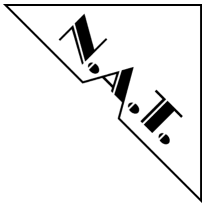
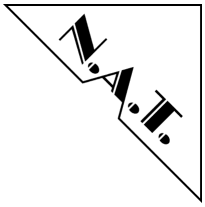


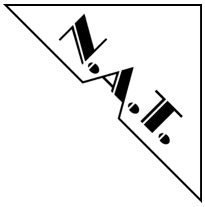
TCP/IP – User's Guide
Version 1.6



N.A.T. GmbH
Kamillenweg 22
D-53757 Sankt Augustin

Phone: ++49/2241/3989-0
Fax: ++49/2241/3989-10

E-Mail: support@nateurope.com
Internet: http://www.nateurope.com



Disclaimer

The following documentation, compiled by N.A.T. GmbH (henceforth called N.A.T.), represents the current status of the products development. The documentation is updated on a regular basis. Any changes which might ensue, including those necessitated by updated specifications, are considered in the latest version of this documentation. N.A.T. is under no obligation to notify any person, organisation, or institution of such changes or to make these changes public in any other way.

We must caution you, that this publication could include technical inaccuracies or typographical errors.

N.A.T. offers no warranty, either expressed or implied, for the contents of this documentation or for the product described therein, including but not limited to the warranties of merchantability or the fitness of the product for any specific purpose.

In no event, will N.A.T. be liable for any loss of data or for errors in data utilisation or processing resulting from the use of this product or the documentation. In particular, N.A.T. will not be responsible for any direct or indirect damages (including lost profits, lost savings, delays or interruptions in the flow of business activities, including but not limited to, special, incidental, consequential, or other similar damages) arising out of the use of or inability to use this product or the associated documentation, even if N.A.T. or any authorised N.A.T. representative has been advised of the possibility of such damages.

The use of registered names, trademarks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations (patent laws, trade mark laws, etc.) and therefore free for general use. In no case does N.A.T. guarantee that the information given in this documentation is free of such third-party rights.

This product (and the associated documentation) is governed by the N.A.T. General Conditions and Terms of Delivery and Payment.

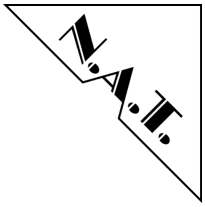
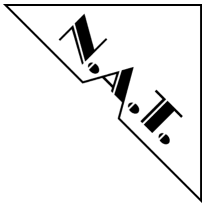
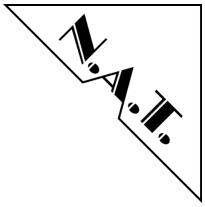


Table of Contents



1 Introduction

Thank you for choosing our product. N.A.T. - TCP/IP is a network software package for systems running under OS-9. The software and functions conform in largest part to those of the University of California, Berkeley UNIX implementation.
This manual must be understood as a supplement to the "N.A.T. Installation Guide" and the "N.A.T. Socket Library Guide". These manuals contain information which is generally the same for the different network protocols implemented by N.A.T. To keep things user friendly, the installation and configuration procedure (covered in the "N.A.T. Installation Guide") and the programming interface of the network software (covered in the "N.A.T. Socket Library Guide") have been designed to be independent of the underlying protocol (TCP/IP or ISO/OSI, for instance).
This manual, however, deals with topics specific to TCP/IP. It explains the usage of network utilities such as ftp and telnet (see section 2, "Socket Interface") and the correspondent server programs (see section 3, "Server Programs").
For the sake of brevity, this guide assumes a certain basic knowledge of TCP/IP networks and is not intended as a tutorial. For details specific to the OS-9 operating system, please refer to your OS-9 manuals.



1.1 Utilities

The N.A.T. - TCP/IP package currently includes the following utilities:

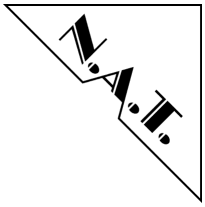
telnet	Terminal emulation program
ftp	File Transfer Program
ifconfig	Program used to configure interfaces and protocols
netstat	Program used to display the network status and the network and protocol statistics
genidb	Program used to generate a data module for a network data basis
dumpidb	Program used to view the entries in a network data basis data module

Plese refer to section 2, "Socket Interface", to get detailed information on these utilities.

1.2 Protocols

The package presently supports all of the following protocols:

TCP	Transport Control Protocol supports a secure connection-oriented data transfer
UDP	User Datagram Protocol supports a fast unsecured connectionless data transfer
IP	Internet Protocol supports the Internet routing algorithm and address recognition
ICMP	Internet Control Message Protocol supports the exchange of network error and status messages
ARP	Address Resolution Protocol translates Internet addresses into Ethenet addresses



1.3 Function Libraries

N.A.T. - TCP/IP includes three function libraries.

1.3.1 socklib.l

The socklib.l library supports, in addition to a a fully compatible BSD socket interface, several other important Berkeley-UNIX functions. In particular, the following functions are supported:

accept()	accept a request for a socket connection
bind()	bind a name to a socket
close()	close a connection to a socket
connect()	initiate a connection to a socket
gethostname()	returns the name of your own host system
getpeername()	returns the socket name of the connected system
getsockname()	returns the name of your own socket
getsockopt()	permits reading the values of a socket´s options
ioctl()	permits manipulation of device parameters
listen()	receives the attempt to establish a socket
read()	reads data from a socket
recv()	reads data from a socket
recvfrom()	receives data from a socket
recvmsg()	receives a message from a socket
select()	simultaneously supervises several I/O paths
send()	writes data via a socket
sendx()	extended send() call (only on FDDI29/ETH29)
sendmsg()	writes a message via socket
sendto()	writes data via a socket
setsockopt()	set socket options
shutdown()	terminates a portion of a full-duplex connection
socket()	creates a communications end-point (socket)
SysReqMem()	allocate system memory (only on FDDI29/ETH29)
SysRetMem()	return system memory (only on FDDI29/ETH29)
write()	write data to a socket

Please refer to section 2, "Socket Interface", in the "N.A.T. Socket Library Guide" to get detailed information on how to use these functions in your own programs.

