Ncat Users Guide
Your General-Purpose Network Connector

Fyodor

Ncat is a feature-packed networking utility which will read and write data across a network from the command line. Ncat was written for the Nmap Project as a much-improved reimplementation of the venerable Netcat. It uses both TCP and UDP for communication and is designed to be a reliable back-end tool to instantly provide network connectivity to other applications and users. Ncat will not only work with IPv4 and IPv6 but provides the user with a virtually limitless number of potential uses.

Among Ncat's vast number of features there is the ability to chain Ncats together, redirect both TCP and UDP ports to other sites, SSL support, and proxy connections via SOCKS4 or HTTP (CONNECT method) proxies (with optional proxy authentication as well). Some general principles apply to most applications and thus give you the capability of instantly adding networking support to software that would normally never support it.   .   .

```
    \`-"'"-'/
     } 6 6 {
    ==. Y ,==
     /^^^\  .
    /     \  )
(  )-(  )/     _
 -""---""---   /
/   Ncat    \_/
  (     ____
   \_.=|____E
```

Ncat is a general-purpose command-line tool for reading, writing, redirecting, and encrypting data across a network. It aims to be your network Swiss Army knife, handling a wide variety of security testing and administration tasks. Ncat is suitable for interactive use or as a network-connected back end for other tools. Ncat can:

- Act as a simple TCP/UDP/SSL client for interacting with web servers, telnet servers, mail servers, and other TCP/IP network services. Often the best way to understand a service (for fixing problems, finding security flaws, or testing custom commands) is to interact with it using Ncat. This lets you you control every character sent and view the raw, unfiltered responses.

- Act as a simple TCP/UDP/SSL server for offering services to clients, or simply to understand what existing clients are up to by capturing every byte they send.

- Redirect or proxy TCP/UDP traffic to other ports or hosts. This can be done using simple redirection (everything sent to a port is automatically relayed somewhere else you specify in advance) or by acting as a SOCKS or HTTP proxy so clients specify their own destinations. In client mode, Ncat can connect to destinations through a chain of anonymous or authenticated proxies.

- Run on all major operating systems. We distribute Linux, Windows, and Mac OS X binaries, and Ncat compiles on most other systems. A trusted tool must be available whenever you need it, no matter what computer you're using.

- Encrypt communication with SSL, and transport it over IPv4 or IPv6.

- Act as a network gateway for execution of system commands, with I/O redirected to the network. It was designed to work like the Unix utility cat, but for the network.

- Act as a connection broker, allowing two (or far more) clients to connect to each other through a third (brokering) server. This enables multiple machines hidden behind NAT gateways to communicate with each other, and also enables the simple Ncat chat mode.

These capabilities become even more powerful and versatile when combined.

Ncat is our modern reinvention of the venerable Netcat (nc) tool released by Hobbit in 1996. While Ncat is similar to Netcat in spirit, they don't share any source code. Instead, Ncat makes use of Nmap's well optimized and tested networking libraries. Compatibility with the original Netcat and some well known variants is maintained where it doesn't conflict with Ncat's enhancements or cause usability problems. Ncat adds many capabilities not found in Hobbit's original nc, including SSL support, proxy connections, IPv6, and connection brokering. The original nc contained a simple port scanner, but we omitted that from Ncat because we have a preferred tool for that function.

This guide starts with examples of basic Ncat usage, then moves on to more advanced features. Those are followed by practical sections which use examples to demonstrate how Ncat can solve common real-world problems. A few neat Ncat tricks are covered as well.

Basic Usage

Ncat always operates in one of two basic modes: connect mode and listen mode. In connect mode, Ncat initiates a connection (or sends UDP data) to a service that is listening somewhere. For those familiar with socket programming, connect mode is like using the connect function. In listen mode, Ncat waits for an incoming connection (or data receipt), like using the bind and listen functions. You can think of connect mode as “client” mode and listen mode as “server” mode.
To use Ncat in connect mode, run

```
ncat <host> [<port>]
```

<host> may be a hostname or IP address, and <port> is a port number. Listen mode is the same, with the addition of the --listen option (or its -l alias):

```
ncat --listen [<host>] [<port>]
ncat -l [<host>] [<port>]
```

In listen mode, <host> controls the address on which Ncat listens; if you omit it, Ncat will bind to all local interfaces (INADDR_ANY). If the port number is omitted, Ncat uses its default port 31337. Typically only privileged (root) users may bind to a port number lower than 1024. A listening TCP server will accept multiple concurrent connections up to the connection limit. The server receives everything sent by any of its clients, and anything the server sends is sent to all of them. A UDP server will communicate with only one client (the first one to send it data), because in UDP there is no list of “connected” clients.

By default, Ncat uses TCP and IPv4. The option --udp or -u enables UDP instead, and -6 enables IPv6. The rest of this guide documents all the Ncat options through descriptions and examples. For a quick summary of options at any time, run ncat --help or man ncat.

A Connect Mode Example

A good way to start learning about Ncat (and network protocols in general) is to connect to a network service and talk with it. In this case we use Ncat to manually retrieve a web page from an HTTP server, just as web browsers do in the background when you visit a web site. Example 1 shows a (truncated) sample session. Try it yourself! Text in bold is what you type; everything else is what comes back. The blank line after the GET line is required—just hit enter twice.

