`term` HOWTO

Bill Reynolds `bill@goshawk.lanl.gov`, Patrick Reijnen `patrickr@cs.kun.nl`

February 5, 1995

Contents

1 Legal Information

1.1 Copyright statement

This document may be distributed freely as a whole in any form and free of charge. Parts of this document may be distributed, provided that this copyright message is included and the reader is informed that this is not the full HOWTO document. Furthermore, there is to be a pointer as to where the full document can be obtained. Specifically, it may be included in commercial distributions, without prior consent. However, I would like to be informed of such usage.

This HOWTO may be translated into any language, whatsoever, provided that you leave this copyright statement and the disclaimer intact, and that a notice is appended stating who translated the document.

1.2 DISCLAIMER

While I have tried to include the most correct and up-to-date information available, I cannot guarantee that usage of the information in this document does not result in loss of data. I provide NO WARRANTY about the information in this HOWTO and I cannot be made liable for any consequences for any damage resulting from using information in this HOWTO.

2 Introduction

2.1 This Document

This HOWTO attempts to clear up some of the confusion of using `term`, Michael O'Reilly's remarkable program that allows you to multiplex your serial line and set up a network connection. By and large, the documents that come with `term` are quite good, and this HOWTO is not intended to replace them. The intention of this document is to give some background on how `term` works and detail the steps in getting some of the more common networking services working under `term`. It should be emphasized that this document does not cover everything there is to know about `term`. After reading it, the `term` manual pages should be read, since they include information not contained here.

2.2 What is `term`?

`term` is a program, written by Michael O'Reilly (michael@iinet.com.au), that is run over a serial line to allow multiple connections to operate concurrently - *i.e.* you may be down-load a file via your modem while working on a (different) remote system via the same modem connection. `term` can also be used to open up X client windows over a serial connection. Through the `tredir` utility and the `tudpredir` utility `term` can provide almost all of the "traditional" TCP/IP and UDP network services: mail, news, ftp, telnet, xarchie, etc. In a sense, `term` is very much like other serial protocols such as SLIP or PPP. `term`'s advantage is that it can be run entirely from user space, requiring no support from system or network administrators.

Unlike SLIP or PPP, your machine does not have its own IP address. All incoming traffic must be addressed to your remote host, and it will be redirected to your local computer by `term`.

3 How `term` works

Before experimenting with `term` it is strongly advised to first read this complete chapter and the INSTALLATION file provided with the package. Also take a look at the manual pages `linecheck`, `(term)test` and `term`. This will help you to work easier and faster.

3.1 Nomenclature

I assume you are dialling a system through some sort of terminal server. I use the terms "local" and "remote" to refer to the home and network connected systems respectively (unless I use them to mean something else :-).

`term` provides the local machine, which has no network connection, but is connected, via a serial line, to a remote machine which is in turn connected to a network with network services. Let us look at how a machine with a "traditional" network connection provides these services. First the user invokes a program, like `ftp` or `telnet`, that requests a network service. What these programs do is make a system call requesting network services. The operating system then obtains these services via its network interface (*e.g.* it sends and receives packets over the ethernet). SLIP and PPP do exactly this, by converting your modem line into a network interface, which is in principle no different from an ethernet. The downside of this is that these protocols make the modem-connected machine part of the network, just like any other machine. This implies all the administrative burdens associated with being a network node (more actually, since the modem link must also be administered).

In the absence of a network connection like SLIP or PPP, what does one typically do? Well, you dial your network connected machine, read your mail, your news etc; if you need a file, you first transfer it to the remote machine and then download it to your local machine using `kermit` or some other communication program. This is a bit of a pain, especially since you can only really do one thing at a time that uses your modem link. The idea behind `term` is basically to automate and multiplex this process. `term` is invoked on both the local and remote machines, and the two processes communicate with one another over the modem line. When you need a network service, you make a request to the local `term` daemon, which forwards the request to the `term` daemon on the remote, network-connected, machine. The result is then returned over the modem line.

To be more concrete, say you want to retrieve a file by `ftp`. First you need a version of `ftp` that can speak to `term`. You invoke this `termftp` as you do a regular `ftp`, say `termftp nethost.gov`, but this special version makes its network request to the local `term` daemon instead of the kernel. The local `term` forwards this request, over the modem line, to the remote `term`, which opens an ftp connection to `nethost.gov`, and transmits the data back over the modem link. `term` is smart enough to have many different things going on at once; so you can have several different network sessions using the same modem link, *e.g.* you can be logged into another distant host via `termtelnet` while the `termftp` transfer is going on.

If this is too abstract (or unclear) do not worry; the important piece of information to get out of this section is that there are *two* copies of `term` running, one on each end of the modem link.

4 Setting Things Up

4.1 What has to be available

Before you start building and using `term` you have to make sure that you have built TCP/IP support into the kernel. Furthermore, make sure that the TCP/IP loopback interface is activated. When this is the case you can go on with the rest of this section.

4.2 Explanation of concepts

In newer `term` version two new concepts have entered `term`. These two concepts will be explained in the next two subsections.

4.2.1 Sharing

Starting with version 1.16 the concept of `sharing` the `term` connection with other users has entered `term`. This means that when you enable shared features, other people can use the same `term` connection you are using, i.e. when you are working on your remote machine via your `term` connection (say, from your local machine you used trsh to get in) another person on your local machine can use the same `term` connection at the same time to ftp a file to his login on your local machine from an ftp site somewhere in the world.

When you disable shared features (i.e. you execute `term` in private mode) you and only you (we do not count root :-) can use the `term` connection.

Of course, you only need to install shared `term` at the end at which you want to allow people to use the same `term` connection you are using. So, if other people have a login on your local machine and they want to use it from somewhere on your remote network you enable shared features on the remote end of your `term` connection. In this way all these people can login on your machine at the same time sharing the same `term` connection with each other and with you. (NOTE: the first example needed shared features to be enabled at the local end of the `term` connection).

NOTE for installation as root: When you install `term` as root you have to create a `term` group first (before compilation) with no member by adding the following line in /etc/group:

```
term::16:root
```

or any other unused `GID` than 16 when 16 is already in use.

After compilation and installation make `term` and its clients SGID term:

```
chgrp term <term_client>
chmod g+s <term_client>
```

Also any other program that you make term-aware must be made SGID term.

4.2.2 Full term networking

Starting with `term` version 2.0.0 the statement `full term networking` is used. When your only connection with the outside world is a `term` connection, you have a full term network and you should build term with full term networking. In this case in the shared directory a file called `termnet` is placed. This tells `term` that your only connection to the outside world is via `term`.

When you also have some other type of network connection beside your `term` connection term-aware programs first try to fulfil their job using this network connection. When this fails `term` is invoked and it is tried to fulfil the job via the `term` connection. To make this more clear now an example is given in which `term-aware telnet` is used. This telnet should work both with and without `term`.

```
telnet localhost
```

does not use `term` to connect, but

```
telnet zeus.cs.kun.nl
```

will use `term` only if you do not have some other type of network connection.

`full term networking` also means to lie about the host name, and say it is the remote host instead. Furthermore, it causes bind(0) to always act on the remote host. In essence it makes many programs unusable when they are not going through `term`, while `term` is running. Unfortunately, most UDP programs and daemons will not work with `term` without these nasty tricks.

4.3 Build `term`

When you are lucky, this should just involve a `make`. Most probably however, you need to do more. Due to new features in newer versions of `term` it is now a bit more complicated to create your `term` binary. A couple of ways can be followed to obtain your binary.

To cover all these ways `term` can be built this section will be split into three parts:

1. Build `term` up to version 1.15
2. Build `term`, versions 1.16 up to 1.19
3. Build `term`, versions 2.0.0 and higher

4.3.1 Build `term` up to version 1.15

For these versions of `term` building should invoke no more than the commands

```
make DO=install OS-type
make installman
```

You will find `term`, its clients and the manual pages nicely built and installed and ready for use after this.

Furthermore, you need to create a directory `$HOME/term`. This directory `term` will use to look for its termrc file.

The only thing you may want to do is change some of the paths in the `Makefile` or change some of the compiler flags.

4.3.2 Build `term`, versions 1.16 up to 1.19

To build these versions of `term` you can now choose one of the following ways:

1. As an ordinary user, build `term` in private mode
2. As an ordinary user, build `term` in shared mode
3. As root, build `term` in private mode
4. As root, build `term` in shared mode

Below, it will be explained how to enable/disable shared features during the compilation of `term`.

