameserver address(es) and to update */etc/resolv.conf* automatically. Refer to the "enable dns" and "resolv" commands below for details.

MANUAL DIALING

In the following examples, we assume that your machine name is awfulhak. when you invoke **ppp** (see *PERMISSIONS* above) with no arguments, you are presented with a prompt:

```
ppp ON awfulhak>
```

The 'ON' part of your prompt should always be in upper case. If it is in lower case, it means that you must supply a password using the "passwd" command. This only ever happens if you connect to a running version of **ppp** and have not authenticated yourself using the correct password.

You can start by specifying the device name and speed:

```
ppp ON awfulhak> set device /dev/cuau0
ppp ON awfulhak> set speed 38400
```

Normally, hardware flow control (CTS/RTS) is used. However, under certain circumstances (as may happen when you are connected directly to certain PPP-capable terminal servers), this may result in **ppp** hanging as soon as it tries to write data to your communications link as it is waiting for the CTS (clear to send) signal - which will never come. Thus, if you have a direct line and cannot seem to make a connection, try turning CTS/RTS off with "set ctsrts off". If you need to do this, check the "set accmap" description below too - you will probably need to "set accmap 000a0000".

Usually, parity is set to "none", and this is **ppp**'s default. Parity is a rather archaic error checking mechanism that is no longer used because modern modems do their own error checking, and most link-layer protocols (that is what **ppp** is) use much more reliable checking mechanisms. Parity has a relatively huge overhead (a 12.5% increase in traffic) and as a result, it is always disabled (set to "none") when PPP is opened. However, some ISPs (Internet Service Providers) may use specific parity settings at connection time (before PPP is opened). Notably, Compuserve insist on even parity when logging in:

```
ppp ON awfulhak> set parity even
```

You can now see what your current device settings look like:

```
ppp ON awfulhak> show physical
Name: deflink
 State:          closed
 Device:         N/A
 Link Type:      interactive
 Connect Count:  0
 Queued Packets: 0
 Phone Number:   N/A

Defaults:
 Device List:    /dev/cuau0
 Characteristics: 38400bps, cs8, even parity, CTS/RTS on

Connect time: 0 secs
0 octets in, 0 octets out
Overall 0 bytes/sec
ppp ON awfulhak>
```

The term command can now be used to talk directly to the device:

```
ppp ON awfulhak> term
at
OK
atdt123456
CONNECT
login: myispusername
Password: myisppassword
Protocol: ppp
```

When the peer starts to talk in *PPP*, **ppp** detects this automatically and returns to command mode.

```
ppp ON awfulhak>            # No link has been established
Ppp ON awfulhak>            # We've connected & finished LCP
PPp ON awfulhak>            # We've authenticated
PPP ON awfulhak>            # We've agreed IP numbers
```

If it does not, it is probable that the peer is waiting for your end to start negotiating. To force **ppp** to start sending *PPP* configuration packets to the peer, use the "~p" command to drop out of terminal mode and enter packet mode.

If you never even receive a login prompt, it is quite likely that the peer wants to use PAP or CHAP authentication instead of using Unix-style login/password authentication. To set things up properly, drop back to the prompt and set your authentication name and key, then reconnect:

```
~.
ppp ON awfulhak> set authname myispusername
ppp ON awfulhak> set authkey myisppassword
ppp ON awfulhak> term
at
OK
atdt123456
CONNECT
```

You may need to tell ppp to initiate negotiations with the peer here too:

```
~p
ppp ON awfulhak>          # No link has been established
Ppp ON awfulhak>          # We've connected & finished LCP
PPp ON awfulhak>          # We've authenticated
PPP ON awfulhak>          # We've agreed IP numbers
```

You are now connected! Note that 'PPP' in the prompt has changed to capital letters to indicate that you have a peer connection. If only some of the three Ps go uppercase, wait until either everything is uppercase or lowercase. If they revert to lowercase, it means that **ppp** could not successfully negotiate with the peer. A good first step for troubleshooting at this point would be to

```
ppp ON awfulhak> set log local phase lcp ipcp
```

and try again. Refer to the "set log" command description below for further details. If things fail at this point, it is quite important that you turn logging on and try again. It is also important that you note any prompt changes and report them to anyone trying to help you.

When the link is established, the show command can be used to see how things are going:

```
PPP ON awfulhak> show physical
* Modem related information is shown here *
PPP ON awfulhak> show ccp
* CCP (compression) related information is shown here *
PPP ON awfulhak> show lcp
* LCP (line control) related information is shown here *
```

```
PPP ON awfulhak> show ipcp
* IPCP (IP) related information is shown here *
PPP ON awfulhak> show ipv6cp
* IPV6CP (IPv6) related information is shown here *
PPP ON awfulhak> show link
* Link (high level) related information is shown here *
PPP ON awfulhak> show bundle
* Logical (high level) connection related information is shown here *
```

At this point, your machine has a host route to the peer. This means that you can only make a connection with the host on the other side of the link. If you want to add a default route entry (telling your machine to send all packets without another routing entry to the other side of the *PPP* link), enter the following command:

```
PPP ON awfulhak> add default HISADDR
```

The string 'HISADDR' represents the IP address of the connected peer. If the "add" command fails due to an existing route, you can overwrite the existing route using:

```
PPP ON awfulhak> add! default HISADDR
```

This command can also be executed before actually making the connection. If a new IP address is negotiated at connection time, **ppp** will update your default route accordingly.

You can now use your network applications (ping, telnet, ftp, etc.) in other windows or terminals on your machine. If you wish to reuse the current terminal, you can put **ppp** into the background using your standard shell suspend and background commands (usually "^Z" followed by "bg").

Refer to the *PPP COMMAND LIST* section for details on all available commands.

AUTOMATIC DIALING

To use automatic dialing, you must prepare some Dial and Login chat scripts. See the example definitions in */usr/share/examples/ppp/ppp.conf.sample* (the format of */etc/ppp/ppp.conf* is pretty simple). Each line contains one comment, inclusion, label or command:

- A line starting with a ("#") character is treated as a comment line. Leading whitespace are ignored when identifying comment lines.

- An inclusion is a line beginning with the word '{!include}'. It must have one argument - the file to {include}. You may wish to "{!include} ~/.ppp.conf" for compatibility with older versions of **ppp**.

- A label name starts in the first column and is followed by a colon (":").

- A command line must contain a space or tab in the first column.

- A string starting with the "$" character is substituted with the value of the environment variable by the same name. Likewise, a string starting with the "~" character is substituted with the full path to the home directory of the user account by the same name, and the "~" character by itself is substituted with the full path to the home directory of the current user. If you want to include a literal "$" or "~" character in a command or argument, enclose them in double quotes, e.g.,

  ```
  set password "pa$ss~word"
  ```

The */etc/ppp/ppp.conf* file should consist of at least a "default" section. This section is always executed. It should also contain one or more sections, named according to their purpose, for example, "MyISP" would represent your ISP, and "ppp-in" would represent an incoming **ppp** configuration. You can now specify the destination label name when you invoke **ppp**. Commands associated with the "default" label are executed, followed by those associated with the destination label provided. When **ppp** is started with no arguments, the "default" section is still executed. The load command can be used to manually load a section from the */etc/ppp/ppp.conf* file:

```
ppp ON awfulhak> load MyISP
```

Note, no action is taken by **ppp** after a section is loaded, whether it is the result of passing a label on the command line or using the "load" command. Only the commands specified for that label in the configuration file are executed. However, when invoking **ppp** with the **-background**, **-ddial**, or **-dedicated** switches, the link mode tells **ppp** to establish a connection. Refer to the "set mode" command below for further details.

Once the connection is made, the 'ppp' portion of the prompt will change to 'PPP':

```
# ppp MyISP
...
ppp ON awfulhak> dial
Ppp ON awfulhak>
PPp ON awfulhak>
PPP ON awfulhak>
```

The Ppp prompt indicates that **ppp** has entered the authentication phase. The PPp prompt indicates that **ppp** has entered the network phase. The PPP prompt indicates that **ppp** has successfully negotiated a network layer protocol and is in a usable state.

If the */etc/ppp/ppp.linkup* file is available, its contents are executed when the *PPP* connection is established. See the provided "pmdemand" example in */usr/share/examples/ppp/ppp.conf.sample* which runs a script in the background after the connection is established (refer to the "shell" and "bg" commands below for a description of possible substitution strings). Similarly, when a connection is closed, the contents of the */etc/ppp/ppp.linkdown* file are executed. Both of these files have the same format as */etc/ppp/ppp.conf*.

In previous versions of **ppp**, it was necessary to re-add routes such as the default route in the *ppp.linkup* file. **ppp** supports 'sticky routes', where all routes that contain the HISADDR, MYADDR, HISADDR6 or MYADDR6 literals will automatically be updated when the values of these variables change.

BACKGROUND DIALING

If you want to establish a connection using **ppp** non-interactively (such as from a crontab(5) entry or an at(1) job) you should use the **-background** option. When **-background** is specified, **ppp** attempts to establish the connection immediately. If multiple phone numbers are specified, each phone number will be tried once. If the attempt fails, **ppp** exits immediately with a non-zero exit code. If it succeeds, then **ppp** becomes a daemon, and returns an exit status of zero to its caller. The daemon exits automatically if the connection is dropped by the remote system, or it receives a TERM signal.

DIAL ON DEMAND

Demand dialing is enabled with the **-auto** or **-ddial** options. You must also specify the destination label in */etc/ppp/ppp.conf* to use. It must contain the "set ifaddr" command to {define} the remote peers IP address. (refer to */usr/share/examples/ppp/ppp.conf.sample*)

```
# ppp -auto pmdemand
```

When **-auto** or **-ddial** is specified, **ppp** runs as a daemon but you can still configure or examine its configuration by using the "set server" command in */etc/ppp/ppp.conf*, (for example, "set server +3000 mypasswd") and connecting to the diagnostic port as follows:

```
# pppctl 3000        (assuming tun0)
Password:
PPP ON awfulhak> show who
tcp (127.0.0.1:1028) *
```

The "show who" command lists users that are currently connected to **ppp** itself. If the diagnostic socket is closed or changed to a different socket, all connections are immediately dropped.

In **-auto** mode, when an outgoing packet is detected, **ppp** will perform the dialing action (chat script) and try to connect with the peer. In **-ddial** mode, the dialing action is performed any time the line is found to

be down. If the connect fails, the default behaviour is to wait 30 seconds and then attempt to connect when another outgoing packet is detected. This behaviour can be changed using the "set redial" command:

set redial *secs*[+*inc*[-*max*]][.*next*] [*attempts*]

secs is the number of seconds to wait before attempting to connect again. If the argument is the literal string 'random', the delay period is a random value between 1 and 30 seconds inclusive.

inc is the number of seconds that *secs* should be incremented each time a new dial attempt is made. The timeout reverts to *secs* only after a successful connection is established. The default value for *inc* is zero.

max is the maximum number of times **ppp** should increment *secs*. The default value for *max* is 10.

next is the number of seconds to wait before attempting to dial the next number in a list of numbers (see the "set phone" command). The default is 3 seconds. Again, if the argument is the literal string 'random', the delay period is a random value between 1 and 30 seconds.

attempts is the maximum number of times to try to connect for each outgoing packet that triggers a dial. The previous value is unchanged if this parameter is omitted. If a value of zero is specified for *attempts*, **ppp** will keep trying until a connection is made.

So, for example:

```
set redial 10.3 4
```

will attempt to connect 4 times for each outgoing packet that causes a dial attempt with a 3 second delay between each number and a 10 second delay after all numbers have been tried. If multiple phone numbers are specified, the total number of attempts is still 4 (it does not attempt each number 4 times).

Alternatively,

```
set redial 10+10-5.3 20
```

tells **ppp** to attempt to connect 20 times. After the first attempt, **ppp** pauses for 10 seconds. After the next attempt it pauses for 20 seconds and so on until after the sixth attempt it pauses for 1 minute. The next 14 pauses will also have a duration of one minute. If **ppp** connects, disconnects and fails to connect again, the timeout starts again at 10 seconds.

Modifying the dial delay is very useful when running **ppp** in **-auto** mode on both ends of the link. If each end has the same timeout, both ends wind up calling each other at the same time if the link drops and both ends have packets queued. At some locations, the serial link may not be reliable, and carrier may be lost at inappropriate times. It is possible to have **ppp** redial should carrier be unexpectedly lost

during a session.

set reconnect timeout ntries

This command tells **ppp** to re-establish the connection *ntries* times on loss of carrier with a pause of *timeout* seconds before each try. For example,

set reconnect 3 5

tells **ppp** that on an unexpected loss of carrier, it should wait *3* seconds before attempting to reconnect. This may happen up to *5* times before **ppp** gives up. The default value of ntries is zero (no reconnect). Care should be taken with this option. If the local timeout is slightly longer than the remote timeout, the reconnect feature will always be triggered (up to the given number of times) after the remote side times out and hangs up. NOTE: In this context, losing too many LQRs constitutes a loss of carrier and will trigger a reconnect. If the **-background** flag is specified, all phone numbers are dialed at most once until a connection is made. The next number redial period specified with the "set redial" command is honoured, as is the reconnect tries value. If your redial value is less than the number of phone numbers specified, not all the specified numbers will be tried. To terminate the program, type

PPP ON awfulhak> close
ppp ON awfulhak> quit all

A simple "quit" command will terminate the pppctl(8) or telnet(1) connection but not the **ppp** program itself. You must use "quit all" to terminate **ppp** as well.

RECEIVING INCOMING PPP CONNECTIONS (Method 1)

To handle an incoming *PPP* connection request, follow these steps:

1. Make sure the modem and (optionally) */etc/rc.serial* is configured correctly.
   - Use Hardware Handshake (CTS/RTS) for flow control.
   - Modem should be set to NO echo back (ATE0) and NO results string (ATQ1).