1.3.2 netdb.l

The netdb.l library contains the functions needed to permit access from the application programs to the network data basis. Two different versions are provided. The first version of the library directly accesses the files hosts, networks, protocols, and services, and must be located in the directory /DD/ETC. The second version of the library retrieves its information from the data module that was created from the previously named files using the utility genidb. Both libraries contain the following functions:

To read host entries

gethostbyname()
gethostbyaddr()
gethostent()
sethostent()
endhostent()

To read network entries

getnetbyaddr()
getnetbyname()
getnetent()
setnetent()
endnetent()

To read protocol entries

getprotobyaddr()
getprotobyname()
getprotoent()
setprotoent()
endprotoent()

To read services entries

getservbyaddr()
getservbyname()
getservent()
setservent()
endservent()

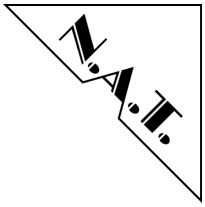
Please refer to section 5, "Network Data Basis", in the "N.A.T. Socket Library Guide" to get detailed information on how to use these functions in your own programs.

1.3.3 inetdb.l

The inetdb.l library contains following functions to enable manipulation of Internet addresses:

inet_addr()	Generates a complete numerical Internet address from a string containing an Internet address in the standard Dot-notation.
inet_network()	Generates an Internet network number from a string containing an Internet address in the standard Dot-notation.
inet_ntoa()	Generates a string containing an Internet address in the standard Dot-notation from a numerical Internet network address.
inet_makeaddr()	Generates a complete numerical Internet address by combining one part containing the network number with a second part providing the host number.
inet_lnaof()	Extracts the host number from a numerical Internet address.
inet_netof()	Extracts the network number from a numerical Internet address.

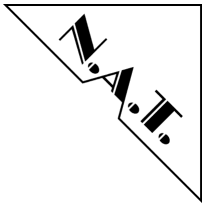
Please refer to section 4, "Address Manipulation Functions", in the "N.A.T. Socket Library Guide" to get detailed information on how to use these functions in your own programs.



1.4 Networking - An Introduction

This section briefly describes the networking facilities available in the system. This section is broken up into three areas: protocol families (domains), protocols, and network interfaces. Since books on any one of these topics may fill an entire shelf in the local book stores, the treatment of the material in this guide is naturally brief.
All network protocols are associated with a specific protocol family. A protocol family provides basic services to the protocol implementation to allow it to function within a specific network environment. These services may include packet fragmentation and reassembly, routing, addressing, and basic transport. A protocol family may support multiple methods of addressing, though the current protocol implementations do not. A protocol family is normally comprised of a number of protocols, one per socket() type. It is not required that a protocol family support all socket types. A protocol family may contain multiple protocols supporting the same socket abstraction.
A protocol supports one of the socket abstractions detailed in socket(). A specific protocol may be accessed either by creating a socket of the appropriate type and protocol family, or by requesting the protocol explicitly when creating a socket. Protocols normally accept only one type of address format, usually determined by the addressing structure inherent in the design of the protocol family/network architecture. Certain semantics of the basic socket abstractions are protocol specific.
All protocols are expected to support the basic model for their particular socket type, but may, in addition, provide non-standard facilities or extensions to a mechanism. For example, a protocol supporting the SOCK_STREAM abstraction may allow more than one byte of out-of-band data to be transmitted per out-of-band message.
A network interface is similar to a device interface. Network interfaces comprise the lowest layer of the networking subsystem, interacting with the actual transport hardware. An interface may support one or more protocol families and/or address formats.

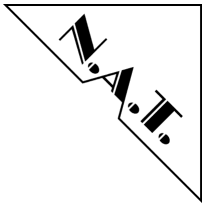
PROTOCOLS
The system currently supports the DARPA Internet protocol. A raw socket interface is provided to the IP protocol layer of the DARPA Internet. Consult the appropriate manual pages in this section for more information regarding the support for each protocol family.



ADDRESSING

Associated with each protocol family is an address format. The following address formats are used by the system (and additional formats are defined for possible future implementation):

```
#define AF_UNSPEC       0       /* unspecified */
#define AF_OS9          1       /* local to host (pipes, portals) */
#define AF_INET         2       /* internetwork: UDP, TCP, etc. */

#define AF_IMPLINK      3       /* arpanet imp addresses */
#define AF_PUP          4       /* pup protocols: e.g. BSP */
#define AF_CHAOS        5       /* CHAOS protocols */
#define AF_NS           6       /* Xerox NS protocols */
#define AF_NBS          7       /* nbs protocols */
#define AF_ECMA         8       /* european computer manufacturers */
#define AF_DATAKIT      9       /* datakit protocols */
#define AF_CCITT        10      /* CCITT protocols X.25 etc. */
#define AF_SNA          11      /* IBM SNA */
#define AF_DECnet       12      /* DECnet */
#define AF_DLI          13      /* Direct data link interface */

#define AF_LAT          14      /* LAT */
#define AF_HYLINK       15      /* NSC Hyperchannel */
#define AF_APPLETALK    16      /* AppleTalk */
```



ROUTING

The network facilities provided limited packet routing. A simple set of data structures comprise a ``routing table" used in selecting the appropriate network interface when transmitting packets. This table contains a single entry for each route to a specific network or host. A user process, the routing daemon, maintains this data base with the aid of two socket-specific ioctl() commands, SIOCADDRT and SIOCDELRT. The commands allow the addition and deletion of a single routing table entry, respectively. Routing table manipulations may only be carried out by the super-user.

A routing table entry has the following form, as defined in <net/route.h>;

```
struct rtentry {
        u_long          rt_hash;                /*to speed lookups*/
        struct          sockaddr rt_dst;        /* key */
        struct          sockaddr rt_gateway;    /* value */
        short           rt_flags;               /* up/down?, host/net */
        short           rt_refcnt;              /* # held references*/
        u_long          rt_use;                 /*raw # packets forwarded */
        struct          ifnet *rt_ifp;          /* answer:interface to use */
};
```

with rt_flags defined as follows:

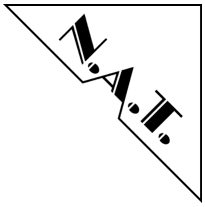
```
#define         RTF_UP          0x1     /* route usable */
#define         RTF_GATEWAY     0x2     /* destination is a gateway */
#define         RTF_HOST        0x4     /* host entry (net otherwise) */
#define         RTF_DYNAMIC     0x10    /* created dynamically  */
                                        /* (by redirect) */
```

Routing table entries come in three flavors:

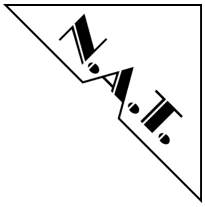
- for a specific host,
- for all hosts on a specific network,
- for any destination not matched

by entries of the first two types (a wildcard route). When the system is booted and addresses are assigned to the network interfaces, each protocol family installs a routing table entry for each interface when it is ready for traffic. Normally the protocol specifies the route through each interface as a "direct" connection to the destination host or network.