1. You are an ordinary user (no root access) and you do NOT want to SHARE the `term` connection with other users.

   As a user who does not want to share the `term` connection with other users you should do the following to build `term`:

   ```
   make DO=install OS-type
   make installman
   ```

   After this `term`, its clients and the manual pages are built and installed.

   Furthermore, you need to create a directory `$HOME/.term`. This is the directory in which `term` will look for its termrc file.

2. You are an ordinary user (no root access) and you want to SHARE the `term` connection with other users.

   As a user who wants to share the `term` connection you should do the following:

   ```
   make DO=installshare USERSHARE=$HOME/term OS-type
   make installman
   ```

After this `term`, its clients and the manual pages are built and installed.

Furthermore, you will have a directory `$HOME/term` (default) with permissions `drwxrwxr-x`. In this directory you will find at least the socket used by `term` for its connection (tmp/private/socket=).

3. You are `root` and you do NOT want to SHARE the `term` connection with other users.

   As root who does not want the `term` connection to be shareable you should do the following to build `term`:

   ```
   make DO=install OS-type
   make installman
   ```

   After this `term`, its clients and the manual pages are built and installed.

   Furthermore, you now have a directory called `/usr/local/lib/term` (default) with permissions `drwxr-xr-x`. In this directory you will at least find the socket used by `term` for its connection (tmp/private/socket=).

4. You are `root` and want to SHARE the `term` connection.

   First, make sure you have read the section about 'sharing' above.

   As root who wants to share the `term` connection you should do the following:

   ```
   make DO=installshare OS-type
   make installman
   ```

   After this `term`, its clients and the manual pages are built and installed.

   Furthermore, you now have a directory called `/usr/local/lib/term` (default) owned by group `term` and with permissions `drwxrwxr-x`. In this directory you will at least find the socket used by `term` for its connection (tmp/private/socket=).

4.3.3 Build `term`, versions 2.0.0 and higher

First, make sure you have read the section about 'full term networking' above.

For `term` versions 2.0.0 and higher there are many ways to build the `term` binary and the clients. All of these can be done both by root and by ordinary user:

1. Build `term` in private mode without full `term` networking
2. Build `term` in private mode with full `term` networking
3. Build `term` in shared mode without full `term` networking
4. Build `term` in shared mode with full `term` networking

In these versions of `term` a new way for compilation has entered `term` using the script `configure`. When `configure` is run it checks on what operating system you are trying to install `term`, whether the source directory is available or not, and if any runtime options are set. According to the things found `configure` then creates a `Makefile` using Makefile.in which is provided in the `term` package.

Two of the more important options to configure are `--root` and `--user` which state whether `term` will be installed by root or an ordinary user. Other options can be used to install `term` the way you want (non-standard paths for example).

1. Build term in private mode without full `term` networking.

   To build `term` in this way you need to execute the following commands (both for root and ordinary user):

   ```
   ./configure --root  OR --user
   make install installman
   ```

   This builds the binaries and installs these binaries and the manual pages.

2. Build term in private mode with full `term` networking.

   To build `term` in this way you need to execute the following commands (both for root and ordinary user):

   ```
   ./configure --root  OR --user
   make installnet installman
   ```

   This builds the binaries and installs these binaries and the manual pages.

3. Build term in shared mode without full `term` networking.

   To build `term` in this way you need to execute the following commands (both for root and ordinary user):

   ```
   ./configure --root  OR --user
   make share installman
   ```

   This builds the binaries and installs these binaries and the manual pages.

4. Build term in shared mode with full `term` networking.

   To build `term` in this way you need to execute the following commands (both for root and ordinary user):

   ```
   ./configure --root  OR --user
   make share installnet installman
   ```

   This builds the binaries and installs these binaries and the manual pages.

4.4 client.a, libtermnet.a, libtermnet.sa, libtermnet.so

With `term` a library with functions for `term` clients is provided.

Up to version 1.16 this library was called `client.a`. During compilation of `term` this library was built and then used during the compilation of the `term` clients. It was not installed in another directory.

Starting with version 1.16 the name of the library is changed to `libtermnet.a`. Up to version 1.19 this library is created in the `term` directory and then used during compilation of the `term` clients. It is not installed in another directory.

Starting with version 2.0.0, beside libtermnet.a also `libtermnet.so` and `libtermnet.sa` (shared library and exported initialized library data) are created during compilation of the `term` package. During the installation of all the parts of the package also these three library files are installed in the directory `/usr/local/lib` (default). Then a link is made from libtermnet.so.2 to libtermnet.so.2.x.x. Finally, `ldconfig` is run to create the necessary links and cache (for use by the run-time linker, ld.so) to the most recent shared libraries found in the directories specified on the command line, in the file /etc/ld.so.conf, and in the trusted directories (/usr/lib and /lib). If the installation is done correctly the three library files can now be used by `term` clients which are built with dynamic instead of static libraries. Also, these libraries can now be used to port your own software to make it `term` aware (see below).

4.5 Setting environment variables

`term` knows a couple of environment variables which can be set by users. The first three of those that I will explain are

- TERMDIR
- TERMSHARE
- TERMMODE

By setting these variables you can control the way `term` is run.

For `term` versions up to 1.15 only the variable `TERMDIR` is important (these versions do not know the shared mode). For these versions TERMDIR should be set as follows:

```
setenv TERMDIR $HOME      (csh or tcsh)
export TERMDIR=$HOME      (bash)
```

Starting with version 1.16 `term` also knows the variables `TERMSHARE` and `TERMMODE`. With these variables `term` can be told to run in `private` mode or in `shared` mode. I will explain how to set the variables for private mode and shared mode.

1. Running `term` in private mode can be done setting the variables TERMDIR and TERMMODE in the following way:

   For csh or tcsh

```
setenv TERMDIR $HOME
setenv TERMMODE 0
```

For bash

```
export TERMDIR=$HOME
export TERMMODE=0
```

2. When you want to use `term` in shared mode there are two ways of setting the variables:
   (a) When term is installed as a SUID program only TERMMODE has to be set (?? README in the term package).

   ```
   setenv TERMMODE 'number'    (csh or tcsh)
   export TERMMODE='number'    (bash)
   ```

   For `'number'` you have to enter 1 when you are using one of the `term` versions 1.16 upto 1.19 (README.share in the package) and 2 when you are using `term` version 2.0.0 or higher (README.security in the package).

   (b) When `term` is installed as a SGID program the variables have to be set in the following way:
   For csh or tcsh

   ```
   setenv TERMMODE 1
   setenv TERMDIR /usr/local/lib/term
   setenv TERMSHARE $TERMDIR
   ```

   For bash

   ```
   export TERMMODE=1
   export TERMDIR=/usr/local/lib/term
   export TERMSHARE=$TERMDIR
   ```

   Setting the variables in this way makes it possible to start old clients (clients linked to an older version of `client.a`) in shared mode.

Starting with version 2.0.0 `term` also knows the variable `TERMSERVER`. You need to set this variable when you have multiple modems and you have more than one connection at a time. To specify which connection to use, you must start `term` with a server name:

```
nohup term -v /dev/modem1 Connection1 &
nohup term -v /dev/modem2 Connection2 &
```

Users should then set the variable TERMSERVER to the connection name they want to use:

```
setenv TERMSERVER Connection1   (csh or tcsh)
export TERMSERVER=Connection2   (bash)
```

4.6 Test `term`

Do a `make test` (or `make termtest` for newer versions of `term`) to build `term`'s test daemon. (term)test works by running two copies of `term` on your system, a "local" and a "remote" copy. Both of these will read your `termrc`; so you can adjust their behaviour. Now execute `(term)test)`. You should now be able to do a `trsh` and a `tupload` (try `tupload ./term /usr/tmp` - you should get a copy of the `term` binary in `/usr/tmp`). The local `term`'s output should show up in `local.log`, the remote's one in `remote.log.` You can start `term` up with a `-d255` flag to enable debugging output to be written to these files, or enable debugging in your `termrc`.

NOTE: Run `test` as `./test` so as to avoid your system's `test`.