2. Edit */etc/ttys* to enable a getty(8) on the port where the modem is attached. For example:

   ttyd1 "/usr/libexec/getty std.38400" dialup on secure

   Do not forget to send a HUP signal to the init(8) process to start the getty(8):

   # kill -HUP 1

It is usually also necessary to train your modem to the same DTR speed as the getty:

```
# ppp
ppp ON awfulhak> set device /dev/cuau1
ppp ON awfulhak> set speed 38400
ppp ON awfulhak> term
deflink: Entering terminal mode on /dev/cuau1
Type '~?' for help
at
OK
at
OK
atz
OK
at
OK
~.
ppp ON awfulhak> quit
```

3. Create a */usr/local/bin/ppplogin* file with the following contents:

```
#! /bin/sh
exec /usr/sbin/ppp -direct incoming
```

Direct mode (**-direct**) lets **ppp** work with stdin and stdout. You can also use pppctl(8) to connect to a configured diagnostic port, in the same manner as with client-side **ppp**.

Here, the *incoming* section must be set up in */etc/ppp/ppp.conf*.

Make sure that the *incoming* section contains the "allow users" command as appropriate.

4. Prepare an account for the incoming user.

ppp:xxxx:66:66:PPP Login User:/home/ppp:/usr/local/bin/ppplogin

Refer to the manual entries for adduser(8) and vipw(8) for details.

5. Support for IPCP Domain Name Server and NetBIOS Name Server negotiation can be enabled using the "accept dns" and "set nbns" commands. Refer to their descriptions below.

RECEIVING INCOMING PPP CONNECTIONS (Method 2)

This method differs in that we use **ppp** to authenticate the connection rather than login(1):

1. Configure your default section in */etc/gettytab* with automatic ppp recognition by specifying the "pp" capability:

   ```
   default:\
           :pp=/usr/local/bin/ppplogin:\
           .....
   ```

2. Configure your serial device(s), enable a getty(8) and create */usr/local/bin/ppplogin* as in the first three steps for method 1 above.

3. Add either "enable chap" or "enable pap" (or both) to */etc/ppp/ppp.conf* under the 'incoming' label (or whatever label *ppplogin* uses).

4. Create an entry in */etc/ppp/ppp.secret* for each incoming user:

   ```
   Pfred<TAB>xxxx
   Pgeorge<TAB>yyyy
   ```

Now, as soon as getty(8) detects a ppp connection (by recognising the HDLC frame headers), it runs "/usr/local/bin/ppplogin".

It is *VITAL* that either PAP or CHAP are enabled as above. If they are not, you are allowing anybody to establish a ppp session with your machine *without* a password, opening yourself up to all sorts of potential attacks.

AUTHENTICATING INCOMING CONNECTIONS

Normally, the receiver of a connection requires that the peer authenticates itself. This may be done using login(1), but alternatively, you can use PAP or CHAP. CHAP is the more secure of the two, but some clients may not support it. Once you decide which you wish to use, add the command 'enable chap' or 'enable pap' to the relevant section of *ppp.conf*.

You must then configure the */etc/ppp/ppp.secret* file. This file contains one line per possible client, each line containing up to five fields:

name key [*hisaddr* [*label* [*callback-number*]]]

The *name* and *key* specify the client username and password. If *key* is "*" and PAP is being used, **ppp**

will look up the password database (passwd(5)) when authenticating. If the client does not offer a suitable response based on any *name*/*key* combination in *ppp.secret*, authentication fails.

If authentication is successful, *hisaddr* (if specified) is used when negotiating IP numbers. See the "set ifaddr" command for details.

If authentication is successful and *label* is specified, the current system label is changed to match the given *label*. This will change the subsequent parsing of the *ppp.linkup* and *ppp.linkdown* files.

If authentication is successful and *callback-number* is specified and "set callback" has been used in *ppp.conf*, the client will be called back on the given number. If CBCP is being used, *callback-number* may also contain a list of numbers or a "*", as if passed to the "set cbcp" command. The value will be used in **ppp**'s subsequent CBCP phase.

PPP OVER TCP and UDP (a.k.a Tunnelling)

Instead of running **ppp** over a serial link, it is possible to use a TCP connection instead by specifying the host, port and protocol as the device:

```
set device ui-gate:6669/tcp
```

Instead of opening a serial device, **ppp** will open a TCP connection to the given machine on the given socket. It should be noted however that **ppp** does not use the telnet protocol and will be unable to negotiate with a telnet server. You should set up a port for receiving this *PPP* connection on the receiving machine (ui-gate). This is done by first updating */etc/services* to name the service:

```
ppp-in 6669/tcp # Incoming PPP connections over TCP
```

and updating */etc/inetd.conf* to tell inetd(8) how to deal with incoming connections on that port:

```
ppp-in stream tcp nowait root /usr/sbin/ppp ppp -direct ppp-in
```

Do not forget to send a HUP signal to inetd(8) after you have updated */etc/inetd.conf*. Here, we use a label named "ppp-in". The entry in */etc/ppp/ppp.conf* on ui-gate (the receiver) should contain the following:

```
ppp-in:
 set timeout 0
 set ifaddr 10.0.4.1 10.0.4.2
```

and the entry in */etc/ppp/ppp.linkup* should contain:

```
ppp-in:
 add 10.0.1.0/24 HISADDR
```

It is necessary to put the "add" command in *ppp.linkup* to ensure that the route is only added after **ppp** has negotiated and assigned addresses to its interface.

You may also want to enable PAP or CHAP for security. To enable PAP, add the following line:

```
enable PAP
```

You will also need to create the following entry in */etc/ppp/ppp.secret*:

```
MyAuthName MyAuthPasswd
```

If *MyAuthPasswd* is a "*", the password is looked up in the passwd(5) database.

The entry in */etc/ppp/ppp.conf* on awfulhak (the initiator) should contain the following:

```
ui-gate:
 set escape 0xff
 set device ui-gate:ppp-in/tcp
 set dial
 set timeout 30
 set log Phase Chat Connect hdlc LCP IPCP IPV6CP CCP tun
 set ifaddr 10.0.4.2 10.0.4.1
```

with the route setup in */etc/ppp/ppp.linkup*:

```
ui-gate:
 add 10.0.2.0/24 HISADDR
```

Again, if you are enabling PAP, you will also need this in the */etc/ppp/ppp.conf* profile:

```
 set authname MyAuthName
 set authkey MyAuthKey
```

We are assigning the address of 10.0.4.1 to ui-gate, and the address 10.0.4.2 to awfulhak. To open the connection, just type

```
awfulhak # ppp -background ui-gate
```

The result will be an additional "route" on awfulhak to the 10.0.2.0/24 network via the TCP connection, and an additional "route" on ui-gate to the 10.0.1.0/24 network. The networks are effectively bridged - the underlying TCP connection may be across a public network (such as the Internet), and the *PPP* traffic is conceptually encapsulated (although not packet by packet) inside the TCP stream between the two gateways.

The major disadvantage of this mechanism is that there are two "guaranteed delivery" mechanisms in place - the underlying TCP stream and whatever protocol is used over the *PPP* link - probably TCP again. If packets are lost, both levels will get in each others way trying to negotiate sending of the missing packet.

To avoid this overhead, it is also possible to do all this using UDP instead of TCP as the transport by simply changing the protocol from "tcp" to "udp". When using UDP as a transport, **ppp** will operate in synchronous mode. This is another gain as the incoming data does not have to be rearranged into packets.

Care should be taken when adding a default route through a tunneled setup like this. It is quite common for the default route (added in */etc/ppp/ppp.linkup*) to end up routing the link's TCP connection through the tunnel, effectively garrotting the connection. To avoid this, make sure you add a static route for the benefit of the link:

```
ui-gate:
 set escape 0xff
 set device ui-gate:ppp-in/tcp
 add ui-gate x.x.x.x
 .....
```

where "x.x.x.x" is the IP number that your route to "ui-gate" would normally use.

When routing your connection across a public network such as the Internet, it is preferable to encrypt the data. This can be done with the help of the MPPE protocol, although currently this means that you will not be able to also compress the traffic as MPPE is implemented as a compression layer (thank Microsoft for this). To enable MPPE encryption, add the following lines to */etc/ppp/ppp.conf* on the server:

```
 enable MSCHAPv2
 disable deflate pred1
 deny deflate pred1
```

ensuring that you have put the requisite entry in */etc/ppp/ppp.secret* (MSCHAPv2 is challenge based, so

passwd(5) cannot be used)

MSCHAPv2 and MPPE are accepted by default, so the client end should work without any additional changes (although ensure you have "set authname" and "set authkey" in your profile).

NETWORK ADDRESS TRANSLATION (PACKET ALIASING)

The **-nat** command line option enables network address translation (a.k.a. packet aliasing). This allows the **ppp** host to act as a masquerading gateway for other computers over a local area network. Outgoing IP packets are NAT'd so that they appear to come from the **ppp** host, and incoming packets are de-NAT'd so that they are routed to the correct machine on the local area network. NAT allows computers on private, unregistered subnets to have Internet access, although they are invisible from the outside world. In general, correct **ppp** operation should first be verified with network address translation disabled. Then, the **-nat** option should be switched on, and network applications (web browser, telnet(1), ftp(1), ping(8), traceroute(8)) should be checked on the **ppp** host. Finally, the same or similar applications should be checked on other computers in the LAN. If network applications work correctly on the **ppp** host, but not on other machines in the LAN, then the masquerading software is working properly, but the host is either not forwarding or possibly receiving IP packets. Check that IP forwarding is enabled in */etc/rc.conf* and that other machines have designated the **ppp** host as the gateway for the LAN. When starting **ppp** with the provided rc script, the default is to enable NAT; see *ppp_nat* in rc.conf(5) and */etc/defaults/rc.conf*.

PACKET FILTERING

This implementation supports packet filtering. There are four kinds of filters: the *in* filter, the *out* filter, the *dial* filter and the *alive* filter. Here are the basics:

- A filter definition has the following syntax:

  set filter *name rule-no action* [!] [[host] *src_addr*[/*width*] [*dst_addr*[/*width*]]] [*proto* [src *cmp port*] [dst *cmp port*] [estab] [syn] [finrst] [timeout *secs*]]

  1. *Name* should be one of 'in', 'out', 'dial' or 'alive'.

  2. *Rule-no* is a numeric value between '0' and '39' specifying the rule number. Rules are specified in numeric order according to *rule-no*, but only if rule '0' is defined.

  3. *Action* may be specified as 'permit' or 'deny', in which case, if a given packet matches the rule, the associated action is taken immediately. *Action* can also be specified as 'clear' to clear the action associated with that particular rule, or as a new rule number greater than the current rule. In this case, if a given packet matches the current rule, the packet will next be matched against the new rule number (rather than the next rule number).

The *action* may optionally be followed with an exclamation mark ("!"), telling **ppp** to reverse the sense of the following match.

4. [*src_addr*[/*width*]] and [*dst_addr*[/*width*]] are the source and destination IP number specifications. If [/*width*] is specified, it gives the number of relevant netmask bits, allowing the specification of an address range.

   Either *src_addr* or *dst_addr* may be given the values MYADDR, HISADDR, MYADDR6 or HISADDR6 (refer to the description of the "bg" command for a description of these values). When these values are used, the filters will be updated any time the values change. This is similar to the behaviour of the "add" command below.

5. *Proto* may be any protocol from protocols(5).

6. *Cmp* is one of 'lt', 'eq' or 'gt', meaning less-than, equal and greater-than respectively. *Port* can be specified as a numeric port or by service name from */etc/services*.

7. The 'estab', 'syn', and 'finrst' flags are only allowed when *proto* is set to 'tcp', and represent the TH_ACK, TH_SYN and TH_FIN or TH_RST TCP flags respectively.

8. The timeout value adjusts the current idle timeout to at least *secs* seconds. If a timeout is given in the alive filter as well as in the in/out filter, the in/out value is used. If no timeout is given, the default timeout (set using **set timeout** and defaulting to 180 seconds) is used.

- Each filter can hold up to 40 rules, starting from rule 0. The entire rule set is not effective until rule 0 is defined, i.e., the default is to allow everything through.

- If no rule in a defined set of rules matches a packet, that packet will be discarded (blocked). If there are no rules in a given filter, the packet will be permitted.

- It is possible to filter based on the payload of UDP frames where those frames contain a *PROTO_IP PPP* frame header. See the *filter-decapsulation* option below for further details.

- Use "set filter *name* -1" to flush all rules.

See */usr/share/examples/ppp/ppp.conf.sample*.

SETTING THE IDLE TIMER

To check/set the idle timer, use the "show bundle" and "set timeout" commands:

```
ppp ON awfulhak> set timeout 600
```

The timeout period is measured in seconds, the default value for which is 180 seconds (or 3 min). To disable the idle timer function, use the command

```
ppp ON awfulhak> set timeout 0
```

In **-ddial** and **-dedicated** modes, the idle timeout is ignored. In **-auto** mode, when the idle timeout causes the *PPP* session to be closed, the **ppp** program itself remains running. Another trigger packet will cause it to attempt to re-establish the link.

PREDICTOR-1 and DEFLATE COMPRESSION

ppp supports both Predictor type 1 and deflate compression. By default, **ppp** will attempt to use (or be willing to accept) both compression protocols when the peer agrees (or requests them). The deflate protocol is preferred by **ppp**. Refer to the "disable" and "deny" commands if you wish to disable this functionality.

It is possible to use a different compression algorithm in each direction by using only one of "disable deflate" and "deny deflate" (assuming that the peer supports both algorithms).

By default, when negotiating DEFLATE, **ppp** will use a window size of 15. Refer to the "set deflate" command if you wish to change this behaviour.

A special algorithm called DEFLATE24 is also available, and is disabled and denied by default. This is exactly the same as DEFLATE except that it uses CCP ID 24 to negotiate. This allows **ppp** to successfully negotiate DEFLATE with **pppd** version 2.3.*.

CONTROLLING IP ADDRESS

For IPv4, **ppp** uses IPCP to negotiate IP addresses. Each side of the connection specifies the IP address that it is willing to use, and if the requested IP address is acceptable then **ppp** returns an ACK to the requester. Otherwise, **ppp** returns NAK to suggest that the peer use a different IP address. When both sides of the connection agree to accept the received request (and send an ACK), IPCP is set to the open state and a network level connection is established. To control this IPCP behaviour, this implementation has the "set ifaddr" command for defining the local and remote IP address:

```
set ifaddr [src_addr[/nn] [dst_addr[/nn] [netmask [trigger_addr]]]]
```

where, 'src_addr' is the IP address that the local side is willing to use, 'dst_addr' is the IP address which the remote side should use and 'netmask' is the netmask that should be used. 'Src_addr' defaults to the current hostname(1), 'dst_addr' defaults to 0.0.0.0, and 'netmask' defaults to whatever mask is

appropriate for 'src_addr'. It is only possible to make 'netmask' smaller than the default. The usual value is 255.255.255.255, as most kernels ignore the netmask of a POINTOPOINT interface.