If the route is direct, the transport layer of a protocol family usually requests the packet be sent to the same host specified in the packet. Otherwise, the interface is requested to address the packet to the gateway listed in the routing entry (i.e. the packet is forwarded).



Routing table entries installed by a user process may not specify the hash, reference count, use, or interface fields; these are filled in by the routing routines. If a route is in use when it is deleted (rt_refcnt is non-zero), the routing entry will be marked down and removed from the routing table, but the resources associated with it will not be reclaimed until all references to it are released. The routing code returns E_EXIST if requested to duplicate an existing entry, E_SRCH if requested to delete a non-existent entry, or E_NOBUFS if insufficient resources were available to install a new route. The rt_use field contains the number of packets sent along the route.
When routing a packet, the kernel will first attempt to find a route to the destination host. Failing that, a search is made for a route to the network of the destination. Finally, any route to a default ("wildcard") gateway is chosen. If multiple routes are present in the table, the first route found will be used. If no entry is found, the destination is declared to be unreachable.
A wildcard routing entry is specified with a zero destination address value. Wildcard routes are used only when the system fails to find a route to the destination host and network. The combination of wildcard routes and routing redirects can provide an economical mechanism for routing traffic.

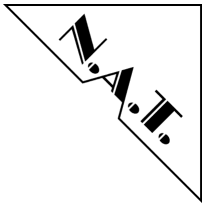


INTERFACES

Each network interface in a system corresponds to a path through which messages may be sent and received. A network interface usually has a hardware device associated with it, though certain interfaces such as the loopback interface, lo(), do not.

The following ioctl() calls may be used to manipulate network interfaces. The ioctl() is made on a socket (typically of type SOCK_DGRAM) in the desired domain. Unless specified otherwise, the request takes an ifrequest structure as its parameter. This structure has the form

```
struct ifreq {
        #define  IFNAMSIZ 16
        char ifr_name[IFNAMSIZE];       /* if name of interface,  "nat0" */
        union {
                struct   sockaddr ifru_addr;
                struct   sockaddr ifru_dstaddr;
                struct   sockaddr ifru_broadaddr;
                short    ifru_flags;
                int ifru_metric;
                caddr_t  ifru_data;
         } ifr_ifru;
         #define  ifr_addr          ifr_ifru.ifru_addr          /* address */
         #define  ifr_dstaddr       ifr_ifru.ifru_dstaddr       /* other end of p-to-p link */
         #define  ifr_broadaddr     ifr_ifru.ifru_broadaddr     /* broadcast address */
         #define  ifr_flags         ifr_ifru.ifru_flags         /* flags */
         #define  ifr_metric              ifr_ifru.ifru_metric         /* metric */
         #define  ifr_data          ifr_ifru.ifru_data          /* for use by interface */
};
```



SIOCSIFADDR	Set interface address for protocol family. Following the address assignment, the "initialization" routine for the interface is called.
SIOCGIFADDR	Get interface address for protocol family.
SIOCSIFDSTADDR	Set point to point address for protocol family and interface.
SIOCGIFDSTADDR	Get point to point address for protocol family and interface.
SIOCSIFBRDADDR	Set broadcast address for protocol family and interface.
SIOCGIFBRDADDR	Get broadcast address for protocol family and interface.
SIOCSIFFLAGS	Set interface flags field. If the interface is marked down, any processes currently routing packets through the interface are notified; some interfaces may be reset so that incoming packets are no longer received. When marked up again, the interface is reinitialized.
SIOCGIFFLAGS	Get interface flags.
SIOCSIFMETRIC	Set interface routing metric. The metric is used only by user-level routers.
SIOCGIFMETRIC	Get interface metric.
SIOCGIFCONF	Get interface configuration list. This request takes an ifconf structure (see below) as a value-result parameter. The ifc_len field should be initially set to the size of the buffer pointed to by ifc_buf. On return it will contain the length, in bytes, of the configuration list.

```
/*
 * Structure used in SIOCGIFCONF request.
 * Used to retrieve interface configuration
 * for machine (useful for programs which
 * must know all networks accessible).
 */
struct   ifconf {
    int ifc_len;                                    /* size of associated buffer */
    union {
            caddr_t         ifcu_buf;
            struct          ifreq *ifcu_req;
    } ifc_ifcu;
#define  ifc_buf            ifc_ifcu.ifcu_buf   /* buffer address */
#define  ifc_req            ifc_ifcu.ifcu_req   /* array of structures returned */
};
```

SEE ALSO
icmp, ifconfig, inet, ioctl(), ip, tco, udp

2 Utilities

This chapter describes the following utilities which are currently supplied in the N.A.T.-TCP/IP package:

ftp	File Transfer Protocol file transfer program to exchange files between systems in the network
tftp	Trivial File Transfer Protocol file transfer program to exchange files between systems in the network
telnet	interactive terminal services program
ping	program to test a connection to another computer in the network
ifconfig	configuration program for interfaces and protocols
netstat	program to display the network status as well as the network and protocol statistics

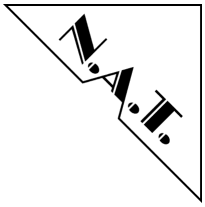
2.1 Internet Addresses in Dot Notation

Internet addresses represented in dot notation can take one of the following four forms:

a.b.c.d	If four parts are given, each part will be interpreted as a single byte. Each part (byte will then be ordered from left to right to the four bytes of a Internet address.
a.b.c	If tree parts are given, the right-most part (c) will be interpreted as a 16-bit unit and will be assigned to the two right-most bytes of the Internet address. This agrees with the class B network address form 128.Net.Host.
a.b	If two parts are given, the last part (b) will be interpreted as a 24-bit unit and will be assigned to the 3 right-most bytes of the Internet address. This agrees with the class A network address form Net.Host.
a	If only a single part is given it will be treated as a direct entry of the Internet address.

In dot notation all the parts of an Internet address can be entered in decimal, octal, or hexadecimal format. The syntax is the same as for the "C" programming language, thus:

- a leading 0x or 0X signifies that the next digit should be interpreted as hexadecimal,
- a leading 0 (zero) signifies that the next digit should be interpreted as octal, and
- all other cases should be treated as being in decimal notation.