4.7 Make a Transparent Link

Presumably, you can establish a modem connection between your local and remote hosts. Typically, you are dialling into some kind of terminal server and connecting to your remote host from there. You are also using some kind of terminal software, such as `kermit` or `seyon` to talk to your modem (the examples in this document will use `kermit,` since that is what its author uses). If you are having trouble with your modem, or your terminal software, take a look at the Serial-HOWTO; that should help you out.

Having established your link, you want to make it as transparent as possible. Check the commands on the terminal server (help or ? will usually get you started). Go for the 8 bit options whenever possible. This may mean changing the way you log in to a system, *e.g.* if the server uses rlogin, you may have to use it and give it the -8 flag to make it transparent. Especially watch out for xon/xoff flow control. You do not want that. Try to enable rts/cts (hardware) flow control. You may need to check your modem docs to learn how to configure it to do 8-bit rts/cts communications.

4.8 Run linecheck

WARNING: In some of the documents the command line options for `linecheck` are mentioned in an incorrect order. I have checked this and found the order of options mentioned below to be the correct ones.

NOTE: starting with `term` version 2.3.0 linecheck no longer needs to have the name of a log-file on its commandline. It will write its output to the file *linecheck.log* in the directory you start linecheck in.

`linecheck` is a program that is supplied with `term`. It checks the transparency of a link, providing configuration information that `term` needs to run correctly. `linecheck` will send each of the 256 possible eight bit characters over the link and verify that each was transmitted successfully. `term` needs to be configured to deal with characters that cannot be transmitted over the link, and `linecheck` determines what characters these are. You use `linecheck` after you have established as transparent a modem link as possible. To run `linecheck,` do the following

1. On the remote system, run `linecheck linecheck.log`
2. Escape back to your local system and suspend your comm program (`^Z` under kermit) (otherwise it will steal characters from linecheck).

3. On the local system run `linecheck linecheck.log > /dev/modem < /dev/modem`

When `linecheck` is done, you will find a set of numbers at the bottom of the `linecheck.log` files. These should be escaped in the termrc at the other end of the link. For example, in my system my local `linecheck.log` said nothing and my remote `linecheck.log` said to escape 29 and 157. Therefore, my local `termrc` escapes these characters and my remote `termrc` escapes nothing. If I `escape` a character at one end, I have to also `ignore` it at the other; so, in this example, I shall have to ignore 29 and 157 in my remote system.

If linecheck hangs, try using

```
linecheck linecheck.log 17 19
```

on the remote system and

```
linecheck linecheck.log 17 19 > /dev/modem < /dev/modem
```

on the local system. This will escape your xon/xoff (flow control) characters, which will hang your line if you have got software flow control. If this solves the hanging problems, you will want to `escape` /`ignore` 17/19 in both `termrc`'s. If your terminal server has other characters that will shut it down, try running linecheck with those characters escaped, as above. You can spot these characters if `linecheck` hangs. If this is the case, kill it, then look in the logfiles. The last characters transmitted are likely to be the culprits. Try it again with these characters escaped.

In summary, my local termrc has the lines

```
escape 29
escape 157
```

and my remote termrc has the lines

```
ignore 29
ignore 157
```

since my *remote* `linecheck.log` said to escape 29 and 157.

4.9 Try Running `term`

Log into the remote system, making the link as transparent as possible (if you have not already done so). Fire up `term` at the remote end. I use the following:

```
exec term -r -l $HOME/tlog -s 38400 -c off -w 10 -t 150.
```

Let us run down each option one by one (note that I could just as easily have put these options in my termrc. I did it this way because it saves editing a file while getting `term` set up).

`exec` means to destroy your current shell, running the given program in its place. I `exec` things because I do not intend to use my login shell again; so it is just wasting memory. If you are debugging the link and can reliably kill the remote `term,` you might not want to do an exec.

The `-r` option is needed at exactly one end. `term` will then see this end as the remote end of the connection (Note that `term`'s remote end can thus be your local machine). If you do not use this option at one end `term` clients will spontaneously crash.

`-l $HOME/tlog.` This logs errors to the file `tlog` in my home directory. Very useful for debugging. No reason not to do this.

`-s 38400` : I have got a 14400 baud modem, with compression. For optimal compression ratios, I want to be able to push bits down the pipe as fast as possible. For a slower modem, you should use something lower. Note that if you have a slower machine with 16450 uart on your serial port, high baudrates can cause data loss by overloading the chip on your serial port. `term` will recover from this, but if you see a lot of error messages in your log file, (or get overrun warnings from linux kernel versions 0.99pl15 and up) you again might want to lower this number.

`-c off` : This turns data compression off. I have got a compressing modem, and I do not want to compress things twice.

`-w 10 -t 150` : Again, these are options to optimize my fast modem link. I have set my window to 10 and my timeout to 150. This is according to the recommendation in the `term_setup` man page.

Escape back to your local machine and suspend your comm program (`^Z` under kermit). You do not want it running while `term` is running, because it will fight with `term` over the serial port. If you can convince your modem to not hang up when you exit your comm program (when it toggles DTR), you could just exit the program at this point.

Now run `term` locally. I use:

```
term -c off -l $HOME/tlog -s 38400 -w 10 -t 150 < /dev/modem > /dev/modem &
```

I need to tell `term` where the modem is; so I point both standard input and output at `/dev/modem` (that is what the `<` and `>` do). I also run it in the background; so I can use this screen for something else if I want to.

`term` should work now :-). Try a `trsh,` and see what happens. If you hang, or your link seems slow, take a look at your `tlog` at each end. Are you getting timeouts or error messages? If so, then you have configured something incorrectly. Try again (after you have finished reading this :-). Note that the connection will not seem blazingly fast, especially if you are using compression - it will be a little jumpy. The real speed comes in during file transfer and the like.

4.10 Terminate your `term` connection

Most certainly, after you have done a lot of work using `term`, you want to finish your work and bring your `term` connection down. For this to be realized there are four ways:

1. Kill the `term` programs at both sides of the connection. This is the least recommended way of terminating your connection.

2. A better way is to execute the following command locally:

   ```
   echo '00000' > /dev/modem
   ```

   This will nicely terminate your `term` connection. It will work for all version of `term`. Make sure that the sequence of zeros contains at least `five` zeros.

3. In the termrc of `term` versions 2.0.0 and higher you can now enter a statement called `terminate '<some string>'`. This sets a string that will cause `term` to exit ('00000' by default). It must be at least 5 characters long, to avoid accidently terminations.

4. Starting with version 1.14 there is the program `tshutdown` (actually for version 1.14 it is available as a patch, for newer versions it is in the package). Executing tshutdown nicely terminates your `term` connection.

4.11 Optimizing your connection

Once you have got `term` running, you might want to try to get things optimized. A good way to measure the speed of your link is to run `tmon` in one window while up/downloading a file in another. Try both (big) text files and compressed files; the plain text should go a factor of two-ish faster than the compressed files. The parameters you want to fiddle with are baudrate (`-s`), compression (`-c`), windows (`-w`), timeout (`-t`) and retrain (`-A`).

Watch out with the retrain parameter. With `term` version 1.19 I got a performance decrease of 80% to 90% compared to running `term` without the retrain parameter. It is not clear if this is a bug in `term` version 1.19 and if this problem exists only with `term` version 1.19.

`Baudrate`: the maximum number of bits per second `term` will try to send over the serial link. `term` will avoid sending characters at a higher datarate than this. The default is to use the speed of your computer's serial port, but be warned that this may be too high if your modem runs at a lower rate over the phone line. The baudrate option is intended for systems that buffer output to the modem. During setup and tuning it is better to use a small baudrate rather than one which is too large. For high speed links (> 38400), making it unlimited is probably advantageous. This is achieved by using the value 'off'. `term` will then rely solely on your kernel to do flow control.

`Compression`: you want this on if you do not have a compressing modem. If you do have such a modem, turn compression off, otherwise you will be compressing things twice, which typically *increases* the amount of data transmitted. Compressing modems are those that use the MNP-5 or V42.bis protocols. Check your modem documentation and the message when your modem connects.

`Windows`: this is the number of chunks of data, or packets, that `term` will let go over the line before it gets an acknowledgment (or ack) from the remote `term.` For fast modems, increasing this can be a win; for slower links this can overwhelm the remote end.

`Timeout`: the time `term` will wait for an ack. If you have increased windows, and you are getting timeouts in your logfile, try increasing this.