Some incorrect *PPP* implementations require that the peer negotiates a specific IP address instead of 'src_addr'. If this is the case, 'trigger_addr' may be used to specify this IP number. This will not affect the routing table unless the other side agrees with this proposed number.

```
set ifaddr 192.244.177.38 192.244.177.2 255.255.255.255 0.0.0.0
```

The above specification means:

- I will first suggest that my IP address should be 0.0.0.0, but I will only accept an address of 192.244.177.38.
- I strongly insist that the peer uses 192.244.177.2 as his own address and will not permit the use of any IP address but 192.244.177.2. When the peer requests another IP address, I will always suggest that it uses 192.244.177.2.
- The routing table entry will have a netmask of 0xffffffff.

This is all fine when each side has a pre-determined IP address, however it is often the case that one side is acting as a server which controls all IP addresses and the other side should go along with it. In order to allow more flexible behaviour, the "set ifaddr" command allows the user to specify IP addresses more loosely:

```
set ifaddr 192.244.177.38/24 192.244.177.2/20
```

A number followed by a slash ("/") represents the number of bits significant in the IP address. The above example means:

- I would like to use 192.244.177.38 as my address if it is possible, but I will also accept any IP address between 192.244.177.0 and 192.244.177.255.
- I would like to make him use 192.244.177.2 as his own address, but I will also permit him to use any IP address between 192.244.176.0 and 192.244.191.255.
- As you may have already noticed, 192.244.177.2 is equivalent to saying 192.244.177.2/32.
- As an exception, 0 is equivalent to 0.0.0.0/0, meaning that I have no preferred IP address and will obey the remote peers selection. When using zero, no routing table entries will be made until a connection is established.
- 192.244.177.2/0 means that I will accept/permit any IP address but I will suggest that 192.244.177.2 be used first.

When negotiating IPv6 addresses, no control is given to the user. IPV6CP negotiation is fully

automatic.

CONNECTING WITH YOUR INTERNET SERVICE PROVIDER

The following steps should be taken when connecting to your ISP:

1. Describe your providers phone number(s) in the dial script using the "set phone" command. This command allows you to set multiple phone numbers for dialing and redialing separated by either a pipe ("|") or a colon (":"):

   set phone *telno*[|*backupnumber*]...[:*nextnumber*]...

   Numbers after the first in a pipe-separated list are only used if the previous number was used in a failed dial or login script. Numbers separated by a colon are used sequentially, irrespective of what happened as a result of using the previous number. For example:

   set phone "1234567|2345678:3456789|4567890"

   Here, the 1234567 number is attempted. If the dial or login script fails, the 2345678 number is used next time, but *only* if the dial or login script fails. On the dial after this, the 3456789 number is used. The 4567890 number is only used if the dial or login script using the 3456789 fails. If the login script of the 2345678 number fails, the next number is still the 3456789 number. As many pipes and colons can be used as are necessary (although a given site would usually prefer to use either the pipe or the colon, but not both). The next number redial timeout is used between all numbers. When the end of the list is reached, the normal redial period is used before starting at the beginning again. The selected phone number is substituted for the \\T string in the "set dial" command (see below).

2. Set up your redial requirements using "set redial". For example, if you have a bad telephone line or your provider is usually engaged (not so common these days), you may want to specify the following:

   set redial 10 4

   This says that up to 4 phone calls should be attempted with a pause of 10 seconds before dialing the first number again.

3. Describe your login procedure using the "set dial" and "set login" commands. The "set dial" command is used to talk to your modem and establish a link with your ISP, for example:

   set dial "ABORT BUSY ABORT NO\\sCARRIER TIMEOUT 4 \"\" \

```
ATZ OK-ATZ-OK ATDT\\T TIMEOUT 60 CONNECT"
```

This modem "chat" string means:

- Abort if the string "BUSY" or "NO CARRIER" are received.
- Set the timeout to 4 seconds.
- Expect nothing.
- Send ATZ.
- Expect OK. If that is not received within the 4 second timeout, send ATZ and expect OK.
- Send ATDTxxxxxxx where xxxxxxx is the next number in the phone list from above.
- Set the timeout to 60.
- Wait for the CONNECT string.

Once the connection is established, the login script is executed. This script is written in the same style as the dial script, but care should be taken to avoid having your password logged:

```
set authkey MySecret
set login "TIMEOUT 15 login:-\\r-login: awfulhak \
  word: \\P ocol: PPP HELLO"
```

This login "chat" string means:

- Set the timeout to 15 seconds.
- Expect "login:". If it is not received, send a carriage return and expect "login:" again.
- Send "awfulhak"
- Expect "word:" (the tail end of a "Password:" prompt).
- Send whatever our current *authkey* value is set to.
- Expect "ocol:" (the tail end of a "Protocol:" prompt).

- Send "PPP".
- Expect "HELLO".

The "set authkey" command is logged specially. When *command* or *chat* logging is enabled, the actual password is not logged; '********' is logged instead.

Login scripts vary greatly between ISPs. If you are setting one up for the first time, *ENABLE CHAT LOGGING* so that you can see if your script is behaving as you expect.

4. Use "set device" and "set speed" to specify your serial line and speed, for example:

```
set device /dev/cuau0
set speed 115200
```

Cuad0 is the first serial port on FreeBSD. If you are running **ppp** on OpenBSD, cua00 is the first. A speed of 115200 should be specified if you have a modem capable of bit rates of 28800 or more. In general, the serial speed should be about four times the modem speed.

5. Use the "set ifaddr" command to {define} the IP address.

- If you know what IP address your provider uses, then use it as the remote address (dst_addr), otherwise choose something like 10.0.0.2/0 (see below).
- If your provider has assigned a particular IP address to you, then use it as your address (src_addr).
- If your provider assigns your address dynamically, choose a suitably unobtrusive and unspecific IP number as your address. 10.0.0.1/0 would be appropriate. The bit after the / specifies how many bits of the address you consider to be important, so if you wanted to insist on something in the class C network 1.2.3.0, you could specify 1.2.3.1/24.
- If you find that your ISP accepts the first IP number that you suggest, specify third and forth arguments of "0.0.0.0". This will force your ISP to assign a number. (The third argument will be ignored as it is less restrictive than the default mask for your 'src_addr').

An example for a connection where you do not know your IP number or your ISPs IP number would be:

```
set ifaddr 10.0.0.1/0 10.0.0.2/0 0.0.0.0 0.0.0.0
```

6. In most cases, your ISP will also be your default router. If this is the case, add the line

   add default HISADDR

   to */etc/ppp/ppp.conf* (or to */etc/ppp/ppp.linkup* for setups that do not use **-auto** mode).

   This tells **ppp** to add a default route to whatever the peer address is (10.0.0.2 in this example). This route is 'sticky', meaning that should the value of HISADDR change, the route will be updated accordingly.

7. If your provider requests that you use PAP/CHAP authentication methods, add the next lines to your */etc/ppp/ppp.conf* file:

   set authname MyName
   set authkey MyPassword

   Both are accepted by default, so **ppp** will provide whatever your ISP requires.

   It should be noted that a login script is rarely (if ever) required when PAP or CHAP are in use.

8. Ask your ISP to authenticate your nameserver address(es) with the line

   enable dns

   Do *NOT* do this if you are running a local DNS unless you also either use "resolv readonly" or have "resolv restore" in */etc/ppp/ppp.linkdown*, as **ppp** will simply circumvent its use by entering some nameserver lines in */etc/resolv.conf*.

Please refer to */usr/share/examples/ppp/ppp.conf.sample* and */usr/share/examples/ppp/ppp.linkup.sample* for some real examples. The pmdemand label should be appropriate for most ISPs.

LOGGING FACILITY

ppp is able to generate the following log info either via syslog(3) or directly to the screen:

All	Enable all logging facilities. This generates a lot of log. The most common use of 'all' is as a basis, where you remove some facilities after enabling 'all' ('debug' and 'timer' are usually best disabled.)
Async	Dump async level packet in hex.
CBCP	Generate CBCP (CallBack Control Protocol) logs.
CCP	Generate a CCP packet trace.

Chat	Generate 'dial', 'login', 'logout' and 'hangup' chat script trace logs.
Command	Log commands executed either from the command line or any of the configuration files.
Connect	Log Chat lines containing the string "CONNECT".
Debug	Log debug information.
DNS	Log DNS QUERY packets.
Filter	Log packets permitted by the dial filter and denied by any filter.
HDLC	Dump HDLC packet in hex.
ID0	Log all function calls specifically made as user id 0.
IPCP	Generate an IPCP packet trace.
LCP	Generate an LCP packet trace.
LQM	Generate LQR reports.
Phase	Phase transition log output.
Physical	Dump physical level packet in hex.
Radius	Dump RADIUS information. RADIUS information resulting from the link coming up or down is logged at "Phase" level unless "Radius" logging is enabled. This log level is most useful for monitoring RADIUS alive information.
Sync	Dump sync level packet in hex.
TCP/IP	Dump all TCP/IP packets.
Timer	Log timer manipulation.
TUN	Include the tun device on each log line.
Warning	Output to the terminal device. If there is currently no terminal, output is sent to the log file using syslogs LOG_WARNING.
Error	Output to both the terminal device and the log file using syslogs LOG_ERROR.
Alert	Output to the log file using LOG_ALERT.

The "set log" command allows you to set the logging output level. Multiple levels can be specified on a single command line. The default is equivalent to "set log Phase".

It is also possible to log directly to the screen. The syntax is the same except that the word "local" should immediately follow "set log". The default is "set log local" (i.e., only the un-maskable warning, error and alert output).

If The first argument to "set log [local]" begins with a '+' or a '-' character, the current log levels are not cleared, for example:

```
PPP ON awfulhak> set log phase
PPP ON awfulhak> show log
Log:   Phase Warning Error Alert
Local: Warning Error Alert
```

```
PPP ON awfulhak> set log +tcp/ip -warning
PPP ON awfulhak> set log local +command
PPP ON awfulhak> show log
Log:   Phase TCP/IP Warning Error Alert
Local: Command Warning Error Alert
```

Log messages of level Warning, Error and Alert are not controllable using "set log [local]".

The *Warning* level is special in that it will not be logged if it can be displayed locally.

SIGNAL HANDLING

ppp deals with the following signals:

INT Receipt of this signal causes the termination of the current connection (if any). This will cause **ppp** to exit unless it is in **-auto** or **-ddial** mode.

HUP, TERM & QUIT
These signals tell **ppp** to exit.

USR1 This signal, tells **ppp** to re-open any existing server socket, dropping all existing diagnostic connections. Sockets that could not previously be opened will be retried.

USR2 This signal, tells **ppp** to close any existing server socket, dropping all existing diagnostic connections. SIGUSR1 can still be used to re-open the socket.

MULTI-LINK PPP

If you wish to use more than one physical link to connect to a *PPP* peer, that peer must also understand the *MULTI-LINK PPP* protocol. Refer to RFC 1990 for specification details.

The peer is identified using a combination of his "endpoint discriminator" and his "authentication id". Either or both of these may be specified. It is recommended that at least one is specified, otherwise there is no way of ensuring that all links are actually connected to the same peer program, and some confusing lock-ups may result. Locally, these identification variables are specified using the "set enddisc" and "set authname" commands. The 'authname' (and 'authkey') must be agreed in advance with the peer.

Multi-link capabilities are enabled using the "set mrru" command (set maximum reconstructed receive unit). Once multi-link is enabled, **ppp** will attempt to negotiate a multi-link connection with the peer.

By default, only one 'link' is available (called 'deflink'). To create more links, the "clone" command is

used. This command will clone existing links, where all characteristics are the same except:

1. The new link has its own name as specified on the "clone" command line.

2. The new link is an 'interactive' link. Its mode may subsequently be changed using the "set mode" command.

3. The new link is in a 'closed' state.

A summary of all available links can be seen using the "show links" command.

Once a new link has been created, command usage varies. All link specific commands must be prefixed with the "link *name*" command, specifying on which link the command is to be applied. When only a single link is available, **ppp** is smart enough not to require the "link *name*" prefix.

Some commands can still be used without specifying a link - resulting in an operation at the 'bundle' level. For example, once two or more links are available, the command "show ccp" will show CCP configuration and statistics at the multi-link level, and "link deflink show ccp" will show the same information at the "deflink" link level.

Armed with this information, the following configuration might be used:

```
mp:
 set timeout 0
 set log phase chat
 set device /dev/cuau0 /dev/cuau1 /dev/cuau2
 set phone "123456789"
 set dial "ABORT BUSY ABORT NO\sCARRIER TIMEOUT 5 \"\" ATZ \
        OK-AT-OK \\dATDT\\T TIMEOUT 45 CONNECT"
 set login
 set ifaddr 10.0.0.1/0 10.0.0.2/0 0.0.0.0 0.0.0.0
 set authname ppp
 set authkey ppppassword

 set mrru 1500
 clone 1,2,3                # Create 3 new links - duplicates of the default
 link deflink remove        # Delete the default link (called ''deflink'')
```

Note how all cloning is done at the end of the configuration. Usually, the link will be configured first, then cloned. If you wish all links to be up all the time, you can add the following line to the end of your

configuration.

```
link 1,2,3 set mode ddial
```

If you want the links to dial on demand, this command could be used:

```
link * set mode auto
```

Links may be tied to specific names by removing the "set device" line above, and specifying the following after the "clone" command:

```
link 1 set device /dev/cuau0
link 2 set device /dev/cuau1
link 3 set device /dev/cuau2
```

Use the "help" command to see which commands require context (using the "link" command), which have optional context and which should not have any context.