2.2 ftp

NAME

ftp - ARPANET file transfer program

SYNOPSIS

ftp [-v] [-d] [-i] [-n] [-g] [host]

DESCRIPTION

ftp is the user interface to the ARPANET standard File Transfer Protocol. The program allows a user to transfer files to and from a remote network site.
The client host with which ftp is to communicate may be specified on the command line. If this is done, ftp will immediately attempt to establish a connection to an FTP server on that host; otherwise, ftp will enter its command interpreter and await instructions from the user. When ftp is awaiting commands from the user the prompt ftp> is provided to the user.

The following commands are recognized by ftp:

! [*command* [*args*]]

Invoke an interactive shell on the local machine. If there are arguments, the first is taken to be a command to execute directly, with the rest of the arguments as its arguments.

$ *macro-name* [*args*]

Execute the macro macro-name that was defined with the macdef command. Arguments are passed to the macro unglobbed.

account [*passwd*]

Supply a supplemental password required by a remote system for access to resources once a login has been successfully completed. If no argument is included, the user will be prompted for an account password in a non-echoing input mode.

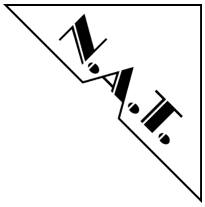
append *local-file* [*remote-file*]

Append a local file to a file on the remote machine. If remote-file is left unspecified, the local file name is used in naming the remote file, after being altered by any ntrans or nmap setting.

File transfer uses the current settings for type, format, mode, and structure.

ascii

Set the file transfer type to network ASCII. This is the default type.



bell
Arrange that a bell be sounded after each file transfer command is completed.

binary
Set the file transfer type to support binary image transfer.

bye
Terminate the FTP session with the remote server and exit ftp. An end of file will also terminate the session and exit.

case
Toggle remote computer file name case mapping during mget commands. When case is on (default is off), remote computer file names with all letters in upper case are written in the local directory with the letters mapped to lower case.

cd *remote-directory*
Change the working directory on the remote machine to remote-directory.

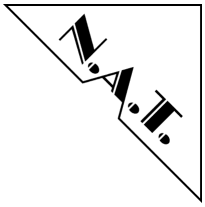
cdup
Change the remote machine working directory to the parent of the current remote machine working directory.

close
Terminate the FTP session with the remote server, and return to the command interpreter. Any defined macros are erased.

cr
Toggle carriage return stripping during ascii type file retrieval. Records are denoted by a carriage return/linefeed sequence during ascii type file transfer. When cr is on (the default), carriage returns are stripped from this sequence to conform with the UNIX single linefeed record delimiter. Records on non-UNIX remote systems may contain single linefeeds; when an ascii type transfer is made, these linefeeds may be distinguished from a record delimiter only when cr is off.

debug [*debug-value*]
Toggle debugging mode. If an optional debug-value is specified it is used to set the debugging level.
When debugging is on, ftp prints each command sent to the remote machine, preceded by the string "-->".

delete *remote-file*
Delete the file remote-file on the remote machine.



dir [*remote-directory*] [*local-file*]
Print a listing of the directory contents in the directory, remote-directory, and, optionally, placing the output in local-file. If interactive prompting is on, ftp will prompt the user to verify that the last argument is indeed the target local file for receiving dir output. If no directory is specified, the current working directory on the remote machine is used. If no local file is specified, or local-file is -, output comes to the terminal.

disconnect
A synonym for close.

form *format*
Set the file transfer form to format. The default format is file.

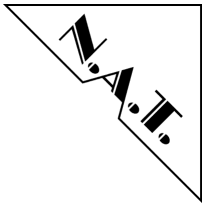
get *remote-file* [*local-file*]
Retrieve the remote-file and store it on the local machine. If the local file name is not specified, it is given the same name it has on the remote machine, subject to alteration by the current case, ntrans, and nmap settings. The current settings for type, form, mode, and structure are used while transferring the file.

glob
Toggle filename expansion for mdelete, mget and mput. If globbing is turned off with glob, the file name arguments are taken literally and not expanded. Globbing for mput is done as in csh(1).
For mdelete and mget, each remote file name is expanded separately on the remote machine and the lists are not merged. Expansion of a directory name is likely to be different from expansion of the name of an ordinary file: the exact result depends on the foreign operating system and ftp server, and can be previewed by doing `mls remote-files -'. Note: mget and mput are not meant to transfer entire directory subtrees of files. That can be done by transferring a tar(1) archive of the subtree (in binary mode).

hash
Toggle hash-sign ("#") printing for each data block transferred. The size of a data block is 1024 bytes.

help [*command*]
Print an informative message about the meaning of command. If no argument is given, ftp prints a list of the known commands.

lcd [*directory*]
Change the working directory on the local machine. If no directory is specified, the user's home directory is used.



lf

Toggles the "linefeed" filter used while receiving ASCII files on/off. Records are normally terminated with a <CRLF> (carriage return/linefeed) during the transmission. If lf is set ON (the default setting), the linefeeds will be stripped when the file is saved on the local system´s disk so that the record terminator matches the OS-9 conventions.

ls [*remote-directory*] [*local-file*]

Print an abbreviated listing of the contents of a directory on the remote machine. If remote-directory is left unspecified, the current working directory is used. If interactive prompting is on, ftp will prompt the user to verify that the last argument is indeed the target local file for receiving ls output. If no local file is specified, or if local-file is -, the output is sent to the terminal.

macdef *macro-name*

Define a macro. Subsequent lines are stored as the macro macro-name; a null line (consecutive newline characters in a file or carriage returns from the terminal) terminates macro input mode. There is a limit of 16 macros and 4096 total characters in all defined macros. Macros remain defined until a close command is executed. The macro processor interprets `$' and `\' as special characters. A `$' followed by a number (or numbers) is replaced by the corresponding argument on the macro invocation command line. A `$' followed by an `i' signals that macro processor that the executing macro is to be looped. On the first pass `$i' is replaced by the first argument on the macro invocation command line, on the second pass it is replaced by the second argument, and so on. A `\' followed by any character is replaced by that character. Use the `\' to prevent special treatment of the `$'.

mdelete [*remote-files*]

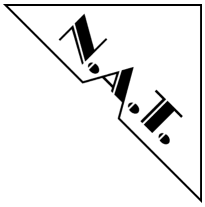
Delete the remote-files on the remote machine.

mdir *remote-files local-file*

Like dir, except multiple remote files may be specified. If interactive prompting is on, ftp will prompt the user to verify that the last argument is indeed the target local file for receiving mdir output.

mget *remote-files*

Expand the remote-files on the remote machine and do a get for each file name thus produced. See glob for details on the filename expansion.
Resulting file names will then be processed according to case, ntrans, and nmap settings. Files are transferred into the local working directory, which can be changed with `lcd directory'; new local directories can be created with `! mkdir directory'.