Example 1
Ncat as a web browser

```
$ ncat -C scanme.nmap.org 80
GET / HTTP/1.0

HTTP/1.1 200 OK
Date: Thu, 05 Feb 2009 15:31:40 GMT
Server: Apache/2.2.2 (Fedora)
Last-Modified: Mon, 19 May 2008 04:49:49 GMT
ETag: "fc8c91-2e3-44d8e17edd540"
Accept-Ranges: bytes
Content-Length: 739
Connection: close
Content-Type: text/html; charset=UTF-8

<html>
<head>
<title>Go ahead and ScanMe!</title>
</head>
```

Here we have instructed Ncat to connect to the host scanme.nmap.org on port 80, the port for HTTP. The -C option turns on CRLF replacement, which replaces any line endings you type with CRLF. CRLF line endings are required by many protocols, including HTTP, though many servers will accept a plain newline (LF) character.

GET / HTTP/1.0 requests the root document of the server; we are retrieving the same document named by the URL http://scanme.nmap.org:80/. The web server responds with a status code (HTTP/1.1 200 OK), followed by the HTTP header and the text of the web page. If you try this with other web servers, note that many of them are actually virtual hosts and you will need to send the Host header field. See RFC 2616 for more information about HTTP.

A Listen Mode Example

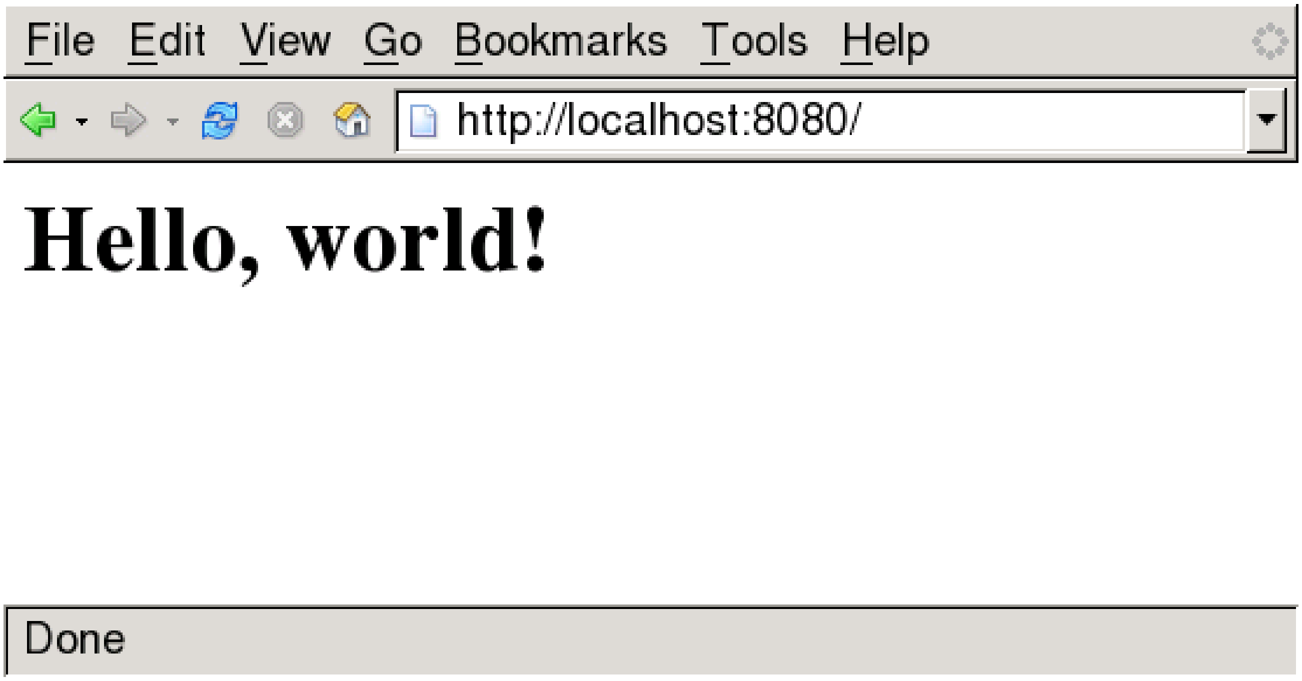
So much for using Ncat as a web browser. What about a web server? That's possible too; it just takes a bit of preparation. The first step is to create the document to serve. Create a text file called hello.http with these contents:

```
HTTP/1.0 200 OK

<html>
  <body>
    <h1>Hello, world!</h1>
  </body>
</html>
```

Now run the command ncat -l localhost 8080 < hello.http. This instructs Ncat to listen on the local port 8080 and read hello.http on its input. Ncat is now primed to send the contents of the file as soon as it receives a connection. Now open a web browser and type in the address http://localhost:8080/. Figure 1 shows a sample of what will appear.

Figure 1
Web page served by Ncat



http://localhost:8080/

Hello, world!