When **ppp** has negotiated *MULTI-LINK* mode with the peer, it creates a local domain socket in the */var/run* directory. This socket is used to pass link information (including the actual link file descriptor) between different **ppp** invocations. This facilitates **ppp**'s ability to be run from a getty(8) or directly from */etc/gettydefs* (using the 'pp=' capability), without needing to have initial control of the serial line. Once **ppp** negotiates multi-link mode, it will pass its open link to any already running process. If there is no already running process, **ppp** will act as the master, creating the socket and listening for new connections.

PPP COMMAND LIST

This section lists the available commands and their effect. They are usable either from an interactive **ppp** session, from a configuration file or from a pppctl(8) or telnet(1) session.

accept|deny|enable|disable *option*....

These directives tell **ppp** how to negotiate the initial connection with the peer. Each "option" has a default of either accept or deny and enable or disable. "Accept" means that the option will be ACK'd if the peer asks for it. "Deny" means that the option will be NAK'd if the peer asks for it. "Enable" means that the option will be requested by us. "Disable" means that the option will not be requested by us.

"Option" may be one of the following:

acfcomp

Default: Enabled and Accepted. ACFComp stands for Address and Control Field Compression. Non LCP packets will usually have an address field of 0xff (the All-Stations address) and a control field of 0x03 (the Unnumbered Information command). If this option is negotiated, these two bytes are simply not sent, thus minimising traffic.

See *rfc1662* for details.

chap[05]

Default: Disabled and Accepted. CHAP stands for Challenge Handshake Authentication Protocol. Only one of CHAP and PAP (below) may be negotiated. With CHAP, the authenticator sends a "challenge" message to its peer. The peer uses a one-way hash function to encrypt the challenge and sends the result back. The authenticator does the same, and compares the results. The advantage of this mechanism is that no passwords are sent across the connection. A challenge is made when the connection is first made. Subsequent challenges may occur. If you want to have your peer authenticate itself, you must "enable chap". in */etc/ppp/ppp.conf*, and have an entry in */etc/ppp/ppp.secret* for the peer.

When using CHAP as the client, you need only specify "AuthName" and "AuthKey" in */etc/ppp/ppp.conf*. CHAP is accepted by default. Some *PPP* implementations use "MS-CHAP" rather than MD5 when encrypting the challenge. MS-CHAP is a combination of MD4 and DES. If **ppp** was built on a machine with DES libraries available, it will respond to MS-CHAP authentication requests, but will never request them.

deflate

Default: Enabled and Accepted. This option decides if deflate compression will be used by the Compression Control Protocol (CCP). This is the same algorithm as used by the gzip(1) program. Note: There is a problem negotiating *deflate* capabilities with **pppd** - a *PPP* implementation available under many operating systems. **pppd** (version 2.3.1) incorrectly attempts to negotiate *deflate* compression using type *24* as the CCP configuration type rather than type *26* as specified in *rfc1979*. Type *24* is actually specified as "PPP Magna-link Variable Resource Compression" in *rfc1975*! **ppp** is capable of negotiating with **pppd**, but only if "deflate24" is *enable*d and *accept*ed.

deflate24

Default: Disabled and Denied. This is a variance of the *deflate* option, allowing negotiation with the **pppd** program. Refer to the *deflate* section above for details. It is disabled by default as it violates *rfc1975*.

dns

Default: Disabled and Denied. This option allows DNS negotiation.

If "enabled," **ppp** will request that the peer confirms the entries in */etc/resolv.conf*. If the peer NAKs our request (suggesting new IP numbers), */etc/resolv.conf* is updated and another request is sent to confirm the new entries.

If "accepted," **ppp** will answer any DNS queries requested by the peer rather than rejecting them. The answer is taken from */etc/resolv.conf* unless the "set dns" command is used as an override.

enddisc

Default: Enabled and Accepted. This option allows control over whether we negotiate an endpoint discriminator. We only send our discriminator if "set enddisc" is used and *enddisc* is enabled. We reject the peers discriminator if *enddisc* is denied.

LANMan|chap80lm

Default: Disabled and Accepted. The use of this authentication protocol is discouraged as it partially violates the authentication protocol by implementing two different mechanisms (LANMan & NT) under the guise of a single CHAP type (0x80). "LANMan" uses a simple DES encryption mechanism and is the least secure of the CHAP alternatives (although is still more secure than PAP).

Refer to the "MSChap" description below for more details.

lqr

Default: Disabled and Accepted. This option decides if Link Quality Requests will be sent or accepted. LQR is a protocol that allows **ppp** to determine that the link is down without relying on the modems carrier detect. When LQR is enabled, **ppp** sends the *QUALPROTO* option (see "set lqrperiod" below) as part of the LCP request. If the peer agrees, both sides will exchange LQR packets at the agreed frequency, allowing detailed link quality monitoring by enabling LQM logging. If the peer does not agree, and if the "echo" option is enabled, **ppp** will send *LCP ECHO* requests instead. These packets pass no information of interest, but they *MUST* be replied to by the peer.

Whether using *LQR* or *LCP ECHO*, **ppp** will abruptly drop the connection if 5 unacknowledged packets have been sent rather than sending a 6th. A message is logged at the *PHASE* level, and any appropriate "reconnect" values are honoured as if the peer were responsible for dropping the connection.

Refer to the "enable echo" command description for differences in behaviour prior to **ppp** version 3.4.2.

mppe

Default: Enabled and Accepted. This is Microsoft Point to Point Encryption scheme. MPPE key size can be 40-, 56- and 128-bits. Refer to "set mppe" command.

MSChapV2|chap81

Default: Disabled and Accepted. It is very similar to standard CHAP (type 0x05) except that it issues challenges of a fixed 16 bytes in length and uses a combination of MD4, SHA-1 and DES to encrypt the challenge rather than using the standard MD5 mechanism.

MSChap|chap80nt

Default: Disabled and Accepted. The use of this authentication protocol is discouraged as it partially violates the authentication protocol by implementing two different mechanisms (LANMan & NT) under the guise of a single CHAP type (0x80). It is very similar to standard CHAP (type 0x05) except that it issues challenges of a fixed 8 bytes in length and uses a combination of MD4 and DES to encrypt the challenge rather than using the standard MD5 mechanism. CHAP type 0x80 for LANMan is also supported - see "enable LANMan" for details.

Because both "LANMan" and "NT" use CHAP type 0x80, when acting as authenticator with both "enabled", **ppp** will rechallenge the peer up to three times if it responds using the wrong one of the two protocols. This gives the peer a chance to attempt using both protocols.

Conversely, when **ppp** acts as the authenticatee with both protocols "accepted", the protocols are used alternately in response to challenges.

Note: If only LANMan is enabled, **pppd** (version 2.3.5) misbehaves when acting as authenticatee. It provides both the NT and the LANMan answers, but also suggests that only the NT answer should be used.

pap

Default: Disabled and Accepted. PAP stands for Password Authentication Protocol. Only one of PAP and CHAP (above) may be negotiated. With PAP, the ID and Password are sent repeatedly to the peer until authentication is acknowledged or the connection is terminated. This is a rather poor security mechanism. It is only performed when the connection is first established. If you want to have your peer authenticate itself, you must "enable pap". in */etc/ppp/ppp.conf*, and have an entry in */etc/ppp/ppp.secret* for the peer (although see the "passwdauth" and "set radius" options below).

When using PAP as the client, you need only specify "AuthName" and "AuthKey" in */etc/ppp/ppp.conf*. PAP is accepted by default.

pred1
Default: Enabled and Accepted. This option decides if Predictor 1 compression will be used by the Compression Control Protocol (CCP).

protocomp
Default: Enabled and Accepted. This option is used to negotiate PFC (Protocol Field Compression), a mechanism where the protocol field number is reduced to one octet rather than two.

shortseq
Default: Enabled and Accepted. This option determines if **ppp** will request and accept requests for short (12 bit) sequence numbers when negotiating multi-link mode. This is only applicable if our MRRU is set (thus enabling multi-link).

vjcomp
Default: Enabled and Accepted. This option determines if Van Jacobson header compression will be used.

The following options are not actually negotiated with the peer. Therefore, accepting or denying them makes no sense.

echo
Default: Disabled. When this option is enabled, **ppp** will send *LCP ECHO* requests to the peer at the frequency defined by "echoperiod". Note, *LQR* requests will supersede *LCP ECHO* requests if enabled and negotiated. See "set lqrperiod" below for details.

Prior to **ppp** version 3.4.2, "echo" was considered enabled if lqr was enabled and negotiated, otherwise it was considered disabled. For the same behaviour, it is now necessary to "enable lqr echo" rather than just "enable lqr".

filter-decapsulation
Default: Disabled. When this option is enabled, **ppp** will examine UDP frames to see if they actually contain a *PPP* frame as their payload. If this is the case, all filters will operate on the payload rather than the actual packet.

This is useful if you want to send PPPoUDP traffic over a *PPP* link, but want that link to do smart things with the real data rather than the UDP wrapper.

The UDP frame payload must not be compressed in any way, otherwise **ppp** will not be able to interpret it. It is therefore recommended that you **disable vj pred1 deflate** and **deny vj pred1**

deflate in the configuration for the **ppp** invocation with the udp link.

force-scripts
Default: Disabled. Forces execution of the configured chat scripts in direct and dedicated modes.

idcheck
Default: Enabled. When **ppp** exchanges low-level LCP, CCP and IPCP configuration traffic, the *Identifier* field of any replies is expected to be the same as that of the request. By default, **ppp** drops any reply packets that do not contain the expected identifier field, reporting the fact at the respective log level. If *idcheck* is disabled, **ppp** will ignore the identifier field.

iface-alias
Default: Enabled if **-nat** is specified. This option simply tells **ppp** to add new interface addresses to the interface rather than replacing them. The option can only be enabled if network address translation is enabled ("nat enable yes").

With this option enabled, **ppp** will pass traffic for old interface addresses through the NAT engine (see libalias(3)), resulting in the ability (in **-auto** mode) to properly connect the process that caused the PPP link to come up in the first place.

Disabling NAT with "nat enable no" will also disable 'iface-alias'.

ipcp
Default: Enabled. This option allows **ppp** to attempt to negotiate IP control protocol capabilities and if successful to exchange IP datagrams with the peer.

ipv6cp
Default: Enabled. This option allows **ppp** to attempt to negotiate IPv6 control protocol capabilities and if successful to exchange IPv6 datagrams with the peer.

keep-session
Default: Disabled. When **ppp** runs as a Multi-link server, a different **ppp** instance initially receives each connection. After determining that the link belongs to an already existing bundle (controlled by another **ppp** invocation), **ppp** will transfer the link to that process.

If the link is a tty device or if this option is enabled, **ppp** will not exit, but will change its process name to "session owner" and wait for the controlling **ppp** to finish with the link and deliver a signal back to the idle process. This prevents the confusion that results from **ppp**'s parent considering the link resource available again.

For tty devices that have entries in */etc/ttys*, this is necessary to prevent another getty(8) from being started, and for program links such as sshd(8), it prevents sshd(8) from exiting due to the death of its child. As **ppp** cannot determine its parents requirements (except for the tty case), this option must be enabled manually depending on the circumstances.

loopback

Default: Enabled. When *loopback* is enabled, **ppp** will automatically loop back packets being sent out with a destination address equal to that of the *PPP* interface. If disabled, **ppp** will send the packet, probably resulting in an ICMP redirect from the other end. It is convenient to have this option enabled when the interface is also the default route as it avoids the necessity of a loopback route.

NAS-IP-Address

Default: Enabled. This option controls whether **ppp** sends the "NAS-IP-Address" attribute to the RADIUS server when RADIUS is in use (see "set radius").

Note, at least one of "NAS-IP-Address" and "NAS-Identifier" must be enabled.

Versions of **ppp** prior to version 3.4.1 did not send the "NAS-IP-Address" attribute as it was reported to break the Radiator RADIUS server. As the latest rfc (2865) no longer hints that only one of "NAS-IP-Address" and "NAS-Identifier" should be sent (as rfc 2138 did), **ppp** now sends both and leaves it up to the administrator that chooses to use bad RADIUS implementations to "disable NAS-IP-Address".

NAS-Identifier

Default: Enabled. This option controls whether **ppp** sends the "NAS-Identifier" attribute to the RADIUS server when RADIUS is in use (see "set radius").

Note, at least one of "NAS-IP-Address" and "NAS-Identifier" must be enabled.

passwdauth

Default: Disabled. Enabling this option will tell the PAP authentication code to use the password database (see passwd(5)) to authenticate the caller if they cannot be found in the */etc/ppp/ppp.secret* file. */etc/ppp/ppp.secret* is always checked first. If you wish to use passwords from passwd(5), but also to specify an IP number or label for a given client, use "*" as the client password in */etc/ppp/ppp.secret*.

proxy

Default: Disabled. Enabling this option will tell **ppp** to proxy ARP for the peer. This means that **ppp** will make an entry in the ARP table using HISADDR and the MAC address of the local

network in which HISADDR appears. This allows other machines connecteed to the LAN to talk to the peer as if the peer itself was connected to the LAN. The proxy entry cannot be made unless HISADDR is an address from a LAN.

proxyall
Default: Disabled. Enabling this will tell **ppp** to add proxy arp entries for every IP address in all class C or smaller subnets routed via the tun interface.

Proxy arp entries are only made for sticky routes that are added using the "add" command. No proxy arp entries are made for the interface address itself (as created by the "set ifaddr" command).

sroutes
Default: Enabled. When the "add" command is used with the HISADDR, MYADDR, HISADDR6 or MYADDR6 values, entries are stored in the 'sticky route' list. Each time these variables change, this list is re-applied to the routing table.

Disabling this option will prevent the re-application of sticky routes, although the 'stick route' list will still be maintained.