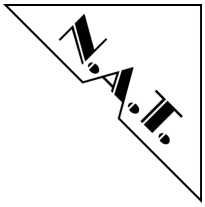
mkdir *directory-name*
Make a directory on the remote machine.

mls *remote-files local-file*
Like ls, except multiple remote files may be specified. If interactive prompting is on, ftp will prompt the user to verify that the last argument is indeed the target local file for receiving mls output.

mode [*mode-name*]
Set the file transfer mode to mode-name. The default mode is stream mode.

mput *local-files*
Expand wild cards in the list of local files given as arguments and do a put for each file in the resulting list. See glob for details of filename expansion. Resulting file names will then be processed according to ntrans and nmap settings.

nmap [*inpattern outpattern*]
Set or unset the filename mapping mechanism. If no arguments are specified, the filename mapping mechanism is unset. If arguments are specified, remote filenames are mapped during mput commands and put commands issued without a specified remote target filename. If arguments are specified, local filenames are mapped during mget commands and get commands issued without a specified local target filename. This command is useful when connecting to a non-UNIX remote computer with different file naming conventions or practices. The mapping follows the pattern set by inpattern and outpattern. Inpattern is a template for incoming filenames (which may have already been processed according to the ntrans and case settings). Variable templating is accomplished by including the sequences `$1', `$2', ..., `$9' in inpattern. Use `\' to prevent this special treatment of the `$' character. All other characters are treated literally, and are used to determine the nmap inpattern variable values. For example, given inpattern $1.$2 and the remote file name "mydata.data", $1 would have the value "mydata", and $2 would have the value "data". The outpattern determines the resulting mapped filename. The sequences `$1', `$2', ..., `$9' are replaced by any value resulting from the inpattern template. The sequence `$0' is replace by the original filename. Additionally, the sequence `[seq1,seq2P]' is replaced by seq1 if seq1 is not a null string; otherwise it is replaced by seq2.
For example, the command "nmap $1.$2.$3 [$1,$2].[$2,file]" would yield the output filename "myfile.data" for input filenames "myfile.data" and "myfile.data.old", "myfile.file" for the input filename "myfile", and "myfile.myfile" for the input filename ".myfile". Spaces may be included in outpattern, as in the example: nmap $1 |sed "s/ *$//" > $1 . Use the `\' character to prevent special treatment of the `$', `[`, `]', and `,' characters.



ntrans [*inchars* [*outchars*]]

Set or unset the filename character translation mechanism. If no arguments are specified, the filename character translation mechanism is unset.

If arguments are specified, characters in remote filenames are translated during mput commands and put commands issued without a specified remote target filename. If arguments are specified, characters in local filenames are translated during mget commands and get commands issued without a specified local target filename. This command is useful when connecting to a non-UNIX remote computer with different file naming conventions or practices.

Characters in a filename matching a character in inchars are replaced with the corresponding character in outchars. If the character's position in inchars is longer than the length of outchars, the character is deleted from the file name.

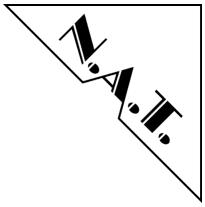
open *host* [*port*]

Establish a connection to the specified host FTP server. An optional port number may be supplied, in which case, ftp will attempt to contact an FTP server at that port. If the auto-login option is on (default), ftp will also attempt to automatically log the user in to the FTP server (see below).

prompt

Toggle interactive prompting. Interactive prompting occurs during multiple file transfers to allow the user to selectively retrieve or store files.

If prompting is turned off (default is on), any mget or mput will transfer all files, and any mdelete will delete all files.



proxy *ftp-command*

Execute an ftp command on a secondary control connection. This command allows simultaneous connection to two remote ftp servers for transferring files between the two servers. The first proxy command should be an open, to establish the secondary control connection. Enter the command "proxy ?" to see other ftp commands executable on the secondary connection.

The following commands behave differently when prefaced by proxy:

open	will not define new macros during the auto-login process,
close	will not erase existing macro definitions,
get and mget	transfer files from the host on the primary control connection to the host on the secondary control connection, and
put, mput, and append	transfer files from the host on the secondary control connection to the host on the primary control connection.

Third party file transfers depend upon support of the ftp protocol PASV command by the server on the secondary control connection.

put *local-file* [*remote-file*]

Store a local file on the remote machine. If remote-file is left unspecified, the local filename is used after processing according to any ntrans or nmap settings in naming the remote file. File transfer uses the current settings for type, format, mode, and structure.

pwd

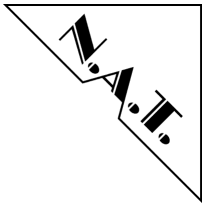
Print the name of the current working directory on the remote machine.

quit

A synonym for bye.

quote *arg1 arg2 ...*

The arguments specified are sent, verbatim, to the remote FTP server.



quote rcmd *<remote command>*

Start a process on the OS9-FTP-server.
On your FTP client, execute the command "quote help" to see all commands known to the OS9-FTP-server. The client command "quote help rcmd" shows you the following syntax of the OS9-FTP-server command "RCMD" :

RCMD <SP> remote command

If you want to start the process "my_process" on the OS9-FTP-server, for instance, you have to execute the following FTP client command :

>quote rcmd my_process

Following things have to be considered :

- At first, the process to be started is loaded into the module memory. For this reason, the correct file attributes for the module have to be set. Otherwise, the call of RCMD will immediately terminate with an FTP error.
- The process to be started "inherits" the environment of his father process which is the FTPD.
- The maximal length of the command line is 94 characters.

The OS9-FTP-Server can return one of the following messages:

- 500 Command line too long (max 94 chars).
  The command line exceeds 94 characters.
- 200 Call for command <cmd> successful, pid = <pid>.
  The process <cmd> with the process ID <pid> has been started successfully.
- 550 Call for command <cmd> failed.
  The process <cmd> could not be started. Possible reasons : no execution permission, wrong parameter list etc.

recv *remote-file* [*local-file*]

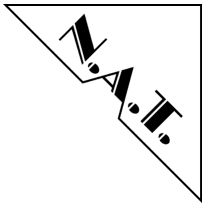
A synonym for get.

remotehelp [*command-name*]

Request help from the remote FTP server. If a command-name is specified it is supplied to the server as well.

rename [*from*] [*to*]

Rename the file from on the remote machine, to the file to.



reset
Clear reply queue. This command re-synchronizes command/reply sequencing with the remote ftp server. Resynchronization may be necessary following a violation of the ftp protocol by the remote server.

rmdir *directory-name*
Delete a directory on the remote machine.

runique
Toggle storing of files on the local system with unique filenames. If a file already exists with a name equal to the target local filename for a get or mget command, a ".1" is appended to the name.
If the resulting name matches another existing file, a ".2" is appended to the original name. If this process continues up to ".99", an error message is printed, and the transfer does not take place. The generated unique filename will be reported. Note that runique will not affect local files generated from a shell command (see below).
The default value is off.