In the terminal where you ran Ncat, you will see everything the web browser sent to request the page. You should see a GET line like the one you sent in the connect mode example. This shows that Ncat by default both sends and receives.

If you try to refresh the page, it won't work. That's because Ncat ran out of input; it won't re-send what has already been sent. For more information on making a server that continually responds to requests, see the examples in the section called “Emulating Diagnostic Services”.

Connection Brokering

One of Ncat's most useful and unique abilities is called connection brokering. A listening Ncat in broker mode accepts connections from multiple clients. Anything received from one of the clients is sent back out to all the others. In this way an Ncat broker acts like a network hub, broadcasting all traffic to everyone connected.

Activate broker mode with the --broker option, which must be combined with --listen. It wouldn't make sense for a client to be a broker. See the section called “File Transfer” for details on using brokering to transfer files through restrictive firewalls, and the section called “Chatting” for using brokering to set up multi-user chat rooms.

SSL

Ncat can encrypt its traffic using SSL. In connect mode, simply add the --ssl option. Here is the syntax for connecting to an HTTPS server:

```
ncat -C --ssl <server> 443
```

Sometimes an SSL server will require a client certificate for authentication. When this is the case, use the --ssl-cert and --ssl-key options to give the locations of PEM-encoded files containing the certificate and private key, respectively. The certificate and key may be in the same file.

By default the client will not do any server certificate verification, so it will not be detected if the server has the wrong certificate or no certificate at all. Use the --ssl-verify option to require verification of the certificate and matching of the domain name.

```
ncat -C --ssl-verify <server> 443
```

Verification is done using the ca-bundle.crt certificate bundle shipped with Ncat, plus whatever trusted certificates the operating system may provide. If you want to verify a connection to a server whose certificate isn't signed by one of the default certification authorities, use the --ssl-trustfile to name a file containing certificates you trust. The file must be in PEM format.

```
ncat -C --ssl-verify --ssl-trustfile <custom-certs.pem> <server> 443
```

Verification should be done whenever it is feasible. Even with encryption, an unverified connection is vulnerable to a man-in-the-middle attack. Ncat does not do certificate revocation checking.
Ncat can act as an SSL server as well. The server must provide a certificate that clients can verify if they choose. If you start an SSL server without using the --ssl-cert and --ssl-key options, Ncat will automatically generate a certificate and 1,024-bit RSA key. The certificate will of course not be trusted by any application doing certificate verification. In verbose mode, the key's fingerprint will be printed so you can do manual verification if desired. Example 2 shows sample output.

Example 2
Automatic certificate generation

```
$ ncat -v --listen --ssl
Ncat ( http://nmap.org/ncat )
Generating a temporary 1024-bit RSA key. Use --ssl-key and --ssl-cert to use a
permanent one.
SHA-1 fingerprint: F0:13:BF:FB:2D:AA:76:88:22:60:3E:17:93:29:3E:0E:6B:92:C0:2F
```

Using an existing certificate and key is recommended whenever possible because it allows for robust server authentication. Use the --ssl-cert and --ssl-key options to pass in PEM-encoded files. For testing purposes you can generate a self-signed certificate and private key. If you have OpenSSL installed, use this command:

```
openssl req -new -x509 -keyout test-key.pem -out test-cert.pem.
```

For purposes of certificate verification, the commonName in the certificate should match the fully qualified domain name of the host that will run the server. After generating the files, start the server:

```
ncat --listen --ssl --ssl-cert test-cert.pem --ssl-key test-key.pem.
```

To make a verified client connection, copy the test-cert.pem file somewhere where the client can access it, then run

```
ncat --ssl-verify --ssl-trustfile test-cert.pem.
```

Command Execution

Ncat can execute an external command after establishing a connection. The command's standard input, output, and error streams are redirected to use Ncat's network connection. Anything received over the connection is given to the command's stdin, and anything the command writes is sent back out over the connection. This makes almost any terminal application accessible over a network (with some caveats).

The --exec option (alias -e) takes the full pathname of a command to execute, along with its arguments. The command is executed directly; Ncat does not interpret the given string beyond splitting the command and its arguments. Example 3 shows an example of usage.

Example 3
Running a command with --exec

```
ncat -l --exec "/bin/echo Hello."
```

The --sh-exec (-c) works the same way as --exec, except that it executes the command by passing it to /bin/sh -c on Unix or cmd.exe /C on Windows. You don't have to use the full pathname of the command if the command is in the PATH. Additionally you have access to shell facilities such as pipelines and environment variable expansion. Example 4 shows a command run with --sh-exec. This server, when connected to, sends back the name of its working directory.

Example 4
Running a command with --sh-exec

```
ncat -l --sh-exec "echo `pwd`"
```

The exec options can be used both in connect mode and in listen mode, but in listen mode they have an extra feature. For each connection received, Ncat forks off a new handler, leaving the original Ncat still running and waiting for connections. This is so even in UDP mode; the usual limit of only one client doesn't apply. In this way Ncat can work much like inetd. Many examples of the use of --exec and --sh-exec in listen mode are found in the section called “Emulating Diagnostic Services”.