For 14400/V42.bis, I use `-c off -w 10 -t 150.` I get around 1700 cps on compressed files and 3500 cps on ascii files using tupload.

4.12 Troubleshooting

In this section some thoughts are given about what to check when you have problems executing `term` or one of its clients.

- Did you clean up the `term` directory structure? With newer versions of `term` the structure of the directory tree under `/usr/local/lib/term` has changed a couple of times. If you are not aware of this, it can cause all kinds of error messages. The best thing to do is to delete the directory tree under /usr/local/lib/term (save your `termrc`) and then install your new `term` version. This way, you avoid the struggle with a messed up directory tree.
- Did you remove old sockets? When you update your `term` version remove all the sockets (called socket=) created by `term`. No doing this can cause strange problems. To find out what socket `term` is listening to you can use the "netstat" program.
- `term` does not compile correct on sunOS 4.1.3? You have configured `term` with './configure –user'. During compilation you are getting a assembler error on a unknown '-k' flag. The reason of this error is unknown. The solution to this error is to configure `term` with static libraries. So, you have to do './configure –user –static' and then continue with the compilation process the way you normally do. Now `term` should compile correct.
- Are you running the right binary? `term` has been updated quite a lot, and many systems have different versions of the programs floating around. Make sure you are using the right version. Note that this applies to linecheck too. You can use bash's `type -a` or the `whereis` command to find which program is being run. `term` versions after 1.11 should print out their version number when they start up. (Although version 1.14 claims to be 1.12. Sigh.)
- Do you have the right `termrc` in the right place? Depending on the version of `term` you are running and the way you installed `term` (being root or user) this file has to be in one of the following directories:

  ```
  $HOME/.term/termrc
  $HOME/.term/termrc.<server>
  $HOME/term/termrc
  $HOME/term/termrc.<server>
  /usr/local/lib/term/termrc
  /usr/local/lib/term/termrc.<server>
  /etc/termrc
  /etc/termrc.<server>
  ```

  Some systems have pre-installed termrc's; make sure they are gone before you set things up. If you are running things as root, lookout for `/.term`. `term` creates files (sockets actually) while it is running; so it has its own directory, `~/.term`, where the file `termrc` goes (note, there is *no* leading dot in `termrc`!).
- Does `term` find its termrc file? when you start up `term` at both sides, you should see messages like the one below:

```
Term version: 2.2.9
Reading file:  /usr/local/lib/term/termrc
Using shared mode.
```

When the second line is missing `term` cannot find its termrc file and you know that something is gone wrong during the installation (unless you are not using a termrc file and enter all the options to the commandline :-). Check the place and the permissions of the termrc file on the site `term` cannot find its termrc file.

- Is you term or .term directory mounted with NFS? If your term or .term directory is mounted with NFS you need to set the flag -DTERM_NFS_DIR in the CFLAGS line of the Makefile. Alas, for the author this flag causes a compile error when `term` 1.19 is compiled on a machine running sunOS 4.*.
- Are all files and directories owned by the correct user and group and do they have the correct permissions? This should be no problems as these permissions are set during the installation phase. However, when you port your own programs to `term` you must be aware of this. Also when you change the mode `term` is working in (i.e. from private mode to shared mode) file and directory ownerships and permissions have to be adapted.
- You are getting the error *gethostbyname: <hostname>: Non-authoritative 'host not found', or, server failed*

  To solve this you have to check the following things:

  1. Is the file `/etc/hosts` configured correctly? <hostname> is not the name of your host (old SLS releases and some old and new Slackware releases are shipped with hostname 'darkstar' for example). Change this in this file. It must at least contain a line like below (the format is described above it):

     ```
     # Local Hosts Format:
     #  IP_NUMBER           HOSTNAME        ALIASES
     #
     # Here is the name of your host, first, followed by any aliases
     127.0.0.1       localhost       linuxpc.domain  linuxpc
     ```

  2. Are your `/etc/rc*` and `/etc/resolv.conf` files world readable (chmod ugo+r)?
  3. Last, make sure that you have installed the TCP/IP loopback-interface on your machine.
- Are you getting all kinds of 'timed out' messages in your `term` log files? This means that your `term` connection is not optimized. A small number of these messages every now and then is not a problem. These are most certainly due to temporal factors influencing the physical connection between your local and remote hosts.

  When you get a lot of these messages all the time your connection will slow down considerably. You have to fiddle with the parameters mentioned in the section 'Optimizing your connection' above. Alas, this part of the installation is a process of trial and error. No hard rules can be given about the numbers to be presented to the various parameters as many factors have an influence on the connection. These factors differ between the connections and even in time.

- Normal FTP with redirected ports does not work for you? Alas, it is a known problem that redirection of the ports needed by FTP (20 and 21) does not give you a working FTP. The only solution is to get a `term`-aware ftp or ncftp version. Alas, also some `term`-aware ftp versions seem not to work.

5 `term` clients

`term` provides several default clients. They include `trsh`, `tmon`, `tupload`, `tredir`, `txconn` and in newer versions `trdate`, `trdated`. Furthermore, starting with version 2.0.0 `tudpredir` is available and from version 2.1.0 `tdownload` is available. This section will deal with `trsh`, `tmon`, `tupload`, `tdownload`, `trdate` and `trdated`. The others each have their own section. No `term` client will work until you have established a `term` link.

`tmon` is a simple utility to monitor the statistics of your link. It prints a time histogram of characters transmitted and received. It is invoked simply as `tmon.` Since around version 1.11, `tmon` has had a bug that causes some information to be garbled (??).

`trsh` is similar to `rsh.` Without arguments, it spawns an interactive shell on the remote system (*i.e.* it logs you in). `trsh` is one of the primary means of accessing the remote end of the link via `term`. If given an argument, `trsh` executes that argument as a command on the remote system. For example `trsh ls` would give you a listing of the files in your home directory on the remote system.

`tupload` will transfer a file, given as its first argument, from local to remote. By default, the files will be put in the same directory that you invoked `term` from at the other side. To put files in another directory, give their names as a second argument to `tupload`. For example, if I want to put a copy of the file `term114.tar.gz` in `/usr/tmp` on the remote system, I would type `tupload term114.tar.gz /usr/tmp`.

When you use tupload you can use wildcards like in 'tupload a.*'. The shell expands the wildcard and tupload is called as 'tupload a.1 a.2 ......'.

`tdownload` will transfer a file, given as its first argument, from remote to local. By default, the files will be put in the same directory that you invoked `term` from at the local side. To put files in another directory, give their names as a second argument to `tdownload`. For example, if I want to put a copy of the file `term114.tar.gz` in `/usr/tmp` on the local system, I would type `tdownload term114.tar.gz /usr/tmp`.

When you use tdownload you *cannot* use wildcards like in 'tdownload a.*'. Reason for this is that the remote directory is not available to your local shell when you use tdownload; so your local shell cannot expand the wildcards.

`trdate` is a time setting utility. It reads the time on the remote machine and sets the local clock to the remote time. It must be run as root.

`trdated` is the daemon version of trdate. When it is started in `rc.local` it is run as daemon in which case it updates the time every 5 minutes (default). Even when there is no `term` connection, this daemon will start up when set in the rc.local. Once a `term` connection is created it starts updating the time.

6 X and `term`

`term` allows users to open up X windows on the local machine from clients that are running on a machine on the network. This is done by using the `txconn` client. `txconn` is executed on the remote, network-connected machine; it is invoked simply as `txconn`. It goes into the background and returns a number on the standard output; this number is the display number that clients should use to access the X server on the local machine. An example should make this clear. I am logged in, via `trsh`, to my remote `term` host, named `foo`. On `foo`, I do the following

```
foo$ txconn
Xconn bound to screen 10
:10
foo$
```

Now, on any host that I wish to run an X client on, that is to display on my local machine's X server, I do

```
setenv DISPLAY foo:10
```

(for `bash` you should use `export DISPLAY=foo:10`). In some cases it can furthermore be necessary to do a `xhost + foo` on your local machine. Now when I start the client, it will try to connect to screen 10 on machine `foo`, but `txconn` is listening to this screen, and will forward all X protocol packets via `term` to the X server on the local host - *i.e.* the window will open up on your local machine.

It is possible to go the other way - run a client on your local machine and have it open up a window on a remote machine on the network; however we will defer explaining this until after we have discussed `tredir`.