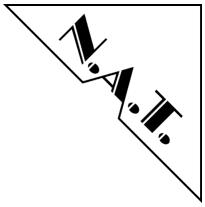
send *local-file* [*remote-file*]
A synonym for put.

sendport
Toggle the use of PORT commands. By default, ftp will attempt to use a PORT command when establishing a connection for each data transfer. The use of PORT commands can prevent delays when performing multiple file transfers. If the PORT command fails, ftp will use the default data port. When the use of PORT commands is disabled, no attempt will be made to use PORT commands for each data transfer.
This is useful for certain FTP implementations which do ignore PORT commands but, incorrectly, indicate they've been accepted.

status
Show the current status of ftp.

struct [*struct-name*]
Set the file transfer structure to struct-name. By default, file structure is used.

sunique
Toggle storing of files on remote machine under unique file names. Remote ftp server must support ftp protocol STOU command for successful completion. The remote server will report unique name.
Default value is off.



tenex
Set the file transfer type to that needed to talk to TENEX machines.

trace
Toggle packet tracing.

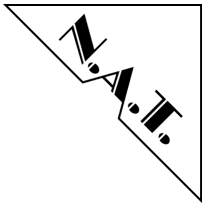
type [*type-name*]
Set the file transfer type to type-name. If no type is specified, the current type is printed.
The default type is network ASCII.

user *user-name* [*password*] [*account*]
Identify yourself to the remote FTP server. If the password is not specified and the server requires it, ftp will prompt the user for it (after disabling local echo). If an account field is not specified, and the FTP server requires it, the user will be prompted for it. If an account field is specified, an account command will be relayed to the remote server after the login sequence is completed if the remote server did not require it for logging in. Unless ftp is invoked with auto-login disabled, this process is done automatically on initial connection to the FTP server.

verbose
Toggle verbose mode. In verbose mode, all responses from the FTP server are displayed to the user. In addition, if verbose is on, when a file transfer completes, statistics regarding the efficiency of the transfer are reported. By default, verbose is on.

? [*command*]
A synonym for help.

Command arguments which have embedded spaces may be quoted with quote (") marks.



ABORTING A FILE TRANSFER

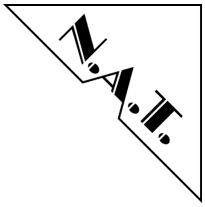
To abort a file transfer, use the terminal interrupt key (usually <Ctrl-C>). Sending transfers will be immediately halted. Receiving transfers will be halted by sending a ftp protocol ABOR command to the remote server, and discarding any further data received. The speed at which this is accomplished depends upon the remote server's support for ABOR processing. If the remote server does not support the ABOR command, an "ftp>" prompt will not appear until the remote server has completed sending the requested file.

The terminal interrupt key sequence will be ignored when ftp has completed any local processing and is awaiting a reply from the remote server. A long delay in this mode may result from the ABOR processing described above, or from unexpected behavior by the remote server, including violations of the ftp protocol. If the delay results from unexpected remote server behavior, the local ftp program must be killed by hand.

FILE NAMING CONVENTIONS

Files specified as arguments to ftp commands are processed according to the following rules.

- If the file name - is specified, the stdin (for reading) or stdout (for writing) is used.
- If the first character of the file name is |, the remainder of the argument is interpreted as a shell command. Ftp then forks a shell and reads (writes) from the stdout (stdin). If the shell command includes spaces, the argument must be quoted; e.g. "| ls -lt". A particularly useful example of this mechanism is: dir |more.
- Failing the above checks, if "globbing" is enabled, local file names are expanded; c.f. the glob command. If the ftp command expects a single local file (.e.g. put), only the first filename generated by the "globbing" operation is used.
- For mget commands and get commands with unspecified local file names, the local filename is the remote filename, which may be altered by a case, ntrans, or nmap setting. The resulting filename may then be altered if runique is on.
- For mput commands and put commands with unspecified remote file names, the remote filename is the local filename, which may be altered by a ntrans or nmap setting. The resulting filename may then be altered by the remote server if sunique is on.



Expanding Filenames - Globbing

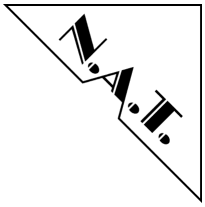
The same mechanisms are used to expand filenames as are by the UNIX C-Shell. If a filename contains a "*", "?", "[", or a "{" or if it begins with a "~", it is a candidate for expansion (also known as "globbing"). The filename is then used as a template will be replaced by an alphabetically sorted list of files which fit this pattern. If a filename template is given that does not find a match with any of the existing filenames, an error is returned. However, it is not necessary that every potential pattern contained in the template find a match in the existing filenames. Only the metacharacters "*", "?", "[", and "{" imply pattern matching.

*****	matches any character string including an empty string or string of zero length
?	matches any single character
[nat]	matches any character "n", "a", or "t"
[a-z]	matches any lower-case character from "a" to "z" inclusively
b{i,a,oo}m	matches bim, bam, boom The order in which the contents are given within the curly brackets will be kept. If metacharacters are used within the curly brackets, the matched files will be alphabetically sorted.

The character "~" is simply an abbreviation for the home directory as it is entered in the file /DD/SYS/password. If the character "~" is entered by itself, the home directory of the user who called ftp will be used. If, on the other hand, the "~" is entered together with a user name, ftp will search the file /DD/SYS/password for an entry under this user name and then (assuming that an entry is found) use the associated home directory. For example:

- If ftp was started by user "uwelin", then the filename **~/SOURCE/xxx.c** will be expanded to
  /DD/UWELIN/SOURCE/xxx.c.

- If the filename **~/BOGART/DOC/MOVIE/yyy.doc** is entered with the valid user name bogart, it will be expanded to
  /DD/BOGART/DOC/MOVIE/yyy.doc.