[tcp]mssfixup
Default: Enabled. This option tells **ppp** to adjust TCP SYN packets so that the maximum receive segment size is not greater than the amount allowed by the interface MTU.

throughput
Default: Enabled. This option tells **ppp** to gather throughput statistics. Input and output is sampled over a rolling 5 second window, and current, best and total figures are retained. This data is output when the relevant *PPP* layer shuts down, and is also available using the "show" command. Throughput statistics are available at the "IPCP" and "physical" levels.

utmp
Default: Enabled. Normally, when a user is authenticated using PAP or CHAP, and when **ppp** is running in **-direct** mode, an entry is made in the utmp and wtmp files for that user. Disabling this option will tell **ppp** not to make any utmp or wtmp entries. This is usually only necessary if you require the user to both login and authenticate themselves.

add[!] *dest*[/*nn*] [*mask*] [*gateway*]
Dest is the destination IP address. The netmask is specified either as a number of bits with */nn* or as an IP number using *mask*. *0 0* or simply *0* with no mask refers to the default route. It is also possible to use the literal name 'default' instead of *0*. *Gateway* is the next hop gateway to get to the

given *dest* machine/network. Refer to the route(8) command for further details.

It is possible to use the symbolic names 'MYADDR', 'HISADDR', 'MYADDR6' or 'HISADDR6' as the destination, and 'HISADDR' or 'HISADDR6' as the *gateway*. 'MYADDR' is replaced with the interface IP address, 'HISADDR' is replaced with the interface IP destination (peer) address, 'MYADDR6' is replaced with the interface IPv6 address, and 'HISADDR6' is replaced with the interface IPv6 destination address,

If the *add!* command is used (note the trailing "!"), then if the route already exists, it will be updated as with the 'route change' command (see route(8) for further details).

Routes that contain the "HISADDR", "MYADDR", "HISADDR6", "MYADDR6", "DNS0", or "DNS1" constants are considered 'sticky'. They are stored in a list (use "show ncp" to see the list), and each time the value of one of these variables changes, the appropriate routing table entries are updated. This facility may be disabled using "disable sroutes".

allow *command* [*args*]

This command controls access to **ppp** and its configuration files. It is possible to allow user-level access, depending on the configuration file label and on the mode that **ppp** is being run in. For example, you may wish to configure **ppp** so that only user 'fred' may access label 'fredlabel' in **-background** mode.

User id 0 is immune to these commands.

allow user[s] *logname...*

By default, only user id 0 is allowed access to **ppp**. If this command is used, all of the listed users are allowed access to the section in which the "allow users" command is found. The 'default' section is always checked first (even though it is only ever automatically loaded at startup). "allow users" commands are cumulative in a given section, but users allowed in any given section override users allowed in the default section, so it is possible to allow users access to everything except a given label by specifying default users in the 'default' section, and then specifying a new user list for that label.

If user '*' is specified, access is allowed to all users.

allow mode[s] *mode...*

By default, access using any **ppp** mode is possible. If this command is used, it restricts the access *modes* allowed to load the label under which this command is specified. Again, as with the "allow users" command, each "allow modes" command overrides any previous settings, and the 'default' section is always checked first.

Possible modes are: 'interactive', 'auto', 'direct', 'dedicated', 'ddial', 'background' and '*'.

When running in multi-link mode, a section can be loaded if it allows *any* of the currently existing line modes.

nat *command* [*args*]

This command allows the control of the network address translation (also known as masquerading or IP aliasing) facilities that are built into **ppp**. NAT is done on the external interface only, and is unlikely to make sense if used with the **-direct** flag.

If nat is enabled on your system (it may be omitted at compile time), the following commands are possible:

nat enable yes|no

This command either switches network address translation on or turns it off. The **-nat** command line flag is synonymous with "nat enable yes".

nat addr [*addr_local addr_alias*]

This command allows data for *addr_alias* to be redirected to *addr_local*. It is useful if you own a small number of real IP numbers that you wish to map to specific machines behind your gateway.

nat deny_incoming yes|no

If set to yes, this command will refuse all incoming packets where an aliasing link does not already exist. Refer to the *CONCEPTUAL BACKGROUND* section of libalias(3) for a description of what an "aliasing link" is.

It should be noted under what circumstances an aliasing link is created by libalias(3). It may be necessary to further protect your network from outside connections using the "set filter" or "nat target" commands.

nat help|?

This command gives a summary of available nat commands.

nat log yes|no

This option causes various NAT statistics and information to be logged to the file */var/log/alias.log*.

nat port *proto targetIP*:*targetPort*[-*targetPort*] *aliasPort*[-*aliasPort*] [*remoteIP*:*remotePort*[-*remotePort*]]

This command causes incoming *proto* connections to *aliasPort* to be redirected to *targetPort* on *targetIP*. *proto* is either "tcp" or "udp".

A range of port numbers may be specified as shown above. The ranges must be of the same size.

If *remoteIP* is specified, only data coming from that IP number is redirected. *remotePort* must either be "0" (indicating any source port) or a range of ports the same size as the other ranges.

This option is useful if you wish to run things like Internet phone on machines behind your gateway, but is limited in that connections to only one interior machine per source machine and target port are possible.

nat proto *proto localIP* [*publicIP* [*remoteIP*]]
This command tells **ppp** to redirect packets of protocol type *proto* (see protocols(5)) to the internal address *localIP*.

If *publicIP* is specified, only packets destined for that address are matched, otherwise the default alias address is used.

If *remoteIP* is specified, only packets matching that source address are matched,

This command is useful for redirecting tunnel endpoints to an internal machine, for example:

```
nat proto ipencap 10.0.0.1
```

nat proxy cmd *arg*...
This command tells **ppp** to proxy certain connections, redirecting them to a given server. Refer to the description of **PacketAliasProxyRule**() in libalias(3) for details of the available commands.

nat punch_fw [*base count*]
This command tells **ppp** to punch holes in the firewall for FTP or IRC DCC connections. This is done dynamically by installing temporary firewall rules which allow a particular connection (and only that connection) to go through the firewall. The rules are removed once the corresponding connection terminates.

A maximum of *count* rules starting from rule number *base* will be used for punching firewall holes. The range will be cleared when the "nat punch_fw" command is run.

If no arguments are given, firewall punching is disabled.

nat skinny_port [*port*]
This command tells **ppp** which TCP port is used by the Skinny Station protocol. Skinny is used by Cisco IP phones to communicate with Cisco Call Managers to setup voice over IP calls. The typical port used by Skinny is 2000.

If no argument is given, skinny aliasing is disabled.

nat same_ports yes|no
When enabled, this command will tell the network address translation engine to attempt to avoid changing the port number on outgoing packets. This is useful if you want to support protocols such as RPC and LPD which require connections to come from a well known port.

nat target [*address*]
Set the given target address or clear it if no address is given. The target address is used by libalias to specify how to NAT incoming packets by default. If a target address is not set or if "default" is given, packets are not altered and are allowed to route to the internal network.

The target address may be set to "MYADDR", in which case libalias will redirect all packets to the interface address.

nat use_sockets yes|no
When enabled, this option tells the network address translation engine to create a socket so that it can guarantee a correct incoming ftp data or IRC connection.

nat unregistered_only yes|no
Only alter outgoing packets with an unregistered source address. According to RFC 1918, unregistered source addresses are 10.0.0.0/8, 172.16.0.0/12 and 192.168.0.0/16.

These commands are also discussed in the file *README.nat* which comes with the source distribution.

[!]bg *command*
The given *command* is executed in the background with the following words replaced:

AUTHNAME This is replaced with the local *authname* value. See the "set authname" command below.

COMPILATIONDATE In previous software revisions, this was replaced with the date on which **ppp** was compiled. This is no longer supported as it breaks the ability to recompile the same code to produce an exact duplicate of a previous

compilation.

DNS0 & DNS1 — These are replaced with the primary and secondary nameserver IP numbers. If nameservers are negotiated by IPCP, the values of these macros will change.

ENDDISC — This is replaced with the local endpoint discriminator value. See the "set enddisc" command below.

HISADDR — This is replaced with the peers IP number.

HISADDR6 — This is replaced with the peers IPv6 number.

INTERFACE — This is replaced with the name of the interface that is in use.

IPOCTETSIN — This is replaced with the number of IP bytes received since the connection was established.

IPOCTETSOUT — This is replaced with the number of IP bytes sent since the connection was established.

IPPACKETSIN — This is replaced with the number of IP packets received since the connection was established.

IPPACKETSOUT — This is replaced with the number of IP packets sent since the connection was established.

IPV6OCTETSIN — This is replaced with the number of IPv6 bytes received since the connection was established.

IPV6OCTETSOUT — This is replaced with the number of IPv6 bytes sent since the connection was established.

IPV6PACKETSIN — This is replaced with the number of IPv6 packets received since the connection was established.

IPV6PACKETSOUT — This is replaced with the number of IPv6 packets sent since the connection was established.

LABEL — This is replaced with the last label name used. A label may be specified on

the **ppp** command line, via the "load" or "dial" commands and in the *ppp.secret* file.

MYADDR — This is replaced with the IP number assigned to the local interface.

MYADDR6 — This is replaced with the IPv6 number assigned to the local interface.

OCTETSIN — This is replaced with the number of bytes received since the connection was established.

OCTETSOUT — This is replaced with the number of bytes sent since the connection was established.

PACKETSIN — This is replaced with the number of packets received since the connection was established.

PACKETSOUT — This is replaced with the number of packets sent since the connection was established.

PEER_ENDDISC — This is replaced with the value of the peers endpoint discriminator.

PROCESSID — This is replaced with the current process id.

SOCKNAME — This is replaced with the name of the diagnostic socket.

UPTIME — This is replaced with the bundle uptime in HH:MM:SS format.

USER — This is replaced with the username that has been authenticated with PAP or CHAP. Normally, this variable is assigned only in -direct mode. This value is available irrespective of whether utmp logging is enabled.

VERSION — This is replaced with the current version number of **ppp**.

These substitutions are also done by the "set proctitle", "ident" and "log" commands.

If you wish to pause **ppp** while the command executes, use the "shell" command instead.

clear physical|ipcp|ipv6 [current|overall|peak...]

Clear the specified throughput values at either the "physical", "ipcp" or "ipv6cp" level. If "physical" is specified, context must be given (see the "link" command below). If no second argument is given,

all values are cleared.

clone *name*[,*name*]...
Clone the specified link, creating one or more new links according to the *name* argument(s). This command must be used from the "link" command below unless you have only got a single link (in which case that link becomes the default). Links may be removed using the "remove" command below.

The default link name is "deflink".

close [lcp|ccp[!]]
If no arguments are given, the relevant protocol layers will be brought down and the link will be closed. If "lcp" is specified, the LCP layer is brought down, but **ppp** will not bring the link offline. It is subsequently possible to use "term" (see below) to talk to the peer machine if, for example, something like "slirp" is being used. If "ccp" is specified, only the relevant compression layer is closed. If the "!" is used, the compression layer will remain in the closed state, otherwise it will re-enter the STOPPED state, waiting for the peer to initiate further CCP negotiation. In any event, this command does not disconnect the user from **ppp** or exit **ppp**. See the "quit" command below.

delete[!] *dest*
This command deletes the route with the given *dest* IP address. If *dest* is specified as 'ALL', all non-direct entries in the routing table for the current interface, and all 'sticky route' entries are deleted. If *dest* is specified as 'default', the default route is deleted.

If the *delete!* command is used (note the trailing "!"), **ppp** will not complain if the route does not already exist.

dial|call [*label*]...
This command is the equivalent of "load label" followed by "open", and is provided for backwards compatibility.

down [*lcp*|*ccp*]
Bring the relevant layer down ungracefully, as if the underlying layer had become unavailable. It is not considered polite to use this command on a Finite State Machine that is in the OPEN state. If no arguments are supplied, the entire link is closed (or if no context is given, all links are terminated). If 'lcp' is specified, the *LCP* layer is terminated but the device is not brought offline and the link is not closed. If 'ccp' is specified, only the relevant compression layer(s) are terminated.

help|? [*command*]
Show a list of available commands. If *command* is specified, show the usage string for that

command.

ident [*text*...]
Identify the link to the peer using *text*. If *text* is empty, link identification is disabled. It is possible to use any of the words described for the **bg** command above. Refer to the **sendident** command for details of when **ppp** identifies itself to the peer.

iface *command* [args]
This command is used to control the interface used by **ppp**. *Command* may be one of the following:

iface add[!] *addr*[/*bits*] [*peer*]

iface add[!] *addr mask peer*
Add the given *addr mask peer* combination to the interface. Instead of specifying *mask*, */bits* can be used (with no space between it and *addr*). If the given address already exists, the command fails unless the "!" is used - in which case the previous interface address entry is overwritten with the new one, allowing a change of netmask or peer address.

If only *addr* is specified, *bits* defaults to "32" and *peer* defaults to "255.255.255.255". This address (the broadcast address) is the only duplicate peer address that **ppp** allows.

iface clear [INET | INET6]
If this command is used while **ppp** is in the OPENED state or while in **-auto** mode, all addresses except for the NCP negotiated address are deleted from the interface. If **ppp** is not in the OPENED state and is not in **-auto** mode, all interface addresses are deleted.

If the INET or INET6 arguments are used, only addresses for that address family are cleared.

iface delete[!]|rm[!] *addr*
This command deletes the given *addr* from the interface. If the "!" is used, no error is given if the address is not currently assigned to the interface (and no deletion takes place).

iface name *name*
Renames the interface to *name*.

iface description *description*
Sets the interface description to *description*. Useful if you have many interfaces on your system.

iface show
Shows the current state and current addresses for the interface. It is much the same as running

"ifconfig INTERFACE".

iface help [*sub-command*]
This command, when invoked without *sub-command*, will show a list of possible "iface" sub-commands and a brief synopsis for each. When invoked with *sub-command*, only the synopsis for the given sub-command is shown.

[data]link *name*[,*name*]... *command* [*args*]
This command may prefix any other command if the user wishes to specify which link the command should affect. This is only applicable after multiple links have been created in Multi-link mode using the "clone" command.

Name specifies the name of an existing link. If *name* is a comma separated list, *command* is executed on each link. If *name* is "*", *command* is executed on all links.

load [*label*]...
Load the given *label*(s) from the *ppp.conf* file. If *label* is not given, the *default* label is used.

Unless the *label* section uses the "set mode", "open" or "dial" commands, **ppp** will not attempt to make an immediate connection.

log *word*...
Send the given word(s) to the log file with the prefix "LOG:". Word substitutions are done as explained under the "!bg" command above.

open [lcp|ccp|ipcp]
This is the opposite of the "close" command. All closed links are immediately brought up apart from second and subsequent *demand-dial* links - these will come up based on the "set autoload" command that has been used.