Any program that takes input and produces output can be executed by Ncat, but not all programs are suited for interaction. The reason is that many programs buffer their input and output, so if they receive some bytes, they many not process those bytes and write output until their input buffer is full, or the output may be deferred until the output buffer is full. If another program sends a few bytes and then waits for a response, it may hang indefinitely. Buffers are flushed when input or output ends, so even those programs that don't work interactively will work when run on an entire file at a time.

Be careful when using the --exec and --sh-exec options. It can be dangerous to connect a new application to a network, especially one that wasn't written with potentially hostile input in mind. Any local vulnerabilities in an application may become remote vulnerabilities when you execute it through Ncat.

Output Options

Like any proper pipeline utility, Ncat reads from standard input and writes to standard output so you can redirect I/O to or from any program or file. The only exception is when Ncat is run with the --exec or --sh-exec options, in which case it communicates with the subprocess instead. Nothing in the streams is added, removed, or altered, unless you specifically ask for it with an option such as -C (CRLF processing) or --telnet (Telnet negotiation). If Ncat prints any diagnostic messages, they are sent to standard error so as not to interfere with the data stream. By default Ncat does not print any such messages, but you can enable them with the --verbose (-v) option. Use -v more than once for even more output.

Use the --output option or its alias -o to record a transcript of everything sent and received to a file:

```
ncat -C --output smtp-debug.log mail.example.com 25
```

The log contains everything sent and received without differentiation. Sometimes a hex dump is more useful than a plain text log; for that use --hex-dump or -x. Let's see what happens if we accidentally speak SMTP to an SSH server:

```
$ ncat -C --hex-dump ssh-hex.log scanme.nmap.org 22
SSH-2.0-OpenSSH_4.3
HELO example.com
Protocol mismatch.
```

The --hex-dump log file for this session:

```
[0000]   53 53 48 2D 32 2E 30 2D   4F 70 65 6E 53 53 48 5F   SSH-2.0- OpenSSH_
[0010]   34 2E 33 0A                                         4.3.
[0000]   48 45 4C 4F 20 65 78 61   6D 70 6C 65 2E 63 6F 6D   HELO exa mple.com
[0010]   0D 0A                                               ..
[0000]   50 72 6F 74 6F 63 6F 6C   20 6D 69 73 6D 61 74 63   Protocol  mismatc
[0010]   68 2E 0A                                            h..
```

Each transmission is dumped separately. There is a break and the counter at the left starts over each time there is a new send.

Access Control

A listening Ncat may control which hosts connect to it with the --allow and --deny options. Each of these takes a comma-separated list of host specifications. The syntax is almost identical to that recognized by Nmap for targets (see the section called “Target Specification”). This includes IPv4 and IPv6 addresses, hostnames, IPv4 octet ranges, and CIDR netmasks. In Ncat (unlike Nmap), CIDR netmasks are supported for IPv6 addresses.

With --allow, any hosts matching one of the listed specifiers are allowed and all others are denied. With --deny, those hosts matching the list are denied and all others are accepted. If a host matches both the --allow and --deny lists, it is denied.

Use --allowfile and --denyfile to allow or deny a list of host/network specifiers stored in a file. Each line of the file contains a specification in one of the forms listed above. Any file acceptable to Nmap's -iL and --excludefile options is suitable for --allowfile and --denyfile.

The following example commands demonstrate various kinds of access control.

Allow one host, deny all others

```
ncat -l --allow 192.168.0.125
ncat -l --allow 2001:db8::7d
ncat -l --allow trusted.example.com
```

Deny one host, allow all others

```
ncat -l --deny 192.168.0.200
ncat -l --deny 2001:db8::c8
```

Allow hosts on a local network, deny all others

```
ncat -l --allow 192.168.0.0/24
ncat -l --allow 192.168.0.0-255
ncat -l --allow 2001:db8::/32
```

Allow or deny hosts from a file

```
ncat -l --allowfile trusted-hosts.txt
ncat -l --denyfile external-hosts.txt
```

Be aware that host-based access control is susceptible to spoofing attacks and various other possible failures. These mechanisms should not be relied on for complete security.

Another kind of access control is simply limiting the maximum number of connections a listening Ncat will accept. Use the --max-conns option or its -m alias to do that. The default maximum number of connections is 100.

```
ncat -l --max-conns 5
```

Proxying

Ncat can route its connections through a SOCKS 4 or HTTP proxy. A basic connection looks like

```
ncat --proxy <proxyhost>[:<proxyport>] --proxy-type  { http  |   socks4 } <host> [<port>]
```

--proxy-type may be omitted; it defaults to http. If <proxyport> is omitted, it defaults to the well-known port for the chosen proxy type: 1080 for SOCKS and 3128 for HTTP. An exception to this rule is when the proxy host is given by a IPv6 address; in this case the port is required because otherwise it would be ambiguous whether the digits after the last colon are the port number or part of the address.

If the proxy server requires authentication, use the --proxy-auth option. Use --proxy-auth <username>:<password> for HTTP proxies and --proxy-auth <username> for SOCKS proxies.