FILE TRANSFER PARAMETERS

The FTP specification specifies many parameters which may affect a file transfer. The type may be one of ascii, image (binary), ebcdic, and local byte size (for PDP-10's and PDP-20's mostly). ftp supports the ascii and image types of file transfer, plus local byte size 8 for tenex mode transfers.

ftp supports only the default values for the remaining file transfer parameters:

- mode "stream",
- form "non-print", and
- struct "file".

OPTIONS

Options may be specified at the command line, or to the command interpreter. These are the available options:

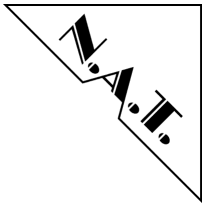
-v The -v (verbose on) option forces ftp to show all responses from the remote server, as well as report on data transfer statistics.

-n The -n option restrains ftp from attempting auto-login upon initial connection. If auto-login is enabled, ftp will check the .netrc (see below) file in the user's home directory for an entry describing an account on the remote machine. If no entry exists, ftp will prompt for the remote machine login name (default is the user identity on the local machine), and, if necessary, prompt for a password and an account with which to login.

-i The -i option turns off interactive prompting during multiple file transfers.

-d The -d option enables debugging.

-g The -g option disables file name globbing.



THE .netrc FILE

The .netrc file contains login and initialization information used by the auto-login process. It resides in the user's home directory. The following tokens are recognized; they may be separated by spaces, tabs, or newlines:

machine *name*

Identify a remote machine name. The auto-login process searches the .netrc file for a machine token that matches the remote machine specified on the ftp command line or as an open command argument. Once a match is made, the subsequent .netrc tokens are processed, stopping when the end of file is reached or another machine token is encountered.

login *name*

Identify a user on the remote machine. If this token is present, the auto-login process will initiate a login using the specified name.

password *string*

Supply a password. If this token is present, the auto-login process will supply the specified string if the remote server requires a password as part of the login process. Note that if this token is present in the .netrc file, ftp will abort the autologin process if the .netrc is readable by anyone besides the user.

account *string*

Supply an additional account password. If this token is present, the auto-login process will supply the specified string if the remote server requires an additional account password, or the auto-login process will initiate an ACCT command if it does not.

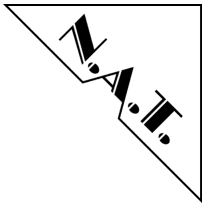
macdef *name*

Define a macro. This token functions like the ftp macdef command functions. A macro is defined with the specified name; its contents begin with the next .netrc line and continue until a null line (consecutive new-line characters) is encountered.

If a macro named init is defined, it is automatically executed as the last step in the auto-login process.

SEE ALSO

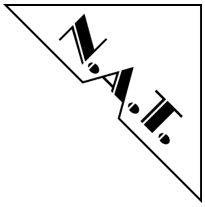
ftpd



BUGS

Correct execution of many commands depends upon proper behavior by the remote server.

The command mput requires the directory /DD/TMP. Temporary files are written in this directory and then later deleted using the function unlink(). Since OS-9 unfortunately still doesn't support multiple links to files, it can happen that these files will not be deleted. You must delete these files by hand from time to time so that mput can create new temporary files.



2.3 telnet

NAME

telnet - user interface to the TELNET protocol

SYNOPSIS

telnet [*host* [*port*]]

DESCRIPTION

telnet is used to communicate with another host using the TELNET protocol. If telnet is invoked without arguments, it enters command mode, indicated by its prompt (telnet>). In this mode, it accepts and executes the commands listed below. If it is invoked with arguments, it performs an open command (see below) with those arguments.

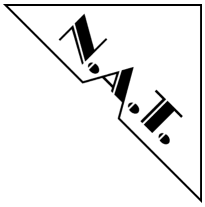
Once a connection has been opened, telnet enters an input mode. The input mode entered will be either character at a time or line by line depending on what the remote system supports.

In character at a time mode, most text typed is immediately sent to the remote host for processing.

In line by line mode, all text is echoed locally, and (normally) only completed lines are sent to the remote host. The local echo character (initially ^E) may be used to turn off and on the local echo (this would mostly be used to enter passwords without the password being echoed).

In either mode, if the localchars toggle is TRUE (the default in line mode; see below), the user's quit, intr, and flush characters are trapped locally, and sent as TELNET protocol sequences to the remote side. There are options (see toggle autoflush and toggle autosynch below) which cause this action to flush subsequent output to the terminal (until the remote host acknowledges the TELNET sequence) and flush previous terminal input (in the case of quit and intr).

While connected to a remote host, telnet command mode may be entered by typing the telnet escape character (initially ^]). When in command mode, the normal terminal editing conventions are available.



COMMANDS

The following commands are available. Only enough of each command to uniquely identify it need be typed (this is also true for arguments to the mode, set, toggle, and display commands).

open *host* [*port*]

Open a connection to the named host. If no port number is specified, telnet will attempt to contact a TELNET server at the default port. The host specification may be either a host name or an Internet address specified in the dot notation.

close

Close a TELNET session and return to command mode.

quit

Close any open TELNET session and exit telnet. An end of file (<EOF> in command mode) will also close a session and exit.

mode *type*

Type is either line (for line by line mode) or character (for character at a time mode). The remote host is asked for permission to go into the requested mode. If the remote host is capable of entering that mode, the requested mode will be entered.

status

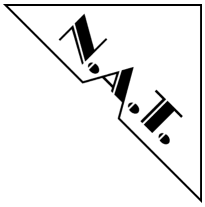
Show the current status of telnet. This includes the peer one is connected to, as well as the current mode.

display [*argument...*]

Displays all, or some, of the set and toggle values (see below).

? [*command*]

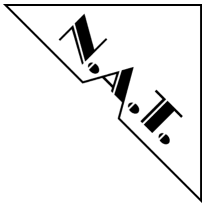
Get help. With no arguments, telnet prints a help summary. If a command is specified, telnet will print the help information for just that command.



send *arguments*

Sends one or more special character sequences to the remote host. The following are the arguments which may be specified (more than one argument may be specified at a time):

escape	Sends the current telnet escape character (initially ^]).
synch	Sends the TELNET SYNCH sequence. This sequence causes the remote system to discard all previously typed (but not yet read) input. This sequence is sent as TCP urgent data (and may not work if the remote system is a 4.2 BSD system -- if it doesn't work, a lower case "r" may be echoed on the terminal).
brk	Sends the TELNET BRK (Break) sequence, which may have significance to the remote system.
ip	Sends the TELNET IP (Interrupt Process) sequence, which should cause the remote system to abort the currently running process.
ao	Sends the TELNET AO (Abort Output) sequence, which should cause the remote system to flush all output from the remote system to the user's terminal.
ayt	Sends the TELNET AYT (Are You There) sequence, to which the remote system may or may not choose to respond.
ec	Sends the TELNET EC (Erase Character) sequence, which should cause the remote system to erase the last character entered.
el	Sends the TELNET EL (Erase Line) sequence, which should cause the remote system to erase the line currently being entered.
ga	Sends the TELNET GA (Go Ahead) sequence, which likely has no significance to the remote system.
nop	Sends the TELNET NOP (No OPeration) sequence.
?	Prints out help information for the send command.



set *argument value*

Set any one of a number of telnet variables to a specific value. The special value off turns off the function associated with the variable. The values of variables may be interrogated with the display command. The variables which may be specified are:

echo This is the value (initially ^E) which, when in line by line mode, toggles between doing local echoing of entered characters (for normal processing), and suppressing echoing of entered characters (for entering, say, a password).

escape This is the telnet escape character (initially ^[) which causes entry into telnet command mode (when connected to a remote system).