If the "lcp" argument is used while the LCP layer is already open, LCP will be renegotiated. This allows various LCP options to be changed, after which "open lcp" can be used to put them into effect. After renegotiating LCP, any agreed authentication will also take place.

If the "ccp" argument is used, the relevant compression layer is opened. Again, if it is already open, it will be renegotiated.

If the "ipcp" argument is used, the link will be brought up as normal, but if IPCP is already open, it will be renegotiated and the network interface will be reconfigured.

It is probably not good practice to re-open the PPP state machines like this as it is possible that the peer will not behave correctly. It *is* however useful as a way of forcing the CCP or VJ dictionaries to be reset.

passwd *pass*

Specify the password required for access to the full **ppp** command set. This password is required when connecting to the diagnostic port (see the "set server" command). *Pass* is specified on the "set server" command line. The value of *pass* is not logged when *command* logging is active, instead, the literal string '*********' is logged.

quit|bye [all]

If "quit" is executed from the controlling connection or from a command file, ppp will exit after closing all connections. Otherwise, if the user is connected to a diagnostic socket, the connection is simply dropped.

If the *all* argument is given, **ppp** will exit despite the source of the command after closing all existing connections.

remove|rm

This command removes the given link. It is only really useful in multi-link mode. A link must be in the CLOSED state before it is removed.

rename|mv *name*

This command renames the given link to *name*. It will fail if *name* is already used by another link.

The default link name is 'deflink'. Renaming it to 'modem', 'cuau0' or 'USR' may make the log file more readable.

resolv *command*

This command controls **ppp**'s manipulation of the resolv.conf(5) file. When **ppp** starts up, it loads the contents of this file into memory and retains this image for future use. *command* is one of the following:

readonly Treat */etc/resolv.conf* as read only. If "dns" is enabled, **ppp** will still attempt to negotiate nameservers with the peer, making the results available via the DNS0 and DNS1 macros. This is the opposite of the "resolv writable" command.

reload Reload */etc/resolv.conf* into memory. This may be necessary if for example a DHCP client overwrote */etc/resolv.conf*.

restore Replace */etc/resolv.conf* with the version originally read at startup or with the last "resolv reload" command. This is sometimes a useful command to put in the */etc/ppp/ppp.linkdown* file.

rewrite Rewrite the */etc/resolv.conf* file. This command will work even if the "resolv readonly" command has been used. It may be useful as a command in the */etc/ppp/ppp.linkup* file if you wish to defer updating */etc/resolv.conf* until after other commands have finished.

writable Allow **ppp** to update */etc/resolv.conf* if "dns" is enabled and **ppp** successfully negotiates a DNS. This is the opposite of the "resolv readonly" command.

save
This option is not (yet) implemented.

sendident
This command tells **ppp** to identify itself to the peer. The link must be in LCP state or higher. If no identity has been set (via the **ident** command), **sendident** will fail.

When an identity has been set, **ppp** will automatically identify itself when it sends or receives a configure reject, when negotiation fails or when LCP reaches the opened state.

Received identification packets are logged to the LCP log (see **set log** for details) and are never responded to.

set[up] *var value*
This option allows the setting of any of the following variables:

set accmap *hex-value*
ACCMap stands for Asynchronous Control Character Map. This is always negotiated with the peer, and defaults to a value of 00000000 in hex. This protocol is required to defeat hardware that depends on passing certain characters from end to end (such as XON/XOFF etc).

For the XON/XOFF scenario, use "set accmap 000a0000".

set [auth]key *value*
This sets the authentication key (or password) used in client mode PAP or CHAP negotiation to the given value. It also specifies the password to be used in the dial or login scripts in place of the '\P' sequence, preventing the actual password from being logged. If *command* or *chat* logging is in effect, *value* is logged as '********' for security reasons.

If the first character of *value* is an exclamation mark ("!"), **ppp** treats the remainder of the string as a program that must be executed to determine the "authname" and "authkey" values.

If the "!" is doubled up (to "!!"), it is treated as a single literal "!", otherwise, ignoring the "!", *value* is parsed as a program to execute in the same was as the "!bg" command above, substituting special names in the same manner. Once executed, **ppp** will feed the program three lines of input, each terminated by a newline character:

- The host name as sent in the CHAP challenge.
- The challenge string as sent in the CHAP challenge.
- The locally defined "authname".

Two lines of output are expected:

- The "authname" to be sent with the CHAP response.
- The "authkey", which is encrypted with the challenge and request id, the answer being sent in the CHAP response packet.

When configuring **ppp** in this manner, it is expected that the host challenge is a series of ASCII digits or characters. An encryption device or Secure ID card is usually required to calculate the secret appropriate for the given challenge.

set authname *id*

This sets the authentication id used in client mode PAP or CHAP negotiation.

If used in **-direct** mode with CHAP enabled, *id* is used in the initial authentication challenge and should normally be set to the local machine name.

set autoload *min-percent max-percent period*

These settings apply only in multi-link mode and default to zero, zero and five respectively. When more than one *demand-dial* (also known as **-auto**) mode link is available, only the first link is made active when **ppp** first reads data from the tun device. The next *demand-dial* link will be opened only when the current bundle throughput is at least *max-percent* percent of the total bundle bandwidth for *period* seconds. When the current bundle throughput decreases to *min-percent* percent or less of the total bundle bandwidth for *period* seconds, a *demand-dial* link will be brought down as long as it is not the last active link.

Bundle throughput is measured as the maximum of inbound and outbound traffic.

The default values cause *demand-dial* links to simply come up one at a time.

Certain devices cannot determine their physical bandwidth, so it is sometimes necessary to use the "set bandwidth" command (described below) to make "set autoload" work correctly.

set bandwidth *value*

This command sets the connection bandwidth in bits per second. *value* must be greater than zero. It is currently only used by the "set autoload" command above.

set callback *option...*

If no arguments are given, callback is disabled, otherwise, **ppp** will request (or in **-direct** mode, will accept) one of the given *option*s. In client mode, if an *option* is NAK'd **ppp** will request a different *option*, until no options remain at which point **ppp** will terminate negotiations (unless "none" is one of the specified *option*). In server mode, **ppp** will accept any of the given protocols - but the client *must* request one of them. If you wish callback to be optional, you must {include} *none* as an option.

The *option*s are as follows (in this order of preference):

auth The callee is expected to decide the callback number based on authentication. If **ppp** is the callee, the number should be specified as the fifth field of the peers entry in */etc/ppp/ppp.secret*.

cbcp Microsoft's callback control protocol is used. See "set cbcp" below.

If you wish to negotiate *cbcp* in client mode but also wish to allow the server to request no callback at CBCP negotiation time, you must specify both *cbcp* and *none* as callback options.

E.164 *|*number*[,*number*]...

The caller specifies the *number*. If **ppp** is the callee, *number* should be either a comma separated list of allowable numbers or a "*", meaning any number is permitted. If **ppp** is the caller, only a single number should be specified.

Note, this option is very unsafe when used with a "*" as a malicious caller can tell **ppp** to call any (possibly international) number without first authenticating themselves.

none If the peer does not wish to do callback at all, **ppp** will accept the fact and continue

without callback rather than terminating the connection. This is required (in addition to one or more other callback options) if you wish callback to be optional.

set cbcp [*|*number*[,*number*...] [*delay* [*retry*]]]
If no arguments are given, CBCP (Microsoft's CallBack Control Protocol) is disabled - ie, configuring CBCP in the "set callback" command will result in **ppp** requesting no callback in the CBCP phase. Otherwise, **ppp** attempts to use the given phone *number*(s).

In server mode (**-direct**), **ppp** will insist that the client uses one of these numbers, unless "*" is used in which case the client is expected to specify the number.

In client mode, **ppp** will attempt to use one of the given numbers (whichever it finds to be agreeable with the peer), or if "*" is specified, **ppp** will expect the peer to specify the number.

set cd [off|*seconds*[!]]
Normally, **ppp** checks for the existence of carrier depending on the type of device that has been opened:

Terminal Devices
Carrier is checked one second after the login script is complete. If it is not set, **ppp** assumes that this is because the device does not support carrier (which is true for most "laplink" NULL-modem cables), logs the fact and stops checking for carrier.

As ptys do not support the TIOCMGET ioctl, the tty device will switch all carrier detection off when it detects that the device is a pty.

PPPoE (netgraph) Devices
Carrier is checked once per second for 5 seconds. If it is not set after the fifth second, the connection attempt is considered to have failed and the device is closed. Carrier is always required for PPPoE devices.

All other device types do not support carrier. Setting a carrier value will result in a warning when the device is opened.

Some modems take more than one second after connecting to assert the carrier signal. If this delay is not increased, this will result in **ppp**'s inability to detect when the link is dropped, as **ppp** assumes that the device is not asserting carrier.

The "set cd" command overrides the default carrier behaviour. *seconds* specifies the maximum number of seconds that **ppp** should wait after the dial script has finished before deciding if carrier

is available or not.

If "off" is specified, **ppp** will not check for carrier on the device, otherwise **ppp** will not proceed to the login script until either carrier is detected or until *seconds* has elapsed, at which point **ppp** assumes that the device will not set carrier.

If no arguments are given, carrier settings will go back to their default values.

If *seconds* is followed immediately by an exclamation mark ("!"), **ppp** will *require* carrier. If carrier is not detected after *seconds* seconds, the link will be disconnected.

set choked [*timeout*]
This sets the number of seconds that **ppp** will keep a choked output queue before dropping all pending output packets. If *timeout* is less than or equal to zero or if *timeout* is not specified, it is set to the default value of *120 seconds*.

A choked output queue occurs when **ppp** has read a certain number of packets from the local network for transmission, but cannot send the data due to link failure (the peer is busy etc.). **ppp** will not read packets indefinitely. Instead, it reads up to *30* packets (or *30 + nlinks * 2* packets in multi-link mode), then stops reading the network interface until either *timeout* seconds have passed or at least one packet has been sent.

If *timeout* seconds pass, all pending output packets are dropped.

set ctsrts|crtscts on|off
This sets hardware flow control. Hardware flow control is *on* by default.

set deflate *out-winsize* [*in-winsize*]
This sets the DEFLATE algorithms default outgoing and incoming window sizes. Both *out-winsize* and *in-winsize* must be values between *8* and *15*. If *in-winsize* is specified, **ppp** will insist that this window size is used and will not accept any other values from the peer.

set dns [*primary* [*secondary*]]
This command specifies DNS overrides for the "accept dns" command. Refer to the "accept" command description above for details. This command does not affect the IP numbers requested using "enable dns".

set device|line *value*...
This sets the device(s) to which **ppp** will talk to the given "value".

All serial device names are expected to begin with */dev/*. Serial devices are usually called *cuaXX*.

If "value" does not begin with */dev/*, it must either begin with an exclamation mark ("!"), be of the format PPPoE:*iface*[:*provider*](on netgraph(4) enabled systems), or be of the format *host*:*port*[/tcp|udp].

If it begins with an exclamation mark, the rest of the device name is treated as a program name, and that program is executed when the device is opened. Standard input, output and error are fed back to **ppp** and are read and written as if they were a regular device.

If a PPPoE:*iface*[:*provider*]specification is given, **ppp** will attempt to create a *PPP* over Ethernet connection using the given *iface* interface by using netgraph(4). If netgraph(4) is not available, **ppp** will attempt to load it using kldload(2). If this fails, an external program must be used such as the pppoed(8) program available under OpenBSD. The given *provider* is passed as the service name in the PPPoE Discovery Initiation (PADI) packet. If no provider is given, an empty value will be used.

When a PPPoE connection is established, **ppp** will place the name of the Access Concentrator in the environment variable ACNAME.

Refer to netgraph(4) and ng_pppoe(4) for further details.

If a *host*:*port*[/tcp|udp] specification is given, **ppp** will attempt to connect to the given *host* on the given *port*. If a "/tcp" or "/udp" suffix is not provided, the default is "/tcp". Refer to the section on *PPP OVER TCP and UDP* above for further details.

If multiple "values" are specified, **ppp** will attempt to open each one in turn until it succeeds or runs out of devices.

set dial *chat-script*

This specifies the chat script that will be used to dial the other side. See also the "set login" command below. Refer to chat(8) and to the example configuration files for details of the chat script format. It is possible to specify some special 'values' in your chat script as follows:

\c When used as the last character in a 'send' string, this indicates that a newline should not be appended.

\d When the chat script encounters this sequence, it delays two seconds.

\p When the chat script encounters this sequence, it delays for one quarter of a second.

\n This is replaced with a newline character.

\r This is replaced with a carriage return character.

\s This is replaced with a space character.

\t This is replaced with a tab character.

\T This is replaced by the current phone number (see "set phone" below).

\P This is replaced by the current *authkey* value (see "set authkey" above).

\U This is replaced by the current *authname* value (see "set authname" above).

Note that two parsers will examine these escape sequences, so in order to have the 'chat parser' see the escape character, it is necessary to escape it from the 'command parser'. This means that in practice you should use two escapes, for example:

```
set dial "... ATDT\\T CONNECT"
```

It is also possible to execute external commands from the chat script. To do this, the first character of the expect or send string is an exclamation mark ("!"). If a literal exclamation mark is required, double it up to "!!" and it will be treated as a single literal "!". When the command is executed, standard input and standard output are directed to the open device (see the "set device" command), and standard error is read by **ppp** and substituted as the expect or send string. If **ppp** is running in interactive mode, file descriptor 3 is attached to */dev/tty*.

