## NAME

nc - TCP/IP swiss army knife

## SYNOPSIS

**nc** *[-options] hostname port[s] [ports] ...*
**nc** *-l -p port [-options] [hostname] [port]*

## DESCRIPTION

**netcat** is a simple unix utility which reads and writes data across network connections, using TCP or UDP protocol. It is designed to be a reliable back-end tool that can be used directly or easily driven by other programs and scripts. At the same time, it is a feature-rich network debugging and exploration tool, since it can create almost any kind of connection you would need and has several interesting built-in capabilities. Netcat, or nc as the actual program is named, should have been supplied long ago as another one of those cryptic but standard Unix tools.

In the simplest usage, nc host port creates a TCP connection to the given port on the given target host. Your standard input is then sent to the host, and anything that comes back across the connection is sent to your standard output. This continues indefinitely, until the network side of the connection shuts down. Note that this behavior is different from most other applications which shut everything down and exit after an end-of-file on the standard input.

Netcat can also function as a server, by listening for inbound connections on arbitrary ports and then doing the same reading and writing. With minor limitations, netcat doesn't really care if it runs in client or server mode -- it still shovels data back and forth until there isn't any more left. In either mode, shutdown can be forced after a configurable time of inactivity on the network side.

And it can do this via UDP too, so netcat is possibly the udp telnet-like application you always wanted for testing your UDP-mode servers. UDP, as the U implies, gives less reliable data transmission than TCP connections and some systems may have trouble sending large amounts of data that way, but it's still a useful capability to have.

You may be asking why not just use telnet to connect to arbitrary ports? Valid question, and here are some reasons. Telnet has the standard input EOF problem, so one must introduce calculated delays in driving scripts to allow network output to finish. This is the main reason netcat stays running until the *network* side closes. Telnet also will not transfer arbitrary binary data, because certain characters are interpreted as telnet options and are thus removed from the data stream. Telnet also emits some of its diagnostic messages to standard output, where netcat keeps such things religiously separated from its *output* and will never modify any of the real data in transit unless you *really* want it to. And of course telnet is incapable of listening for inbound connections, or using UDP instead. Netcat doesn't have any of these limitations, is much smaller and faster than telnet, and has many other advantages.

## OPTIONS

| | |
|---|---|
| *-c string* | specify shell commands to exec after connect (use with caution). The string is passed to /bin/sh -c for execution. See the *-e* option if you don't have a working /bin/sh (Note that POSIX-conformant system must have one). |
| *-e filename* | specify filename to exec after connect (use with caution). See the *-c* option for enhanced functionality. |
| *-g gateway* | source-routing hop point[s], up to 8 |
| *-G num* | source-routing pointer: 4, 8, 12, ... |
| *-h* | display help |
| *-i secs* | delay interval for lines sent, ports scanned |
| *-l* | listen mode, for inbound connects |
| *-n* | numeric-only IP addresses, no DNS |

| | |
|---|---|
| *-o file* | hex dump of traffic |
| *-p port* | local port number (port numbers can be individual or ranges: lo-hi [inclusive]) |
| *-q seconds* | after EOF on stdin, wait the specified number of seconds and then quit. If *seconds* is negative, wait forever. |
| *-b* | allow UDP broadcasts |
| *-r* | randomize local and remote ports |
| *-s addr* | local source address |
| *-t* | enable telnet negotiation |
| *-u* | UDP mode |
| *-v* | verbose [use twice to be more verbose] |
| *-w secs* | timeout for connects and final net reads |
| *-C* | Send CRLF as line-ending |
| *-z* | zero-I/O mode [used for scanning] |
| *-T type* | set TOS flag (type may be one of Minimize-Delay, Maximize-Throughput, Maximize-Reliability, or Minimize-Cost.) |

## COPYRIGHT

Netcat is entirely my own creation, although plenty of other code was used as examples. It is freely given away to the Internet community in the hope that it will be useful, with no restrictions except giving credit where it is due. No GPLs, Berkeley copyrights or any of that nonsense. The author assumes NO responsibility for how anyone uses it. If netcat makes you rich somehow and you're feeling generous, mail me a check. If you are affiliated in any way with Microsoft Network, get a life. Always ski in control. Comments, questions, and patches to hobbit@avian.org.

## NOTES

Some port names in /etc/services contain hyphens -- netcat currently will not correctly parse those unless you escape the hyphens with backslashes (e.g. netcat localhost 'ftp-data').

## BUGS

Efforts have been made to have netcat do the right thing in all its various modes. If you believe that it is doing the wrong thing under whatever circumstances, please notify me and tell me how you think it should behave. If netcat is not able to do some task you think up, minor tweaks to the code will probably fix that. It provides a basic and easily-modified template for writing other network applications, and I certainly encourage people to make custom mods and send in any improvements they make to it. Continued feedback from the Internet community is always welcome!

## EXAMPLES

For several netcat recipes, please see /usr/share/doc/netcat/README.gz and /usr/share/doc/netcat/README.Debian.gz.

## AUTHOR

This manual page was written by Joey Hess <joeyh@debian.org> and Robert Woodcock <rcw@debian.org>, cribbing heavily from Netcat's README file.

Netcat was written by a guy we know as the Hobbit <hobbit@avian.org>.