`txconn` is not terribly secure; anyone can connect to your local server via `term` and perform all sorts of mischief. If you are worried about this sort of thing, it might be a good idea to consider using `xauth` to authorize your connections. Consult the `xauth` man page.

The X protocol is not very efficient; it wastes some bandwidth. This is usually not a problem over an ethernet, but can be murder over a modem. X11R6 is supposed to introduce a low bandwidth version of the X protocol, LBX. If however you are using X11R5 you can use a utility named `sxpc` which compresses the X protocol, improving response over serial lines. `sxpc` includes a write-up on how to get it working with `term`, and is recommended. The `sxpc` package also explains how to use `xauth`; so it is doubly recommended.

7 `tredir`

`tredir` is one of `term`'s most powerful utilities, allowing most important network services to be performed over a `term` link. Before we explain how to use `tredir`, it is necessary to give some background on network services. We have talked about network services before, but we have not said exactly what they are. Services are just that - services that are provided by the network. Examples of services include `telnet`, which provides logins between machines, the File Transfer Protocol, `ftp`, which transfers files between machines, and smtp, the Simple Mail Transfer Protocol, which is used whenever you send electronic mail. Each network service has a *port number* associated with it. The mapping of port numbers to services is given in the file `/etc/services`. This file should be the same on all internet-connected machines.

How are these services invoked? Each networked machine runs a daemon called `inetd`, which listens for attempts to connect to the network ports. These requests can come from either the network or the local machine. A network service is obtained by connecting to a particular `inetd` port. When a network request is made, `inetd` knows exactly which service is involved by the port number the request is made on. If `inetd` is configured to do so, it provides the relevant service to the requesting connection. `inetd`'s configuration is given by the file `/etc/inetd.conf`, which has a list of the services that `inetd` provides. For more information, see the man pages for `inetd` and `inetd.conf`.

You can communicate directly with network services by using `telnet` (*n.b.* not `termtelnet`). For example, to talk to the `sendmail`, (or smtp) daemon on machine `machine_name`, you can do a `telnet machine_name smtp,` or `telnet machine_name 25`, (since 25 is the number assigned to smtp in `/etc/services`). You should get a polite greeting from the daemon on the remote machine. This is a very useful trick for debugging network problems and checking ports redirected with `tredir` (see below).

`tredir` works very much like `inetd`. It runs in the background as a daemon, listening to the network ports, waiting for a request. When a request for a service is made, instead of providing that service, as `inetd` does, `tredir` forwards the request over the `term` link to the remote `term`, which makes the request over the network, returning the result back over the link to the local client. `tredir` can forward the request to any machine on the network, but by default sends it to the machine at the other end of the `term` link. `tredir` "redirects" TCP (Transmission Control Protocol) network services over the `term` link.

The common command format of `tredir` is:

```
tredir [this_computer:]port [that_computer:]port
```

An example should make this clear. Let us redirect a local port to the telnet port on the remote machine. To do this we would do `tredir 2023 23`. Now, anyone who connects to port 2023 on the local machine will be redirected to port 23 (telnet) on the remote machine. Here is an example session; the local machine is `mymachine.modem.home` and the remote machine is `netsun`.

```
$ tredir 2023 23
Redirecting 2023 to 23
$ telnet localhost 2023
Trying 127.0.0.1...
Connected to mymachine.modem.home
Escape character is '^]'.


SunOS UNIX (netsun)
login:
```

This example is actually quite useful. If I were instead to do the tredir on `netsun`. I could then telnet in to `mymachine` from the network simply by connecting to the redirected port on the networked machine (using telnet) - *i.e.* `telnet netsun 2023`.

The general principle in using `tredir` is to redirect the desired service to a machine on the network. Our next example will allow us to read news on the local machine over our `term` link from a news server on the network. News is provided by the nntp service, port number 119. All decent news readers allow you to specify what port number they will use, either via a configuration file or an environment variable. Let us specify this local port to be 2119. Now, let us say that our news server is `news.domain.org`. We will redirect port 2119 to port 119 on `news.domain.org`; we will then tell our news reading software that the nntp server is located at port 2119 on the local host. Since this will depend on the news reader that you use, I will just test the link with `telnet` instead of firing up a news reader:

```
$ tredir 2119 news.domain.org:119
Redirecting 2119 to news.domain.org:119
$ telnet localhost 2119
Trying 127.0.0.1...
Connected to mymachine.modem.home.
Escape character is '^]'.
200 news.domain.org InterNetNews NNRP server INN 1.4 07-Dec-41 ready
(posting ok).
```

If you can get this far, all you have to do is configure your news reader to be able to read news via `term`. (*n.b.,* if you read news like this, be sure that in all your posts you set a Reply-To: header to an network email address that you can be reached at, otherwise people who want to get in touch with you will be sending mail to whatever (wrong) data your newsreader puts in the From: header).

7.1 `tredir` can bite!

The astute reader, after reading the last example will be wondering why port 2119 was redirected to port 119 - since newsreaders default to port 119, why could I not do a `tredir 119 news.domain.org:119` and skip the newsreader configuration? The answer is that all ports numbered less than 1024 are "reserved ports", and only the superuser can listen to them. If one is willing to take a security risk and make `tredir` an suid program, or run `tredir` as root, then one can redirect reserved ports and avoid the hassle of renaming services.

Another problem with using reserved ports is that `inetd` is often already listening to these ports, and only one program at a time can listen to a port. In order to use such a port, you must change `inetd.conf` so that `inetd` no longer listens to the port you want to redirect. This is most easily done by commenting out the line with the offending service by putting a `#` character at the beginning of the line. The superuser must then send `inetd` a HUP signal (kill -1 inetd-pid) to get it to reread its configuration.

7.2 Stupid `tredir` tricks

In this section we will describe some of the more common uses for `tredir.` We have already described how to redirect nntp and telnet services; here we will give some more complicated examples.

7.2.1 X windows

In a previous section, we described how to get an X client running on the network to open a window on your home machine using `txconn`. The same technique could be used on your home machine to display a client on the machine at the remote end of your `term` link. But how does one display an X client on a network machine that is not the remote end? The answer lies in knowing that X uses a particular network service just like the other programs we have been discussing. An X server listens for a network request on a port whose number is given by the formula $port = 6000 + displaynumber$, *e.g.* an X server managing screen 0 on a machine would listen to port 6000, if it were managing screen 2, it would listen to port 6002. When you set your DISPLAY environment variable to `xmachine:n`, your X clients will try to connect to port $6000 + n$ on `xmachine`.

We can use this to trick X clients on your local machine to open up windows on remote displays. Let us say I want to open up an `xterm`, running on my local machine, on display 0 of machine `xmachine`, which is running some place on the network. I first pick a local display number, say 2 (do not use 0, since that is what your local X server will be using). I will map this display to display 0 on `xmachine`. In terms of ports, this means I want to redirect the local port 6002 to the remote port 6000. I do the following

```
$ tredir 6002 xmachine:6000
$ setenv DISPLAY localhost:2
$ xterm
```

This should open up an `xterm` on machine `xmachine`. Note that I set the DISPLAY to `localhost:2`. This is because X clients will sometimes use unix domain sockets instead of internet domain sockets, at their own option, when connecting to a local display, if DISPLAY is set to `:2`. `localhost:2` says to use a tcp connection.

Note that as far as `xmachine` is concerned, the X request is coming from the machine on the remote end of your `term` link (`remotemachine`) - so if you need to authorize the connection, you should either do an `xhost + remotemachine` on `xmachine` or use `xauth` to update the `.Xauthority` file on your local machine for display number 2, using the key from `xmachine`.

Again, to speed up X connections, you can use the program `sxpc`, which includes an explanation of how to use `tredir` to establish the link and authorize it using `xauth`.

7.2.2 Mail with `term`

Well, you asked for it. Electronic mail has the justifiable reputation of being one of the most difficult things to get working right on a UNIX system. To really get `term` working correctly with mail means that you have to understand how mail works, which is beyond the scope of this document. To learn more about mail, you should consult a book on UNIX system administration and/or the comp.mail.misc FAQ, available for anonymous ftp on rtfm.mit.edu in `pub/usenet/comp.mail.misc`. There are also currently two packages available for anonymous ftp on `sunsite.unc.edu` that will help you get mail running under `term` - they are `term.mailerd+smail` by Byron A. Jeff and the `BCRMailHandlerXXX` by Bill C. Riemers.