For example (wrapped for readability):

```
set login "TIMEOUT 5 \"\" \"\" login:--login: ppp \
word: ppp \"!sh \\-c \\\"echo \\-n label: >&2\\\"\" \
\"!/bin/echo in\" HELLO"
```

would result in the following chat sequence (output using the 'set log local chat' command before dialing):

```
Dial attempt 1 of 1
dial OK!
```

```
Chat: Expecting:
Chat: Sending:
Chat: Expecting: login:--login:
Chat: Wait for (5): login:
Chat: Sending: ppp
Chat: Expecting: word:
Chat: Wait for (5): word:
Chat: Sending: ppp
Chat: Expecting: !sh \-c "echo \-n label: >&2"
Chat: Exec: sh -c "echo -n label: >&2"
Chat: Wait for (5): !sh \-c "echo \-n label: >&2" --> label:
Chat: Exec: /bin/echo in
Chat: Sending:
Chat: Expecting: HELLO
Chat: Wait for (5): HELLO
login OK!
```

Note (again) the use of the escape character, allowing many levels of nesting. Here, there are four parsers at work. The first parses the original line, reading it as three arguments. The second parses the third argument, reading it as 11 arguments. At this point, it is important that the "-" signs are escaped, otherwise this parser will see them as constituting an expect-send-expect sequence. When the "!" character is seen, the execution parser reads the first command as three arguments, and then sh(1) itself expands the argument after the **-c**. As we wish to send the output back to the modem, in the first example we redirect our output to file descriptor 2 (stderr) so that **ppp** itself sends and logs it, and in the second example, we just output to stdout, which is attached directly to the modem.

This, of course means that it is possible to execute an entirely external "chat" command rather than using the internal one. See chat(8) for a good alternative.

The external command that is executed is subjected to the same special word expansions as the "!bg" command.

set enddisc [label|IP|MAC|magic|psn value]

This command sets our local endpoint discriminator. If set prior to LCP negotiation, and if no "disable enddisc" command has been used, **ppp** will send the information to the peer using the LCP endpoint discriminator option. The following discriminators may be set:

label The current label is used.

IP Our local IP number is used. As LCP is negotiated prior to IPCP, it is possible that the IPCP layer will subsequently change this value. If it does, the endpoint discriminator stays at the old value unless manually reset.

MAC This is similar to the *IP* option above, except that the MAC address associated with the local IP number is used. If the local IP number is not resident on any Ethernet interface, the command will fail.

As the local IP number defaults to whatever the machine host name is, "set enddisc mac" is usually done prior to any "set ifaddr" commands.

magic A 20 digit random number is used. Care should be taken when using magic numbers as restarting **ppp** or creating a link using a different **ppp** invocation will also use a different magic number and will therefore not be recognised by the peer as belonging to the same bundle. This makes it unsuitable for **-direct** connections.

psn *value*
The given *value* is used. *Value* should be set to an absolute public switched network number with the country code first.

If no arguments are given, the endpoint discriminator is reset.

set escape *value...*
This option is similar to the "set accmap" option above. It allows the user to specify a set of characters that will be 'escaped' as they travel across the link.

set filter dial|alive|in|out *rule-no* permit|deny|clear|*rule-no* [!] [[host] *src_addr*[/*width*] [*dst_addr*[/*width*]]] [*proto* [src lt|eq|gt *port*] [dst lt|eq|gt *port*] [estab] [syn] [finrst] [timeout *secs*]]
ppp supports four filter sets. The *alive* filter specifies packets that keep the connection alive - resetting the idle timer. The *dial* filter specifies packets that cause **ppp** to dial when in **-auto** mode. The *in* filter specifies packets that are allowed to travel into the machine and the *out* filter specifies packets that are allowed out of the machine.

Filtering is done prior to any IP alterations that might be done by the NAT engine on outgoing packets and after any IP alterations that might be done by the NAT engine on incoming packets. By default all empty filter sets allow all packets to pass. Rules are processed in order according to *rule-no* (unless skipped by specifying a rule number as the *action*). Up to 40 rules may be given for each set. If a packet does not match any of the rules in a given set, it is discarded. In the case of *in* and *out* filters, this means that the packet is dropped. In the case of *alive* filters it means that the packet will not reset the idle timer (even if the *in*/*out* filter has a "timeout" value)

and in the case of *dial* filters it means that the packet will not trigger a dial. A packet failing to trigger a dial will be dropped rather than queued. Refer to the section on *PACKET FILTERING* above for further details.

set hangup *chat-script*

This specifies the chat script that will be used to reset the device before it is closed. It should not normally be necessary, but can be used for devices that fail to reset themselves properly on close.

set help|? [*command*]

This command gives a summary of available set commands, or if *command* is specified, the command usage is shown.

set ifaddr [*myaddr*[/*nn*] [*hisaddr*[/*nn*] [*netmask* [*triggeraddr*]]]]

This command specifies the IP addresses that will be used during IPCP negotiation. Addresses are specified using the format

a.b.c.d/nn

Where "a.b.c.d" is the preferred IP, but *nn* specifies how many bits of the address we will insist on. If /*nn* is omitted, it defaults to "/32" unless the IP address is 0.0.0.0 in which case it defaults to "/0".

If you wish to assign a dynamic IP number to the peer, *hisaddr* may also be specified as a range of IP numbers in the format

IP[-*IP*][,*IP*[-*IP*]]...

for example:

set ifaddr 10.0.0.1 10.0.1.2-10.0.1.10,10.0.1.20

will only negotiate "10.0.0.1" as the local IP number, but may assign any of the given 10 IP numbers to the peer. If the peer requests one of these numbers, and that number is not already in use, **ppp** will grant the peers request. This is useful if the peer wants to re-establish a link using the same IP number as was previously allocated (thus maintaining any existing tcp or udp connections).

If the peer requests an IP number that is either outside of this range or is already in use, **ppp** will suggest a random unused IP number from the range.

If *triggeraddr* is specified, it is used in place of *myaddr* in the initial IPCP negotiation. However, only an address in the *myaddr* range will be accepted. This is useful when negotiating with some PPP implementations that will not assign an IP number unless their peer requests "0.0.0.0".

It should be noted that in **-auto** mode, **ppp** will configure the interface immediately upon reading the "set ifaddr" line in the config file. In any other mode, these values are just used for IPCP negotiations, and the interface is not configured until the IPCP layer is up.

Note that the *HISADDR* argument may be overridden by the third field in the *ppp.secret* file once the client has authenticated itself (if PAP or CHAP are "enabled"). Refer to the *AUTHENTICATING INCOMING CONNECTIONS* section for details.

In all cases, if the interface is already configured, **ppp** will try to maintain the interface IP numbers so that any existing bound sockets will remain valid.

set ifqueue *packets*

Set the maximum number of packets that **ppp** will read from the tunnel interface while data cannot be sent to any of the available links. This queue limit is necessary to flow control outgoing data as the tunnel interface is likely to be far faster than the combined links available to **ppp**.

If *packets* is set to a value less than the number of links, **ppp** will read up to that value regardless. This prevents any possible latency problems.

The default value for *packets* is "30".

set ccpretry|ccpretries [*timeout* [*reqtries* [*trmtries*]]]

set chapretry|chapretries [*timeout* [*reqtries*]]

set ipcpretry|ipcpretries [*timeout* [*reqtries* [*trmtries*]]]

set ipv6cpretry|ipv6cpretries [*timeout* [*reqtries* [*trmtries*]]]

set lcpretry|lcpretries [*timeout* [*reqtries* [*trmtries*]]]

set papretry|papretries [*timeout* [*reqtries*]]

These commands set the number of seconds that **ppp** will wait before resending Finite State Machine (FSM) Request packets. The default *timeout* for all FSMs is 3 seconds (which should suffice in most cases).

If *reqtries* is specified, it tells **ppp** how many configuration request attempts it should make while receiving no reply from the peer before giving up. The default is 5 attempts for CCP, LCP and IPCP and 3 attempts for PAP and CHAP.

If *trmtries* is specified, it tells **ppp** how many terminate requests should be sent before giving up waiting for the peers response. The default is 3 attempts. Authentication protocols are not terminated and it is therefore invalid to specify *trmtries* for PAP or CHAP.

In order to avoid negotiations with the peer that will never converge, **ppp** will only send at most 3 times the configured number of *reqtries* in any given negotiation session before giving up and closing that layer.

set log [local] [+|-]*value*...
This command allows the adjustment of the current log level. Refer to the Logging Facility section for further details.

set login *chat-script*
This *chat-script* compliments the dial-script. If both are specified, the login script will be executed after the dial script. Escape sequences available in the dial script are also available here.

set logout *chat-script*
This specifies the chat script that will be used to logout before the hangup script is called. It should not normally be necessary.

set lqrperiod|echoperiod *frequency*
This command sets the *frequency* in seconds at which *LQR* or *LCP ECHO* packets are sent. The default is 30 seconds. You must also use the "enable lqr" and/or "enable echo" commands if you wish to send *LQR* or *LCP ECHO* requests to the peer.

set mode *interactive*|*auto*|*ddial*|*background*
This command allows you to change the 'mode' of the specified link. This is normally only useful in multi-link mode, but may also be used in uni-link mode.

It is not possible to change a link that is 'direct' or 'dedicated'.

Note: If you issue the command "set mode auto", and have network address translation enabled, it may be useful to "enable iface-alias" afterwards. This will allow **ppp** to do the necessary address translations to enable the process that triggers the connection to connect once the link is up despite the peer assigning us a new (dynamic) IP address.

set mppe [40|56|128|* [stateless|stateful|*]]

This option selects the encryption parameters used when negotiation MPPE. MPPE can be disabled entirely with the "disable mppe" command. If no arguments are given, **ppp** will attempt to negotiate a stateful link with a 128 bit key, but will agree to whatever the peer requests (including no encryption at all).

If any arguments are given, **ppp** will *insist* on using MPPE and will close the link if it is rejected by the peer (Note; this behaviour can be overridden by a configured RADIUS server).

The first argument specifies the number of bits that **ppp** should insist on during negotiations and the second specifies whether **ppp** should insist on stateful or stateless mode. In stateless mode, the encryption dictionary is re-initialised with every packet according to an encryption key that is changed with every packet. In stateful mode, the encryption dictionary is re-initialised every 256 packets or after the loss of any data and the key is changed every 256 packets. Stateless mode is less efficient but is better for unreliable transport layers.

set mrru [*value*]

Setting this option enables Multi-link PPP negotiations, also known as Multi-link Protocol or MP. There is no default MRRU (Maximum Reconstructed Receive Unit) value. If no argument is given, multi-link mode is disabled.

set mru [max[imum]] [*value*]

The default MRU (Maximum Receive Unit) is 1500. If it is increased, the other side *may* increase its MTU. In theory there is no point in decreasing the MRU to below the default as the *PPP* protocol says implementations *must* be able to accept packets of at least 1500 octets.

If the "maximum" keyword is used, **ppp** will refuse to negotiate a higher value. The maximum MRU can be set to 2048 at most. Setting a maximum of less than 1500 violates the *PPP* rfc, but may sometimes be necessary. For example, *PPPoE* imposes a maximum of 1492 due to hardware limitations.

If no argument is given, 1500 is assumed. A value must be given when "maximum" is specified.

set mtu [max[imum]] [*value*]

The default MTU is 1500. At negotiation time, **ppp** will accept whatever MRU the peer requests (assuming it is not less than 296 bytes or greater than the assigned maximum). If the MTU is set, **ppp** will not accept MRU values less than *value*. When negotiations are complete, the MTU is used when writing to the interface, even if the peer requested a higher value MRU. This can be useful for limiting your packet size (giving better bandwidth sharing at the expense of more header data).

If the "maximum" keyword is used, **ppp** will refuse to negotiate a higher value. The maximum MTU can be set to 2048 at most. Note, it is necessary to use the "maximum" keyword to limit the MTU when using PPPoE.

If no *value* is given, 1500, or whatever the peer asks for is used. A value must be given when "maximum" is specified.

set nbns [*x.x.x.x* [*y.y.y.y*]]
This option allows the setting of the Microsoft NetBIOS name server values to be returned at the peers request. If no values are given, **ppp** will reject any such requests.

set openmode active|passive [*delay*]
By default, *openmode* is always *active* with a one second *delay*. That is, **ppp** will always initiate LCP/IPCP/CCP negotiation one second after the line comes up. If you want to wait for the peer to initiate negotiations, you can use the value *passive*. If you want to initiate negotiations immediately or after more than one second, the appropriate *delay* may be specified here in seconds.

set parity odd|even|none|mark
This allows the line parity to be set. The default value is *none*.

set phone *telno*[|*backupnumber*]...[:*nextnumber*]...
This allows the specification of the phone number to be used in place of the \\T string in the dial and login chat scripts. Multiple phone numbers may be given separated either by a pipe ("|") or a colon (":").

Numbers after the pipe are only dialed if the dial or login script for the previous number failed.

Numbers after the colon are tried sequentially, irrespective of the reason the line was dropped.

If multiple numbers are given, **ppp** will dial them according to these rules until a connection is made, retrying the maximum number of times specified by "set redial" below. In **-background** mode, each number is attempted at most once.

set pppoe [standard|3Com]
This option configures the underlying ng_pppoe(4) node to either standard RFC2516 PPPoE or proprietary 3Com mode. If not set the system default will be used.

set [proc]title [*value*]
The current process title as displayed by ps(1) is changed according to *value*. If *value* is not

specified, the original process title is restored. All the word replacements done by the shell commands (see the "bg" command above) are done here too.

Note, if USER is required in the process title, the "set proctitle" command must appear in *ppp.linkup*, as it is not known when the commands in *ppp.conf* are executed.

set radius [*config-file*]

This command enables RADIUS support (if it is compiled in). *config-file* refers to the radius client configuration file as described in radius.conf(5). If PAP, CHAP, MSCHAP or MSCHAPv2 are "enabled", **ppp** behaves as a *N*etwork *A*ccess *S*erver and uses the configured RADIUS server to authenticate rather than authenticating from the *ppp.secret* file or from the passwd database.

If none of PAP, CHAP, MSCHAP or MSCHAPv2 are enabled, "set radius" will do nothing.

ppp uses the following attributes from the RADIUS reply:

RAD_FRAMED_IP_ADDRESS

The peer IP address is set to the given value.

RAD_FRAMED_IP_NETMASK

The tun interface netmask is set to the given value.

RAD_FRAMED_MTU

If the given MTU is less than the peers MRU as agreed during LCP negotiation, *and* it is less that any configured MTU (see the "set mru" command), the tun interface MTU is set to the given value.

RAD_FRAMED_COMPRESSION

If the received compression type is "1", **ppp** will request VJ compression during IPCP negotiations despite any "disable vj" configuration command.