Ncat can act as a proxy server itself in listen mode. The only proxy type supported is http.

```
ncat -l 3128 --proxy-type http
ncat -l 3128 --proxy-type http --proxy-auth <user>:<pass>
```

In listen mode the proxy port number is not automatically set and will be the default of 31337 unless specified. The proxy supports the GET, HEAD, and POST methods used in web browsing, as well as the CONNECT method that allows tunneling arbitrary TCP connections. (When Ncat connects as a client, it uses CONNECT.) Use --proxy-auth to make the server require authentication with a specific username and password. Be aware that the server uses the insecure Basic authentication mode and the credentials will be vulnerable to being intercepted. The proxy server uses the fork system call, so it is not supported on Windows.

Warning: Ncat's HTTP proxy is designed to stay out of your way and help you make temporary network connections. It shouldn't be used as an everyday proxy exposed to the Internet. You can limit who connects using --allow, --deny, and --proxy-auth, but these are not strong forms of authentication. An unauthenticated proxy is dangerous because it may enable others to perform attacks or help them evade detection. The CONNECT capability is especially dangerous because it enables any kind of traffic, not just HTTP.

Other Options

This section contains descriptions of all options that haven't been discussed so far.

The --nodns option (and its short form -n) instructs Ncat never to resolve names into addresses. All hosts must appear as IPv4 or IPv6 addresses.

Ncat can be used as a Telnet client or server with the --telnet option (-t). This simply causes Ncat to respond negatively to any questions asked by the other host in the binary Telnet protocol, removing such negotiations from the stream seen by the user. The primary use of this option is to allow running canned Telnet scripts.

The --send-only and --recv-only options do what their names imply, turning Ncat into a one-way communications channel instead of its default two-way channel. A usage example is gathering data from a server without the possibility of accidentally sending something typed at the keyboard. --send-only in both connect and listen modes causes Ncat to quit when its input runs out. Normally it will not quit until the network connection is closed because the remote side may still send something, but in the case of --send-only there's no reason to receive anything more.

Source Options

In connect mode, you may set the source address and port used for the connection with the --source (-s) and --source-port (-p). The -s option only works for locally configured addresses; it doesn't work like Nmap's -S option. The value of -p is that sometimes firewalls will allow traffic that comes from certain source ports (such as 20 or 53).

The -g option allows hops selection for IPv4 loose source routing. List the hops in order by giving -g multiple times or by separating the hops with commas. By default the source routing pointer is 4 in the packets sent, indicating the first hop in the list. You may set the pointer to another value with the -G option. The pointer value must be a multiple of 4 between 4 and 28, but some operating systems only support 4.

Timing

Ncat offers various options to control timing. Each of them take an argument that is assumed to be in milliseconds, unless followed by “s” for seconds, “m” for minutes, or “h” for hours. “30s” means 30 seconds. This format should already be familiar to Nmap users.

The --delay option and its short form -d make Ncat wait the given amount of time between each discrete read or write operation. For example, --delay 500 enforces a delay of half a second.
The --idle-timeout option and it synonym -i allow setting a timeout for reads and writes in connect mode. If the client fails to read or write for the given time period, the connection is dropped. These options do not work in listen mode.

The --wait (or -w for short) option sets how long Ncat will wait for a connection to be established in connect mode. The default is 10 seconds.

File Transfer

There is no shortage of ways to transfer a file over a network. Most file transfers are ably handled by email, network file systems, HTTP, SFTP, or other protocols. What do you do, though, when that file is too big to email, the transfer is between two machines not connected to the Internet, or you just need to do one quick file transfer without having to set up and tear down a file server? In these and other situations Ncat can be the right tool for the job. Some tricky file transfer scenarios can really make you appreciate the flexibility of a raw network pipe.

As you know, Ncat by default sends all its traffic without encryption, so it is possible for someone to intercept files in transit. See the section called “SSL” for one method of encrypting traffic.

By default, Ncat doesn't close its connection until it is closed by the remote end, even after it has exhausted its input. That is because (as far as Ncat knows) the remote server may still have data to send back. The --send-only option, when applicable, changes this behavior to close the connection and quit at the end of input. This is normally what you want when doing a one-way file transfer.

A basic file transfer is very simple: Start Ncat in listen mode on one end, start Ncat in connect mode on the other end, and pipe the file over the connection. There are two ways to do this that differ only in which end listens, the sender or the receiver. Sometimes you can't create a listening socket on one end of the transfer because of a lack or permissions, NAT, or filtering. As long as you can listen on at least one end, though, you can use this technique.

These examples show how to transfer inputfile on host1 to outputfile on host2. Here no port number was specified so Ncat will use its default port of 31337. To use a different port just list it on the command line.

Transfer a file, receiver listens

```
host2$ ncat -l > outputfile
host1$ ncat --send-only host2 < inputfile
```

Transfer a file, sender listens

```
host1$ ncat -l --send-only < inputfile
host2$ ncat host1 > outputfile
```

Note the order of the commands. The listener must be started first, regardless of the direction of transfer, or else the client will not have anything to connect to.

