NAME

trsh, tupload, tredir, txconn, tmon – clients to term

SYNOPSIS

trsh [options] [command]

tupload [options] file ... [remote_directory]

tredir [options] {local_port [remotehost:]remote_port} ...

txconn [options]

tmon [options]

DESCRIPTION

These clients connect to a local *term*(1) daemon and initiate a communication link with the remote daemon and remote processes. *Trsh* runs an interactive shell or commands on the remote system, *tupload* transfers files from the local system to the remote, *tredir* redirects connections from ports on the local system to ports on a remote system, *txconn* redirects X-server connections from the local system to the remote, and *tmon* prints summary information about a *term* link.

As far as the clients are concerned, the link established by *term*(1) is completely symetric; the clients don't know which system is your physically local system (the system you're typing at) and which is physically remote. From a client's point of view, the local system is the one on which the client is executing. The remote system is the other system, the one on which the client is not executing.

USAGE

trsh With no arguments, *trsh* is similar to *rlogin*(1). An interactive shell is started on the remote system. Terminal features such as character echoing, line editing, and window resizing are handled by a tty on the remote system. If a *command* is given, it executes on the remote system instead of the default shell. Specifying *command* without the –s option is primarily useful for starting a non-default shell remotely.

There is a single *trsh* specific option:

–s Use a simple connection. This makes *trsh* behave more like *rsh*(1). The specified *command* is executed by a shell on the remote system. Like *rsh*(1), quoted shell metacharacters are interpreted remotely, so filename redirection, multiple commands, or pipes can be used remotely. If no *command* is given, the –s option is ignored.

The –s option should almost always be used when a *command* is provided. In particular, it should be used if *trsh* is part of a pipeline. The only exception to this rule is when remote terminal features are needed.

Trsh differs from *rlogin*(1) and *rsh*(1) in a couple of significant ways. First, the stdout and stderr of the remote shell or command are both directed to the local stdout. Second, the connection is broken as soon as a local end-of-file occurs. This means that any output generated by the remote system after EOF is lost. For example,

```
trsh -s wc < /etc/passwd
```

runs the remote *wc*(1) command on the contents of the local /etc/passwd, but will not produce any output.

tredir *Tredir* accepts pairs of port numbers, the first of each being the local port number to redirect, and the second the remote port to redirect into. *Tredir* will also accept a hostname with each of the second port numbers. For example,

```
tredir 119 my.nntp.host:119
```

can be run on the non-internet side of the link. Then, any local program that connects to port 119 (like an NNTP newsreader), will be connected to port 119 on my.nntp.host. *Tredir* becomes a background daemon once it has successfully established the local port. More than one port can be redirected at once with a single *tredir.* For example,

```
tredir 119 my.nntp.host:119 4000 remote.home:23
```

would redirect ports 119 and 4000, 119 to the nntp port on my.nntp.host, and port 4000 to the login port on remote.host.

tupload *Tupload* takes a list of local file names to upload. If there is more than one name, then the last name is checked to see if it matches a remote directory. If it does match a remote directory, the files are uploaded to this directory. If a remote file exists, and is smaller than the local file, tupload will assume that it was a previous upload that got aborted, and will attempt to resume the upload. Tupload specific options are.

–f Tells *upload* to not do any possible resumptions. Always overwrite the remote file.

–u Unlink the local file after a succesful upload.

–v Verbose mode. Will write out the number of bytes transfered, the name of the remote file, and the CPS rate. Two 'v's will produce more output.

–q Quiet mode. Tupload will not write out any messages.

–as <remote_file>
Any of the local *file* names may be followed by the –as <remote_file> option to specify a new name for the remote copy of the file. By default the remote file will have the same name as the local file.

txconn *Txconn* establishes an X screen port on the system it is run on. Connections to that screen will be redirected to the X server on the other system. If your DISPLAY environment variable is set, *txconn* will try to use the screen number given by DISPLAY. *Txconn* prints the number of the screen it actually establishes in a format suitable for re-setting your DISPLAY variable. The recommended method of starting *txconn* and setting you DISPLAY variable at the same time is to run

```
setenv DISPLAY ‘hostname‘‘txconn‘
```

from csh and related shells, or

```
export DISPLAY=‘hostname‘‘txconn‘
```

from sh and related shells. *Txconn* becomes a background daemon once it has successfully established the port.

tmon *Tmon* periodically queries the *term* daemon for information about the number of clients and the data flow-rate of the channels. It updates its output on the terminal it is running in.

GLOBAL OPTIONS

The following options are accepted by all *term* clients

–t <server>
Specifies which term daemon socket to connect to (˜/.term/socket<server>). This is useful if you have specific socket names for connections to specific remote hosts, or if you have multiple terms using different serial connections.

–r Specifies a raw link (no compression).

–c Specifies a compressing link.

–p <number>
Specifies priority for this link. A client with higher priority will be able to send all its data before lower priority clients. This may be changed in future releases. The priority of *trsh* defaults to 2 and *tupload* defaults to -2. All other clients default to zero.

EXAMPLES

Some simple, contrived examples:

```
trsh
```

This establishes an interactive shell on the remote machine. The connection is maintained until you exit the remote shell.

```
trsh -s who
```

prints out a list of current users of the remote machine.

```
ls -li | awk '{print $1, $5}' | trsh -s plot
```

will generate a scatter plot of file size vs inode. Most likely this would be run on the physically remote system to produce local graphical output about the files on the remote system.

```
tupload -vv -r lin-0.99.3.tar.Z -as lin993.tar.Z /new
```

This will upload the local file 'lin-0.99.3.tar.Z' to the remote system with the name '/new/lin993.tar.Z'. It will resume an upload if there is an existing /new/lin993.tar.Z, and will writeout the CPS and bytes transfered every 2K. It will not do compression on the way.

```
tredir 6667 munagin.ee.mu.oz.au:6500 9017 17
```

This will map any connections to port 6667 on your local host to port 6500 on munagin.ee.mu.oz.au and any connections on your local host to port 9017 to port 17 on the remote machine. So if you do 'telnet 0 6667' locally, you will be connected to port 6500 on munagin, and if you do 'telnet 0 9017', you will be connected to port 17 on the remote machine.

Let's say the physically remote machine, earlobe.mit.edu, is on the internet. You can establish a port on earlobe which will allow anyone on the internet to telnet to your physically local, non-internet machine. On the local machine you could type

```
trsh -s tredir 4000 23
```

Alternatively, if you already have an interactive shell on the remote machine you could just run the tredir command from that shell. In either case, once the tredir daemon is running, users typing

```
telnet earlobe.mit.edu 4000
```

from any internet host will get a login prompt from your physically local machine.

BUGS

None of the local environment is propogated by *trsh,* so the remote TERM variable is likely to be wrong. If the remote command for 'trsh -s <command>' doesn't consume its input fast enough, the channel can lose data. *Trsh* breaks the channel as soon as the local EOF is received, thus losing any final output from the remote process. *Tmon* gives bogus output on some machines (e.g. NeXT).

SEE ALSO

term(1), *term_setup*(1).