RAD_FILTER_ID

If this attribute is supplied, **ppp** will attempt to use it as an additional label to load from the *ppp.linkup* and *ppp.linkdown* files. The load will be attempted before (and in addition to) the normal label search. If the label does not exist, no action is taken and **ppp** proceeds to the normal load using the current label.

RAD_FRAMED_ROUTE

The received string is expected to be in the format *dest*[/*bits*] *gw* [*metrics*]. Any

specified metrics are ignored. MYADDR and HISADDR are understood as valid values for *dest* and *gw*, "default" can be used for *dest* to specify the default route, and "0.0.0.0" is understood to be the same as "default" for *dest* and HISADDR for *gw*.

For example, a returned value of "1.2.3.4/24 0.0.0.0 1 2 -1 3 400" would result in a routing table entry to the 1.2.3.0/24 network via HISADDR and a returned value of "0.0.0.0 0.0.0.0" or "default HISADDR" would result in a default route to HISADDR.

All RADIUS routes are applied after any sticky routes are applied, making RADIUS routes override configured routes. This also applies for RADIUS routes that do not {include} the MYADDR or HISADDR keywords.

RAD_FRAMED_IPV6_PREFIX

If this attribute is supplied, the value is substituted for IPV6PREFIX in a command. You may pass it to an upper layer protocol such as DHCPv6 for delegating an IPv6 prefix to a peer.

RAD_FRAMED_IPV6_ROUTE

The received string is expected to be in the format *dest*[/*bits*] *gw* [*metrics*]. Any specified metrics are ignored. MYADDR6 and HISADDR6 are understood as valid values for *dest* and *gw*, "default" can be used for *dest* to specify the default route, and "::" is understood to be the same as "default" for *dest* and HISADDR6 for *gw*.

For example, a returned value of "3ffe:505:abcd::/48 ::" would result in a routing table entry to the 3ffe:505:abcd::/48 network via HISADDR6 and a returned value of ":: ::" or "default HISADDR6" would result in a default route to HISADDR6.

All RADIUS IPv6 routes are applied after any sticky routes are applied, making RADIUS IPv6 routes override configured routes. This also applies for RADIUS IPv6 routes that do not {include} the MYADDR6 or HISADDR6 keywords.

RAD_SESSION_TIMEOUT

If supplied, the client connection is closed after the given number of seconds.

RAD_REPLY_MESSAGE

If supplied, this message is passed back to the peer as the authentication SUCCESS text.

RAD_MICROSOFT_MS_CHAP_ERROR

If this RAD_VENDOR_MICROSOFT vendor specific attribute is supplied, it is passed back to the peer as the authentication FAILURE text.

RAD_MICROSOFT_MS_CHAP2_SUCCESS
If this RAD_VENDOR_MICROSOFT vendor specific attribute is supplied and if MS-CHAPv2 authentication is being used, it is passed back to the peer as the authentication SUCCESS text.

RAD_MICROSOFT_MS_MPPE_ENCRYPTION_POLICY
If this RAD_VENDOR_MICROSOFT vendor specific attribute is supplied and has a value of 2 (Required), **ppp** will insist that MPPE encryption is used (even if no "set mppe" configuration command has been given with arguments). If it is supplied with a value of 1 (Allowed), encryption is made optional (despite any "set mppe" configuration commands with arguments).

RAD_MICROSOFT_MS_MPPE_ENCRYPTION_TYPES
If this RAD_VENDOR_MICROSOFT vendor specific attribute is supplied, bits 1 and 2 are examined. If either or both are set, 40 bit and/or 128 bit (respectively) encryption options are set, overriding any given first argument to the "set mppe" command. Note, it is not currently possible for the RADIUS server to specify 56 bit encryption.

RAD_MICROSOFT_MS_MPPE_RECV_KEY
If this RAD_VENDOR_MICROSOFT vendor specific attribute is supplied, it is value is used as the master key for decryption of incoming data. When clients are authenticated using MSCHAPv2, the RADIUS server MUST provide this attribute if inbound MPPE is to function.

RAD_MICROSOFT_MS_MPPE_SEND_KEY
If this RAD_VENDOR_MICROSOFT vendor specific attribute is supplied, it is value is used as the master key for encryption of outgoing data. When clients are authenticated using MSCHAPv2, the RADIUS server MUST provide this attribute if outbound MPPE is to function.

Values received from the RADIUS server may be viewed using "show bundle".

set rad_alive *timeout*
When RADIUS is configured, setting "rad_alive" to a non-zero *timeout* value will tell **ppp** to sent RADIUS accounting information to the RADIUS server every *timeout* seconds.

set rad_port_id *option*

When RADIUS is configured, setting the "rad_port_id" value specifies what should be sent to the RADIUS server as NAS-Port-Id. The *option*s are as follows:

pid PID of the corresponding tunnel.

tunnum
tun(4) interface number.

ifnum index of the interface as returned by if_nametoindex(3).

default keeps the default behavior.

set reconnect *timeout ntries*

Should the line drop unexpectedly (due to loss of CD or LQR failure), a connection will be re-established after the given *timeout*. The line will be re-connected at most *ntries* times. *Ntries* defaults to zero. A value of *random* for *timeout* will result in a variable pause, somewhere between 1 and 30 seconds.

set recvpipe [*value*]

This sets the routing table RECVPIPE value. The optimum value is just over twice the MTU value. If *value* is unspecified or zero, the default kernel controlled value is used.

set redial *secs*[+*inc*[-*max*]][.*next*] [*attempts*]

ppp can be instructed to attempt to redial *attempts* times. If more than one phone number is specified (see "set phone" above), a pause of *next* is taken before dialing each number. A pause of *secs* is taken before starting at the first number again. A literal value of "random" may be used here in place of *secs* and *next*, causing a random delay of between 1 and 30 seconds.

If *inc* is specified, its value is added onto *secs* each time **ppp** tries a new number. *secs* will only be incremented at most *max* times. *max* defaults to 10.

Note, the *secs* delay will be effective, even after *attempts* has been exceeded, so an immediate manual dial may appear to have done nothing. If an immediate dial is required, a "!" should immediately follow the "open" keyword. See the "open" description above for further details.

set sendpipe [*value*]

This sets the routing table SENDPIPE value. The optimum value is just over twice the MTU value. If *value* is unspecified or zero, the default kernel controlled value is used.

set server|socket *TcpPort*|*LocalName*|none|open|closed [password [*mask*]]

This command tells **ppp** to listen on the given socket or 'diagnostic port' for incoming command connections.

The word "none" instructs **ppp** to close any existing socket and clear the socket configuration. The word "open" instructs **ppp** to attempt to re-open the port. The word "closed" instructs **ppp** to close the open port.

If you wish to specify a local domain socket, *LocalName* must be specified as an absolute file name, otherwise it is assumed to be the name or number of a TCP port. You may specify the octal umask to be used with a local domain socket. Refer to umask(2) for umask details. Refer to services(5) for details of how to translate TCP port names.

You must also specify the password that must be entered by the client (using the "passwd" variable above) when connecting to this socket. If the password is specified as an empty string, no password is required for connecting clients.

When specifying a local domain socket, the first "%d" sequence found in the socket name will be replaced with the current interface unit number. This is useful when you wish to use the same profile for more than one connection.

In a similar manner TCP sockets may be prefixed with the "+" character, in which case the current interface unit number is added to the port number.

When using **ppp** with a server socket, the pppctl(8) command is the preferred mechanism of communications. Currently, telnet(1) can also be used, but link encryption may be implemented in the future, so telnet(1) should be avoided.

Note; SIGUSR1 and SIGUSR2 interact with the diagnostic socket.

set speed *value*

This sets the speed of the serial device. If speed is specified as "sync", **ppp** treats the device as a synchronous device.

Certain device types will know whether they should be specified as synchronous or asynchronous. These devices will override incorrect settings and log a warning to this effect.

set stopped [*LCPseconds* [*CCPseconds*]]

If this option is set, **ppp** will time out after the given FSM (Finite State Machine) has been in the stopped state for the given number of "seconds". This option may be useful if the peer sends a

terminate request, but never actually closes the connection despite our sending a terminate acknowledgement. This is also useful if you wish to "set openmode passive" and time out if the peer does not send a Configure Request within the given time. Use "set log +lcp +ccp" to make **ppp** log the appropriate state transitions.

The default value is zero, where **ppp** does not time out in the stopped state.

This value should not be set to less than the openmode delay (see "set openmode" above).

set timeout *idleseconds* [*mintimeout*]
This command allows the setting of the idle timer. Refer to the section titled *SETTING THE IDLE TIMER* for further details.

If *mintimeout* is specified, **ppp** will never idle out before the link has been up for at least that number of seconds.

set urgent [tcp|udp|none] [[+|-]*port*] ...
This command controls the ports that **ppp** prioritizes when transmitting data. The default priority TCP ports are ports 21 (ftp control), 22 (ssh), 23 (telnet), 513 (login), 514 (shell), 543 (klogin) and 544 (kshell). There are no priority UDP ports by default. See services(5) for details.

If neither "tcp" or "udp" are specified, "tcp" is assumed.

If no *port*s are given, the priority port lists are cleared (although if "tcp" or "udp" is specified, only that list is cleared). If the first *port* argument is prefixed with a plus ("+") or a minus ("-"), the current list is adjusted, otherwise the list is reassigned. *port*s prefixed with a plus or not prefixed at all are added to the list and *port*s prefixed with a minus are removed from the list.

If "none" is specified, all priority port lists are disabled and even IPTOS_LOWDELAY packets are not prioritised.

set urgent length *length*
This command tells ppp to prioritize small packets up to *length* bytes. If *length* is not specified, or 0, this feature is disabled.

set vj slotcomp on|off
This command tells **ppp** whether it should attempt to negotiate VJ slot compression. By default, slot compression is turned *on*.

set vj slots *nslots*

This command sets the initial number of slots that **ppp** will try to negotiate with the peer when VJ compression is enabled (see the 'enable' command above). It defaults to a value of 16. *Nslots* must be between *4* and *16* inclusive.

shell|! [*command*]

If *command* is not specified a shell is invoked according to the SHELL environment variable. Otherwise, the given *command* is executed. Word replacement is done in the same way as for the "!bg" command as described above.

Use of the ! character requires a following space as with any of the other commands. You should note that this command is executed in the foreground; **ppp** will not continue running until this process has exited. Use the bg command if you wish processing to happen in the background.

show *var*

This command allows the user to examine the following:

show bundle

Show the current bundle settings.

show ccp

Show the current CCP compression statistics.

show compress

Show the current VJ compression statistics.

show escape

Show the current escape characters.

show filter [*name*]

List the current rules for the given filter. If *name* is not specified, all filters are shown.

show hdlc

Show the current HDLC statistics.

show help|?

Give a summary of available show commands.

show iface

Show the current interface information (the same as "iface show").

show ipcp
Show the current IPCP statistics.

show layers
Show the protocol layers currently in use.

show lcp
Show the current LCP statistics.

show [data]link
Show high level link information.

show links
Show a list of available logical links.

show log
Show the current log values.

show mem
Show current memory statistics.

show ncp
Show the current NCP statistics.

show physical
Show low level link information.

show mp
Show Multi-link information.

show proto
Show current protocol totals.

show route
Show the current routing tables.

show stopped
Show the current stopped timeouts.

show timer

Show the active alarm timers.

show version
Show the current version number of **ppp**.

term
Go into terminal mode. Characters typed at the keyboard are sent to the device. Characters read from the device are displayed on the screen. When a remote *PPP* peer is detected, **ppp** automatically enables Packet Mode and goes back into command mode.

MORE DETAILS

- Read the example configuration files. They are a good source of information.
- Use "help", "nat ?", "enable ?", "set ?" and "show ?" to get online information about what is available.
- The following URL contains useful information:
  - https://docs.freebsd.org/en/books/handbook/ppp-and-slip/

FILES

ppp refers to four files: *ppp.conf*, *ppp.linkup*, *ppp.linkdown* and *ppp.secret*. These files are placed in the */etc/ppp* directory.

/etc/ppp/ppp.conf
System default configuration file.

/etc/ppp/ppp.secret
An authorisation file for each system.

/etc/ppp/ppp.linkup
A file to check when **ppp** establishes a network level connection.

/etc/ppp/ppp.linkdown
A file to check when **ppp** closes a network level connection.

/var/log/ppp.log
Logging and debugging information file. Note, this name is specified in */etc/syslog.conf*. See syslog.conf(5) for further details.

*/var/spool/lock/LCK..**

tty port locking file. Refer to uucplock(3) for further details.

/var/run/tunN.pid
The process id (pid) of the **ppp** program connected to the tunN device, where 'N' is the number of the device.

/var/run/ttyXX.if
The tun interface used by this port. Again, this file is only created in **-background**, **-auto** and **-ddial** modes.

/etc/services
Get port number if port number is using service name.

/var/run/ppp-authname-class-value
In multi-link mode, local domain sockets are created using the peer authentication name ('authname'), the peer endpoint discriminator class ('class') and the peer endpoint discriminator value ('value'). As the endpoint discriminator value may be a binary value, it is turned to HEX to determine the actual file name.

This socket is used to pass links between different instances of **ppp**.

SEE ALSO

at(1), ftp(1), gzip(1), hostname(1), login(1), tcpdump(1), telnet(1), kldload(2), pipe(2), socketpair(2), libalias(3), libradius(3), syslog(3), uucplock(3), netgraph(4), ng_pppoe(4), crontab(5), group(5), passwd(5), protocols(5), radius.conf(5), resolv.conf(5), syslog.conf(5), adduser(8), chat(8), getty(8), inetd(8), init(8), ping(8), pppctl(8), pppoed(8), route(8), sshd(8), syslogd(8), traceroute(8), vipw(8)

HISTORY

This program was originally written by Toshiharu OHNO <*tony-o@iij.ad.jp*>, and was submitted to FreeBSD 2.0.5 by Atsushi Murai <*amurai@spec.co.jp*>.

It was substantially modified during 1997 by Brian Somers <*brian@Awfulhak.org*>, and was ported to OpenBSD in November that year (just after the 2.2 release).

Most of the code was rewritten by Brian Somers in early 1998 when multi-link ppp support was added.