The above technique works fine for sending a single file. One way to send multiple files is to bundle them up with tar or zip and send the archive file. But there's an even easier way. Just pipe the output of tar directly into Ncat on the sending side, and pipe Ncat's output into tar on the receiving side. This is especially useful when the sending computer doesn't have enough free disk space to hold the archive file. Here's how to transfer <files> using the “receiver listens” method, though of course the “sender listens” method works just as well.

Transfer a bundle of files

```
host2$ ncat -l | tar xzv
host1$ tar czv <files> | ncat --send-only host2
```

Not only tar files but any stream of bytes can be transferred in this way. Here is an example of transferring an entire disk image from host1 to host2. Naturally, the disk should be unmounted or mounted read-only.

Transfer a disk image

```
host2$ ncat -l > host1-hda.image
host1$ ncat --send-only host2 < /dev/hda
```

Disk images are typically large files that take a long time to transfer. You can compress the image on the fly while sending and decompress it on the other end. Whether this makes an improvement depends on the speed of the network and the compression program.

Transfer a disk image with compression

```
host2$ ncat -l | bzip2 -d > host1-hda.image
host1$ cat /dev/hda | bzip2 | ncat --send-only host2
```

The basic file transmission technique described at the beginning of this section fails if neither participating host is capable of listening, or the two hosts can't communicate directly. This situation has become common with the prevalence of network address translation. A way to work around it is to use a third host as an intermediary. The intermediate host listens in connection brokering mode and the other two hosts connect to it. Recall from the section called “Connection Brokering” that in connection brokering mode any input received on one socket is copied and sent out to all other sockets. With just two hosts connected this is especially simple: anything coming from one host gets forwarded to the other. This example shows host1 sending inputfile to outputfile on host2, using host3 as an intermediary.

Transfer a file through an intermediary

```
host3$ ncat -l --broker
host2$ ncat host3 > outputfile
host1$ ncat --send-only host3 < inputfile
```

Note that it's important for host2 (the receiving host) to connect to the broker before host1 (the sending host) does. The broker does not buffer received data to send to hosts that connect later. After the file is transferred, it is necessary to forcible disconnect the Ncat on host2 with ctrl+C. The broker never disconnects any of its clients.

Chatting

In its most basic form, Ncat simply moves bits from one place to another. This is all that is needed to set up a simple chat system. By default, Ncat reads from standard input and writes to standard output, meaning that it will send whatever is typed at the keyboard and will show on the screen whatever is received.

Two-user chat

```
host1$ ncat -l
host2$ ncat host1
```

With this setup, two users can communicate with each other. Whatever one types will appear on the screen of the other. Be aware that standard input is probably line-buffered so it may be necessary to press enter before a line is sent. Which side listens and which side connects is not important in this situation, except that the listener must start ncat first.

The above technique is limited to one-on-one conversations. If more users connect to the server, each one will effectively create a new chat channel with the server; none of the connecting users will hear each other. Multi-user chatting is easily supported using connection brokering with the --broker option (see the section called “Connection Brokering”). In broker mode, anything received on one connection is sent out to all other connections, so everyone can talk with everyone else.

When many users are chatting through a connection broker, it can be hard to know who is saying what. For these cases Ncat provides a simple hack to tell users apart. When the --chat option is given, connection brokering is automatically enabled. Each message received is prefixed with an ID before being relayed to all other clients. The ID is unique for each client connection, and therefore functions something like a username. Also, in chat mode any control characters are escaped so they won't mess up your terminal. The server is started with

```
server$ ncat -l –chat
```

Once the server is started, this is how the chat appears to one of the connected users. The lines that begin with <user<n>> are from other connected users. The line beginning with <user0> was sent by the listening broker.

```
client$ ncat server
<user6> Is anyone there?
I'm here.
<user5> Me too.
<user0> Go away, all of you.
```

The user IDs generated by Ncat are based on the file descriptor for each connection, and must be considered arbitrary. There is no way to choose a particular ID or make one persist across sessions. Nevertheless, --chat can come in handy for those quick multi-user conversations.

Neat Tricks

Send Mail

It is great fun to interact with text-based network protocols with nothing more than Ncat and a keyboard. Here's a short example showing how to send email by talking to an SMTP server. SMTP is described in RFC 5321, but you don't need to know much about the protocol to send a simple message. The service's assigned port number is 25, and we use -C because it requires CRLF line endings. Example 5 contains a transcript of a session.

Example 5
Ncat as mail client

```
$ ncat -C mail.example.com 25
220 mail.example.com ESMTP
HELO client.example.com
250 mail.example.com Hello client.example.com
MAIL FROM:a@example.com
250 OK
```

```
RCPT TO:b@example.com
250 Accepted
DATA
354 Enter message, ending with "." on a line by itself
From: a@example.com
To: b@example.com
Subject: Greetings from Ncat

Hello. This short message is being sent by Ncat.
.
250 OK
QUIT
221 mail.example.com closing connection
```

To make this example work for you, change mail.example.com to your SMTP server and client.example.com to your domain name. Naturally you'll want to change the email addresses and message too. It will likely only work when using your normal mail server with your real email address, or when using the recipient's mail server (look up the MX record for the domain name in their email address).