That being said, we will give a thumbnail description of how mail works. There are two parts to getting mail running, sending messages and receiving messages. We will begin with sending messages from your local box to the network.

There are two classes of mail programs. The first is the mail user agent (MUA). MUA's help you read, compose and send messages. Examples of MUA's are `elm`, `pine`, `Mail` and `vm`. MUA's do not really do any networking; they just put the messages together - the real work of sending mail is done by the second class of mail programs, the mail transfer agents (MTA's). These are invoked by the MUA's. They take the message, decide where to send it by looking at the address, and then actually deliver it over the network.

The two most common MTA's on Linux systems are `sendmail` and `smail`. The basic idea is to get your MTA to connect to another MTA running on a machine on the net that will know what to do with your message. This is done by redirecting a local port to the smtp port on the net machine. You then have to tell you MTA to take any message it does not know what to do with, and send it out over the redirected port on your local machine to the MTA on the remote machine, which will then route your message to its correct destination.

How do we do this using `smail`? We first redirect a port to the smtp port on the network mail machine (`mailhost`):

```
tredir XXXX mailhost:25
```

here XXXX is the port number that the `smail` on the localhost will connect to (note that I have to give this port a name in my `/etc/services` to get `smail` to recognize it). `smail` has several configuration files that usually live in `/usr/local/lib/smail`. The ones we care about are `config`, `routers` and `transports`. Note that I am assuming you have already got `smail` configured correctly for local mail - delivery to files and pipes and such things. Again, consult the documentation if you have not.

In the file `config`, we put the following definition:

```
smart_path=localhost
```

localhost is the machine that `smail` connects to when it does not know what to do with a message.

In `routers` we put

```
smart_host:
driver=smarthost,
transport=termsmtp;
path = localhost
```

In `transports` we put

```
termsmtp:       driver=tcpsmtp,
        inet,
        return_path,
```

```
        remove_header="From",
        append_header="From: YOUR_NET_ADDRESS",
        -received,
        -max_addrs, -max_chars;
        service=YOUR_SMTP_SERVICE,
```

In the above, the header lines change the `From` header in all your outgoing mail to the address, `YOUR_NET_ADDRESS`, which is the network address you want mail sent to. If more than one user is going to be using your `term` link, you will have to do something more fancy, like keep a database of local user's network addresses and insert these in the `From:` headers.

The service line is the name of the local port number that you have redirected to the smtp port on the network connected machine. In my version of `smail`, I cannot just set this to a number, I have to set it to a name, like "foo", and then define "foo" in my `/etc/services` to be the number of my redirected port. If you use a suid `tredir` and just redirect the smtp port (25), you do not need to define this.

This should be enough to get you going. If you decide to use `sendmail,` the principles are the same but the details differ. Ronald Florence (ron@mlfarm.com) told me that the stock Sun `sendmail` will not send multiple queued messages over a redirected port; BSD sendmail 8.6.9 works fine. He made the following changes to `sendmail.cf` to get it working with `term`. In his case, the default sendmail port (25) is used for SMTP traffic over a local ethernet so Internet mail is forwarded to a redirected TCP port.

```
#
#Create the termsmtp mailer, which sends mail via a re-directed TCP port
#
Mtermsmtp,P=[TCP], F=mDFMuCXe, S=22, R=22, A=TCP $h PORTNUMBER
```

Here, `PORTNUMBER` is the number of the redirected port on the local machine. This should be an unused port over 2000. We next tell sendmail which machine to connect to, and set termsmtp as the default mailer.

```
#
# major relay mailer
#
DMtermsmtp
#
# major relay host: use the $M mailer to send mail to other domains
#
DR HOSTNAME
CR HOSTNAME
```

Here `HOSTNAME` is the name of your local host (does `localhost` work?). The last entry goes under Rule 0 to forward Internet mail.

```
# Pass other valid names up the ladder to our forwarder
R$*<@$*.$+>$*           $#$M    $@$R $:$1<@$2.$3>$4     user@any.domain
```

When the term connection is established to the Internet host, run the following commands on the local machine.

```
tredir PORTNUMBER internet.host:25
/usr/lib/sendmail -q
```

We now turn to receiving electronic mail using `term`. We will assume that mail is sent to your account on the network machine `mailhost.` The simplest solution is to just use `trsh` or `termtelnet` to log on to `mailhost` and read your mail on there. However, it is also possible to have your mail automatically downloaded to your local machine. One way to do this is to use the Post Office Protocol, (POP). POP was designed for exactly this purpose: to deliver mail to machines that have intermittent network connections. To use POP, you must have a POP server installed on `mailhost`. Assuming that you do, you can then use a POP client to download your mail every few minutes. This is done, as you might expect, using `tredir`. The POP service is 110 (*note* that there is an older protocol, POP-2, which uses port 109; in this document we describe POP-3, which is the latest version of POP). There are several POP clients available. One, written in the script language `perl` is pop-perl-1.X, written by William Perry and maintained by myself - it can be found on `sunsite` in `/pub/Linux/system/Mail`.

To use POP, you redirect a local port to port 110 on `mailhost` and configure your client to retrieve your mail from localhost using the local port. As an example, we will assume that there is a POP server running on `mailhost`. We will redirect the local port 2110, and fire up the pop-perl client:

```
$ tredir 2110 mailhost:110
Redirecting 2110 to mailhost:110
$ pop
Username: bill
Password: <enter your password for mailhost>
Pop Host: name of local
Pop Port: 2110
Starting popmail daemon for bill
```

If you do not have a POP server available, the BCRMailHandler package has a program to download your mail over a `term` link to your local machine. I have not used it, but anyone who has is welcome to comment. You can also use the term.mailerd+smail package for this purpose. Alas, both BCRMailHandler and the term.mailerd.smail package do not work anymore with `term` versions 2.0.0 and higher.

8 tudpredir

`tudpredir` is similar to tredir when you look at what these programs do and how they are executed. The big difference between the two programs is that tredir is used to redirect TCP network services while `tudpredir` redirects UDP (User Datagram Protocol) network services over the `term` link. One more important difference between the two programs is that tredir becomes a background daemon once it has successfully established the local port, while `tudpredir` commands must be placed in the background manually.

The common command format of `tudpredir` is:

```
tudpredir [this_computer:]port [that_computer:]port
```

9 Automating Things

Now that you know how to get all your network services over `term`, it would be nice to set things up in such a way that your link is set up and configured automatically. There are basically an infinite number of ways of doing so, depending on what communication program you use and how you log in to your remote system.

One program that I have not used, but I have heard is quite nice, is `fet`: a front end for `term`. It is designed to log you into a remote system and fire up `term` and all your `tredir`'s. Any comments on `fet` would be most welcome.

I shall now give an example of a set of commands that use `kermit` to log into the remote system and then performs all of the `term` initializations. Obviously, if you use these examples, you will have to modify them for your own login procedures.

The command which is actually invoked is the shell script `knet,` given by:

```
#!/bin/sh
/usr/bin/kermit -y $HOME/.kerm_term > $HOME/klog < /dev/null 2>& 1
exec $HOME/bin/tstart >> $HOME/klog 2>& 1
```

The script `.kerm_term` is given by:

```
pause 2
# The number you want to dial
output atdtXXXXXXX \13
# Login to the terminal server
input 145 {name: }
output MYNAME \13
input 3 {word: }
output MYPASSWORD \13
input 5 {xyplex>}
# Make the line transparent
output term telnet-t \13
output term stopb 1 \13
# Connect to the remote host
output telnet remotehost.somedomain.org \13
input 10 {ogin: }
output MYOTHERNAME \13
input 3 word:
output MYOTHERPASSWORD \13
pause 5
```

```
# Fire up term on the remote host
output exec term -s 38400 -l $HOME/tlog -w 10 -t 150  \13
! /usr/bin/term -r -l $HOME/tlog -s 38400 -c off -w 10 -t 150 < /dev/modem > /dev/modem &
# Open other clients here
suspend
!killall -KILL term
```

and finally, the script `tstart` which fires up the `term` clients is given by

```
#!/bin/sh
#
# This lets mail get out, can read news here, can pick up my mail here
#
/usr/local/bin/tredir 2025 25 2119 newshost:119 2110 pophost:110
#
# So I can open up Xwindows here
#
/usr/local/bin/trsh -s txconn
#
# So I will receive mail....
#
/usr/local/bin/pop
#
# Clean out the queue, in case of boo-boos
#
/usr/bin/runq
#
# Done now
#
echo ^G^G > /dev/console
```

When finally you want to close the connection, you resume and terminate kermit. The last line of the script kills the local term and returns the system in its initial state.