Obviously this technique can be used for more than just sending mail. Ncat is a great interactive debugging tool for any text-based protocol. Such debugging is sometimes done with the telnet command, because it provides something like a raw text stream. Ncat offers a few advantages over telnet, though. Ncat doesn't print anything except what is sent by the remote host. Telnet isn't suitable for arbitrary binary data because it reserves some bytes as control characters. The telnet command quits when its input runs out, so you may not see what the other end sends. And finally, telnet doesn't do UDP.

Chain Ncats Together

Ncat is designed to work within a pipeline, so naturally the output of one instance of Ncat can be fed into the input of another. Here is one way to send a log file from host1 to host3 by way of host2:

```
host3$ ncat -l > log.txt
host2$ ncat -l | ncat host3
host1$ ncat --send-only host2 < log.txt
```

A possible problem with this technique is that it is one-way: host1 can send to host3 but there is no way for host3 to send anything back to host1. In this case it doesn't matter, but it can be done with a small change. Consider this:

```
host3$ ncat -l > log.txt
host2$ ncat -l --sh-exec "ncat host3"
host1$ ncat --send-only host2 < log.txt
```

The Ncat listening on host2, upon receiving a connection, creates a new Ncat to speak to host3 and connects the inputs and outputs of the programs running on host1 and host3 together. The same trick can be used on the local host too. This example forwards the local port 8080 to the web server on example.org:

```
ncat -l localhost 8080 --sh-exec "ncat example.org 80"
```

Unwrap SSL

Suppose you need to connect to an IMAP server that requires SSL, but your mail reader doesn't support SSL. Ncat can act as the encrypted bridge to connect the client and server. You will connect the mail client to a local port and Ncat will forward the traffic, encrypted, to the server. Here's how to connect IMAP (port 143) on the local host to IMAP over SSL (port 993) on imap.example.com.

```
ncat -l localhost 143 --sh-exec "ncat --ssl imap.example.com 993"
```

Once this is in place, instruct the mail client to connect to the IMAP server on localhost.

This trick works for protocols that pass traffic strictly between two hosts. It doesn't work well for HTTP because HTTP is usually aware of hostnames and often involves multiple hosts.

Use SSH Through an Ncat Tunnel

With Ncat and OpenSSH you can SSH to a host behind a NAT router without having to forward ports on the router. The router must have Ncat installed. Here is how to SSH to <host> through <router>:

```
ssh -o ProxyCommand="ssh -q <router> ncat %h %p" <host>
```

The ProxyCommand option of ssh tells how to open the SSH connection to <host>. It does this by opening another SSH session to <router> and connecting it to <host> with Ncat.

Watch What Nmap's Version Detection is Doing

Ncat can show you at a low level what's going on when Nmap version-scans a service. We'll make a service that only listens and instruct Nmap to use every version probe in the book. Set up Ncat to listen and record a hex dump log. The --keep-open option will make Ncat keep listening and accepting more connections after the first one is finished, contrary to the normal listen mode behavior of quitting when the first connection ends. Some version probes are binary so redirect standard output to /dev/null to avoid writing them to the screen.

```
ncat -l --keep-open 5200 --hex-dump vscan.log > /dev/null
```

Now scan the open port you made:

```
nmap -d -sV --version-all localhost -p 5200
```

An except of the hex dump is shown in Example 6.

Example 6
Hex dump of Nmap version detection

```
[0000]   0D 0A 0D 0A                                          ....
[0000]   47 45 54 20 2F 20 48 54   54 50 2F 31 2E 30 0D 0A   GET / HT TP/1.0..
[0010]   0D 0A                                                ..
[0000]   4F 50 54 49 4F 4E 53 20   2F 20 48 54 54 50 2F 31   OPTIONS  / HTTP/1
[0010]   2E 30 0D 0A 0D 0A                                    .0....
[0000]   4F 50 54 49 4F 4E 53 20   2F 20 52 54 53 50 2F 31   OPTIONS  / RTSP/1
[0010]   2E 30 0D 0A 0D 0A                                    .0....
[0000]   80 00 00 28 72 FE 1D 13   00 00 00 00 00 00 00 02   ...(r... ........
[0010]   00 01 86 A0 00 01 97 7C   00 00 00 00 00 00 00 00   .......| ........
[0020]   00 00 00 00 00 00 00 00   00 00 00 00               ........ ....
[0000]   00 1E 00 06 01 00 00 01   00 00 00 00 00 00 07 76   ........ .......v
[0010]   65 72 73 69 6F 6E 04 62   69 6E 64 00 00 10 00 03   ersion.b ind.....
```

```
[0000]   00 0C 00 00 10 00 00 00   00 00 00 00 00 00          ........ ......
[0000]   45 48 4C 4F 0D 0A                                    EHLO..
[0000]   48 45 4C 50 0D 0A                                    HELP..
```

At the beginning, Nmap would have sent its NULL probe, which isn't shown in the log file because the NULL probe doesn't send anything. At the top of the log is the GenericLines probe (0D 0A 0D 0A, or \r\n\r\n). After that is our old friend the HTTP GET request. Then come all the other probes in the nmap-service-probes file. In this excerpt are shown probes designed to get a response from RPC, DNS, and SMTP.