`(Note of the author: instead of doing '!killall -KILL term', I think it should be possible just to do '!tshutdown'. This should also work?)`

As I said, there are zillions of ways to do so; these are just meant as examples to get you started. Other examples can be found in the packages `autoterm` and `JoelTermStuff`.

10 Porting software for use with `term`

In principle, all programs that can be used over a network can also be used in combination with `term`. Some of them you can get as binaries with `term` support already built in. These include telnet, (nc)ftp, Mosaic and many others. Most of these programs are compiled for `term` 1.17 or earlier. They should, however, still work with the newer versions of `term`.

Another way to make programs `term` aware is to port them yourself. This process will be described in the next subsection.

The last way to make your programs `term`-aware is to `termify` them.

10.1 Port and compile the sources

Porting software to `term` can be done using a fairly simple porting procedure:

If installed in /usr/local by root:

1. Add to the compile flags `-include /usr/local/include/termnet.h`
2. and add to the library list `-ltermnet`

If installed in your home directory:

1. Add to the compile flags `-include ~/term/termnet.h`
2. and add to the library list `-L~/term -ltermnet`

Now compile the software as described in the INSTALL or README document that came with the software. That should do!

At this point the commands should work both with and without `term`.

```
telnet localhost
```

does not use `term` to connect, but

```
telnet bohr.physics.purdue.edu
```

will use `term` only if you do not have some other type of network connection.

Some commands like `rlogin` can only be executed by root and the owner of the `term` connection (privileged persons).

Some `term` commands will be `term` transparent and only use `term` when there is not another option. Some common examples are `telnet` and `ftp`.

Others require an external flag to tell them it is alright to use `term`. These programs include `xarchie`, `fsp` and `ytalk`.

You can either flag these programs to use `term` by setting the environmental variable TERMMODE as specified in README.security, or running `make installnet`. Eventually, the `termnet` file created will contain special networking instructions, but for now only its existence is checked.

If you add an `ethernet connection`, you can then simply remove the `termnet` file and continue to use the same binaries!

`NOTE`: Programs that were ported back in the days of `client.a`, can still be recompiled for use with newer versions of `term` simply by changing the `client.a` reference to `libtermnet.a`.

10.2 Termify

This package will convert dynamically linked binaries for `term` use.

Before you can use `termify` you have to make sure that you have `term` version 2.2i (is this version 2.2.8?) or later and libc.so.4.5.26 or later. Then you have to create the file *libt.so.4* in the directory */lib* (see the README file in the package).

Problem at this moment is that you have to remake the file *libt.so.4* every time you upgrade `term` versions.

After you have created the library you can let `termify` 'crunch' the program you want to make `term`-aware, by using the command:

```
termify <command name>
```

When you do not like the result you can 'un'termify the program you have just termified using the command:

```
termify -u <command name>
```

Last, the package also contains a script for completely termifying 'smail'; so no special transport definitions are necessary. The only thing you possibly want to change is the 'From: ' address.

11 Term clients

11.1 Term clients available on ftp sites.

Below a list of application running with `term` is given. I am not stating that this list is complete; so any completion is welcome. As far as possible I will present the site and directory where the application can be found (all to my knowledge). When I state `sunsite.unc.edu` as the place to find the application I mean that you can find it in one of the following two directories:

1. /pub/Linux/apps/comm/term/apps
2. /pub/Linux/apps/comm/term/extra

Here we go :-)

`term` package:

tupload	
tdownload	(versions 2.1.0 and higher)
trsh	
tmon	
tredir	
tudpredir	(versions 2.0.0 and higher)
txconn	
trdate(d)	
tshutdown	
libtermnet	

File transfer:

ftpd	sunsite.unc.edu
termncftp	sunsite.unc.edu
ncftp185	sunsite.unc.edu:/pub/Linux/system/Network/file-transfer
fsp	sunsite.unc.edu:/pub/Linux/system/Network/file-transfer

Information systems:

lynx	
Mosaic	sunsite.unc.edu:/pub/Linux/system/Network/info-systems/Mosaic
chimera	
netscape	sunsite.unc.edu:/pub/Linux/system/Network/info-systems
httpd	
xgopher	
gopher	sunsite.unc.edu

Remote login:

termtelnet	sunsite.unc.edu
rlogin	physics.purdue.edu:/pub/bcr/term/extra
rsh	physics.purdue.edu:/pub/bcr/term/extra

Netnews:

tin 1.3	sunsite.unc.edu:/pub/Linux/system/Mail/news
news2	sunsite.unc.edu

Mail:

slurp	sunsite.unc.edu
smail	sunsite.unc.edu
term.mailerd+smail	sunsite.unc.edu
BCRMailHandlerXXX	physics.purdue.edu:/pub/bcr/term

Automating scripts:

`JoelTermStuff`	`sunsite.unc.edu`
`autoterm`	`sunsite.unc.edu`
`fet`	`sunsite.unc.edu`

Other programs:

`inetd`	`sunsite.unc.edu`
`rdate`	`sunsite.unc.edu`
`xgospel`	`sunsite.unc.edu:/pub/Linux/games/x11/networked`
`termify`	`physics.purdue.edu:/pub/bcr/term/extra`
`xboard`	`sunsite.unc.edu`
`ircII`	`sunsite.unc.edu:/pub/Linux/system/Network/chat`
`whois`	
`xwebster`	`sunsite.unc.edu`
`sxpc`	`ftp.x.org:/R5contrib`
`xztalk`	`sunsite.unc.edu:/pub/Linux/apps/sound/talk`

11.2 The termnet package

The package `termnet-2.0.4-Linux-bin.tar.gz` (sunsite.unc.edu:/pub/Linux/apps/comm/term) contains a couple of precompiled `term` clients and a couple of scripts, manual pages and libtermnet.so.2.00.04. The clients are compiled using this version of libtermnet.so. The package contains the following clients:

```
fet        perl       sperl4.036  tmon       tshutdown  xgopher
finger     perl4.036  suidperl    trdate     tudpredir  ytalk
ftp        rcp        taintperl   trdated    tupload
fwhois     rlogin     telnet      tredir     txconn
ncftp      rsh        term        trsh       xarchie
```

WARNING: The package also contains the complete set of compiled clients of `term` 2.0.4 including `term` itself. Do not install this package before you are sure about what you want. You will destroy other versions of `term` and its clients when you start moving executables around.

11.3 Asked for but not yet supported

1. `DOOM`: The problem with this game seems to be the fact that it uses port 5029 both as client and as server.
2. `NFS`: The NFS server is only supposed to accept requests if the socket requesting the connection is bound to a port below 1024. This seems to be troublesome. However, some NFS servers have an 'insecure' option. In this case NFS might work eventually, if RPC support is added to Term.

12 Term and Security

In this section I will point to some security aspects of `term`. The problems will be explained and a way to improve security will be given.

12.1 trsh

`trsh` is insecure when it is used to access the local Linux box from the remote system. The problem with `term` and its clients is that beside the owner of the `term` connection also 'root' can execute `term`-aware programs over the connection.

This also means that 'root' on the remote system can execute `trsh` and thus can enter the login which owns the `term` connection quite easy. If this owner on the local box is 'root' then you will be in hell.

The solution to this problem is easy: you just have to put the following line in the termrc file on the local box:

```
denyrsh on
```

With this set in the termrc file, nobody can use `trsh` on the remote site anymore to access your machine. When you and others want to access your local Linux box over the `term` connection this can still be done using `telnet` and redirected ports.

12.2 sxpc, xhost and xauth

`sxpc` in combination with 'xhost +' is very dangerous when you are not using `xauth`.

Using xauth is very important to maintaining security when using sxpc. If you do not use xauth when using sxpc all the dangers of running with 'xhost +' apply. These dangers include but are not limited to:

- Someone watching what is displayed on your screen
- Someone watching what you type
- Someone typing in one of your windows (for example: a command to delete all your files :-(

xauth is available in X releases R4 and later. Here I will describe how to set up basic usage of xauth. This configuration is vulnerable to network snooping, but if you can live with that it should be fine.

NOTE: when using xauth your $DISPLAY variable must NOT be set to localhost (or localhost:whatever). If your $DISPLAY variable does use localhost the clients will be unable to find the appropriate authorization information. The workaround is to use the real hostname of the machine. If you follow the compilation instructions in the README, and compile without -DNOGETHOSTNAME then everything should work.

The machine where you will be running clients will be called C, the machine where you wish to display them will be called D.

First choose a 'key', up to 16 pairs of hexadecimal digits (so an even number of characters from the ranges 0-9 and a-f). You will need to supply this key in place of <key> in the example below.

On C:

```
% xauth
xauth:  creating new authority file $HOME/.Xauthority
Using authority file $HOME/.Xauthority
xauth> add Chostname:8 MIT-MAGIC-COOKIE-1 <key>
xauth> exit
```

On D:

```
% xauth
xauth:  creating new authority file $HOME/.Xauthority
Using authority file $HOME/.Xauthority
xauth> add Dhostname/unix:0 MIT-MAGIC-COOKIE-1 <key>
xauth> add Dhostname:0 MIT-MAGIC-COOKIE-1 <key>
xauth> exit
```

When starting the X server on D you should give the flag *-auth $HOME/.Xauthority*. You may need to edit or create a *$HOME/.xserverrc* to control how the X server is started. For example:

```
#!/bin/sh
exec X  -auth $HOME/.Xauthority $*
```

Make sure that your *.Xauthority* file is readable only by you on both C and D.

13 Things to remember

In this section I try to present you with a list of useful ftp addresses, URL's etc. where you can find software and information about `term`.

ftp:

```
sunsite.unc.edu:/pub/Linux/apps/comm/term/<whole-directory-tree>
sunsite.unc.edu:/pub/Linux/docs/HOWTO
physics.purdue.edu:/pub/bcr/term/<whole-directory-tree>
```

URL:

```
http://sunsite.unc.edu/mdw/HOWTO/Term-HOWTO.html
http://zeus.cs.kun.nl:4080/term-howto/Term-HOWTO.html
http://physics.purdue.edu/~bcr/homepage.html
```

netnews:

`comp.os.linux.announce`	`announce of new term versions and Term-HOWTO`
`comp.os.linux.help`	`ask your questions about term here`
`comp.os.linux.misc`	`or here`
`comp.protocols.misc`	`answers to term questions are also posted here.`

When you start asking questions on netnews please make sure that you give people in the groups as much information as they need to solve your problem (`term` version number, way you set up your connection etc.). At this moment many `term` versions are in use and all have their specific and common problems. Therefore, when you want a useful answer, at least state the version of `term` you are using. Otherwise, in some cases only wild guesses are possible to help you solve your problems.

14 Reliability of `term` versions

Many versions of `term` are around now. The maintainer of `term`, Bill Riemers, has made a list of `term` versions stating what versions are reliable and what versions you had better avoid. This list is the following:

```
term110         --> cannot really say
term111         --> cannot really say
term112         --> cannot really say
term113         --> cannot really say
term114         --> fairly stable BETA version
term115         --> unstable BETA version
term116         --> unstable BETA version
term117         --> unstable BETA version
term118         --> semi-stable BETA version
term119         --> stable GAMMA version
term-2.0.X      --> semi-stable BETA versions
term-2.1.X      --> more stable BETA versions
term-2.2.X      --> new BETA versions
term-2.3.X      -->
```

15 Term speed table

Thanks to Bill McCarthy we now have a table with `term` speed information for different modems, `term` versions and connection conditions. Its purpose is to give new and experienced users some idea what other people are using and the results they are getting.

```
                          LINUX TERM CHART 8/14/94


|___modem speed/make___|___line speed__|__avg cps__|__high__|__term ver_|
```

```
| 1)  USR SP 14.4      |   9600          |  950      |  963   | 1.17      |
| 2)  USR SP 14.4      |  14400          | 1376      |  n/a   | 1.18p06   |
| 3)  Zoom 2400        |     2400        |  220      |  230   | 1.19      |
| 4)  Boca V.32bis 14  |  57600          | 1400      |  n/a   | 1.01/09?  |
| 5)  Viva 14.4        |  14400          | 1300      |  n/a   | 1.16      |
| 6)  USR SP 14.4      |  14400+         | 1550      | 1680   | 1.19      |
| 7)  Intel 14.4 Fax   |  14400          | 1400      | 1650   | 2.0.4     |
| 8)  cable tv hookup  |  57600          | 1500      | 1800   | 1.18p06   |
| 9)  Twincom 144/DFi  |  57600          | 1500      | 4000?  | 2.0.4     |
| 10) USR SP 14.4      |  14400          | 1200      | 1500   | 1.08      |
| 11) cable tv hookup  |  19200          | 1300      | 1800   | 1.19      |
|-----------------------------------------------------------------------|

+Command flags/termrc settings:

1) default escapes   2) window 5       3) baudrate 2400    4) n/a
   baudrate 9600        timeout 200       window 3
   window 10                              noise on
   timeout 150

5) compress off      6) baudrate 19200  7) ignore 19+17    8) compress off
   window 10            compress on        window 4           escape 0, 13,
   timeout 150                             timeout 90         16-19, 255
   baudrate 38400                                             baudrate 0
                                                              shift 224
                                                              flowcrtl 500
                                                              window 10
                                                              timeout 70
                                                              retrain on
                                                              breakout 24
9) compress off     10) compress off      11) baudrate 19200
   baudrate 57600       baudrate 38400        compress on
   window 10            escape 17, 19         shift 224
   timeout 200          remote                escape 0, 13 16-17
   noise on                                   19, 255
   share on                                   window 10
   remote                                     timeout 40

Escaping characters at one end also implies ignoring them on the other end.
```

16 Hints and Tricks found on the net

In the Linux related newsgroups many questions about `term` are coming back every couple of weeks, together with the answers to these questions. To reduce traffic to the newsgroup, in this section I shall try to make a composition of these questions and the answers to them. Some of the answers have been checked by me as I also had related problems. Others, I have just taken from the newsgroups without testing them.

- Many people, especially those who are using Ultrix, seem to have trouble with `vi` presenting less than 24 lines on a window with 24 lines. There are two ways to get rid of this problem:
  1. Log in into the remote system using: `trsh -s telnet < hostname>`
  2. Put 'resize; clear' in your `.login` file
  3. The best solutions seems to be to enter the following remote: `stty 38400`

- Many people seem to have problems with crashing `term` connections, whatever may be the reason of the crashes. So before starting applications people want to know whether their `term` connection is still alive or not. This can be checked using the following small shell script examples:

```
When you are using tcsh:

if ( { trsh -s true } ) then
  ...
endif
```

  When you are using bash:

```
if trsh -s true; then
  ...
fi
```

- The new WWW browser called `Netscape` is causing people problems to get it to work with `term`. The good news is that it will work under `term`. Here is how:

```
1. Termify netscape
2. Edit /etc/services to declare port 1080 (or another port if you
   like) for http like:
      Service          Portno./kind            Used by
      http             1080/tcp                term
3. execute 'tredir 1080 80'
4. Fire up termnetscape
      under Options | Preferences | Mail/Proxys
      leave _all_ of the proxy boxes blank
      set the SOCKS box to 'term' & 1080
```

  When you exit the Options menu, you will get a warning about 'unknown socks server'; just ignore it.

17 Other Things

Some things that might be included:

- Extension of troubleshooting

- Extension of security issues
- Termwrap
- Suggestions

Anyway, if you have suggestions, criticism, suggestions, or anything else to say about this document, please fire away. As Bill Reynolds is very busy at the moment I, Patrick Reijnen, have taken over the authorship of the Term-HOWTO. I can (currently) be reached at `patrickr@cs.kun.nl` or `patrickr@sci.kun.nl`.

18 Acknowledgements

A lot of people have to be thanked. First and foremost Michael O'Reilly and all the developers of `term`, who have provided us with such a great tool. I would also like to thank everyone who gave feedback and contributed to this HOWTO. They include Ronald Florence, Tom Payerle, Bill C. Riemers, Hugh Secker-Walker, Matt Welsh, Bill McCarthy, Sergio, Weyman Martin and everybody I forgot to mention.