Emulating Diagnostic Services

There are a number of simple Internet protocols intended for testing and measurement purposes. Because they deal with simple, fundamental network operations they are a good match for Ncat's capabilities. This section shows how to to emulate services of increasing complexity: discard, echo, daytime, qotd, and chargen. These particular commands assume you are on a UNIX system such as Linux or Mac OS X, and using a /bin/sh compatible shell, such as Bash.

The discard service, defined in RFC 863, simply ignores anything sent to it. It runs on TCP or UDP port 9. By default, Ncat doesn't send any information unless instructed to, so nothing special is needed to emulate discard. Send Ncat's output to /dev/null to avoid filling the screen with characters received, or just let it write to the terminal if you're curious to see what's there. Use the --recv-only option to prohibit sending any characters that might be entered at the terminal.

TCP discard server

```
ncat -l --keep-open 9 --recv-only > /dev/null
```

UDP discard server

```
ncat -l 9 --udp --sh-exec "cat > /dev/null"
```

With the TCP server we used --keep-open so the server could handle multiple simultaneous connections, not just one. For the UDP server we had to use --sh-exec to allow multiple conncurrent connections. Recall from the section called “Basic usage” that a UDP server can handle only one client but with --exec and --sh-exec this limitation does not apply.

The echo service is defined in RFC 862. It runs on TCP or UDP port 7. One step more advanced than discard, it sends back any data received until the connection is closed. How do you instruct Ncat to return what it receives? One easy way is to run everything through /bin/cat.

TCP echo server

```
ncat -l 7 --exec "/bin/cat"
```

UDP echo server

```
ncat -l 7 --udp --exec "/bin/cat"
```

The daytime service, defined in RFC 867, sends a human-readable date and time string over TCP or UDP port 13. It ignores any input. The format of the date and time string is left unspecified, so we are free to use the output of /bin/date. Because we are not interested in anything sent by the client we use the --send-only option.

TCP daytime server

```
ncat -l 13 --send-only --exec "/bin/date"
```

UDP daytime server

```
ncat -l 13 --udp --send-only --exec "/bin/date"
```

Nmap comes with a daytime.nse script that works with the daytime service. Here is its output running against Ncat daytime servers on TCP and UDP.

Example 7
daytime.nse against an Ncat daytime server

```
# nmap -sSU -p 13 --script=daytime localhost
Starting Nmap ( http://nmap.org )

Interesting ports on localhost (127.0.0.1):
PORT   STATE SERVICE
13/tcp open  daytime
|_ daytime: Mon Jan 19 17:43:18 MST 2009
13/udp open  daytime
|_ daytime: Mon Jan 19 17:43:18 MST 2009

Nmap done: 1 IP address (1 host up) scanned in 0.31 seconds
```

The qotd (quote of the day) service is defined in RFC 865. When a connection is made to TCP or UDP port 17, it sends back a short message, ignoring any input. Ncat can do this by invoking a program that generates messages. A traditional choice is /usr/games/fortune, though there are many possibilities. /usr/bin/uptime, for example, could be useful.

TCP qotd server

```
ncat -l 17 --send-only --exec "/usr/games/fortune"
```

UDP qotd server

```
do ncat -l 17 --udp --send-only --exec "/usr/games/fortune"
```

In this example it's instructive to consider the difference between ncat -l 17 --exec "/usr/games/fortune" and /usr/games/fortune | ncat -l 17. Think about why the second command stops working after the first connection.

The chargen service from RFC 864 rounds out our tour of diagnostic services. It runs on TCP and UDP port 19. With TCP, chargen ignores any input and sends a never-ending stream of data. Never-ending, that is, until the connection is closed by the user, who the RFC suggests may have “had enough”. There are many ways of generating the characters; reading from /dev/zero and running yes come to mind.

TCP chargen server

```
yes "chargenchargenchargen" | ncat -l --keep-open 19 --send-only
```

Notice that in this case the program pipes its output into ncat rather than being invoked with --exec. For chargen either method would work, because the output of yes never changes. Using a pipe requires only one process other than ncat, but all users connected simultaneously will see the same output stream in synchrony. If the contents must be independent for each stream, then use the --exec method, with the understanding that a new process will be started for each connection.

The UDP chargen protocol is a little different. When a datagram is received, it sends back one datagram containing a random number of characters. Implementing this is starting to get away from Ncat, but one way it could be done with the Bash shell is this:

UDP chargen server

```
ncat -l 19 --udp --send-only --sh-exec \
  "yes chargenchargenchargen | dd count=1 bs=$(($RANDOM % 512)) 2> /dev/null"
```

Notice the use of --sh-exec rather than --exec to allow the use of the shell's environment variables and arithmetic evaluation. Standard error is redirected to /dev/null to avoid including dd's summary lines (1+0 records out), which would otherwise be included by Ncat.

This completes the tour of simple diagnostic services. These have been easy to implement with Ncat because (with the exception of UDP chargen) they all map directly onto a familiar command-line program. As services become more complex it gets harder to do everything in the shell. For complicated services it's better to write a separate program and have Ncat exec it directly.