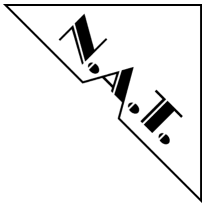
interrupt If telnet is in localchars mode (see toggle localchars below) and the interrupt character is typed, a TELNET IP sequence (see send ip above) is sent to the remote host. The initial value for the interrupt character is taken to be the terminal's intr character.

quit If telnet is in localchars mode (see toggle localchars below) and the quit character is typed, a TELNET BRK sequence (see send brk above) is sent to the remote host. The initial value for the quit character is taken to be the terminal's quit character.

flushoutput If telnet is in localchars mode (see toggle localchars below) and the flushoutput character is typed, a TELNET AO sequence (see send ao above) is sent to the remote host. The initial value for the flush character is taken to be the terminal's flush character.

erase If telnet is in localchars mode (see toggle localchars below), and if telnet is operating in character at a time mode, then when this character is typed, a TELNET EC sequence (see send ec above) is sent to the remote system. The initial value for the erase character is taken to be the terminal's erase character.

kill If telnet is in localchars mode (see toggle localchars below), and if telnet is operating in character at a time mode, then when this character is typed, a TELNET EL sequence (see send el above) is sent to the remote system. The initial value for the kill character is taken to be the terminal's kill character.



eof If telnet is operating in line by line mode, entering this character as the first character on a line will cause this character to be sent to the remote system. The initial value of the eof character is taken to be the terminal's eof character.

toggle *arguments...*

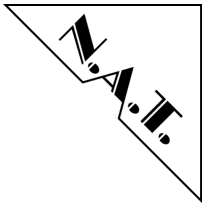
Toggle (between TRUE and FALSE) various flags that control how telnet responds to events. More than one argument may be specified. The state of these flags may be interrogated with the display command. Valid arguments are:

localchars If this is TRUE, then the flush, interrupt, quit, erase, and kill characters (see set above) are recognized locally, and transformed into (hopefully) appropriate TELNET control sequences (respectively ao, ip, brk, ec, and el; see send above). The initial value for this toggle is TRUE in line by line mode, and FALSE in character at a time mode.

autoflush If autoflush and localchars are both TRUE, then when the ao, intr, or quit characters are recognized (and transformed into TELNET sequences; see set above for details), telnet refuses to display any data on the user's terminal until the remote system acknowledges (via a TELNET Timing Mark option) that it has processed those TELNET sequences. The initial value for this toggle is TRUE if the terminal user had not done an "stty noflsh", otherwise FALSE.

autosynch If autosynch and localchars are both TRUE, then when either the intr or quit characters is typed (see set above for descriptions of the intr and quit characters), the resulting TELNET sequence sent is followed by the TELNET SYNCH sequence. This procedure should cause the remote system to begin throwing away all previously typed input until both of the TELNET sequences have been read and acted upon. The initial value of this toggle is FALSE.

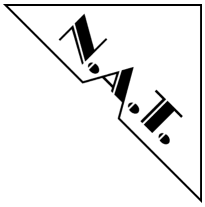
crmod Toggle carriage return mode. When this mode is enabled, most carriage return characters received from the remote host will be mapped into a carriage return followed by a linefeed. This mode does not affect those characters typed by the user, only those received from the remote host. This mode is not very useful unless the remote host only sends carriage return, but never line feed. The initial value for this toggle is FALSE.



debug	Toggles socket level debugging (useful only to the superuser). The initial value for this toggle is FALSE.
options	Toggles the display of some internal telnet protocol processing (having to do with TELNET options). The initial value for this toggle is FALSE.
netdata	Toggles the display of all network data (in hexadecimal format). The initial value for this toggle is FALSE.
?	Displays the legal toggle commands.

Known Problems

- There is no proper handling of flow control (handshake).
- On some remote systems, you must manually turn off character echo when you are in line for line mode.
- In line for line mode, the eof character is only recognized and sent to the remote system when it is the first character on a line.



2.4 tftp

NAME

tftp - ARPANET Trivial File Transfer Program

SYNOPSIS

tftp [*host*]

DESCRIPTION

The program tftp provides the user with an interface to the Internet TFTP (Trivial File Transfer Protocol). The program enables a simple transfer of files between systems within the network.

The remote system (with which you wish to exchange files) can be specified in the command line. If this option is used, the system specified on the command line will be used by tftp as the default host for all following file transfers.

Once tftp is started, it will display the prompt tftp>, and then the following commands may be used:

? [*command*]

Display the help record for the specified command.

ascii

short form for "mode ascii"

binary

short form for "mode binary"

connect *host_name* [*port*]

Set the host (and optionally the port) for a file transfer. The variable "default-host" will be set to the system specified in host_name.

Please note: in contrast to ftp, tftp does not create a true logical connection between the two systems for a file transfer. The connect command can be skipped, since both the get and put commands permit the entry of the remote system name.

get *filename*
get *remote_filename local_filename*
get *file_1 file_2 ... file_n remote_directory*

Copy the specified file(s) from the remote system. The file names can be given in either of the two following forms:

- simple file name
  if the desired remote system is the current "default-host"

- path string - host:filename
  giving the name of the remote system (host) and the desired file name

If the second form is used, the variable "default-host" will be set to the name specified as host in this command.

mode *transfer_mode*

Set the file transfer mode. The transfer_mode can be either ascii or binary. The default mode is ascii.

put *filename*
put *local_filename remote_filename*
put *file_1 file_2 ... file_n remote_directory*

Transfer the specified file(s) to the remote system(s). The name of the remote system can be given in either of the two following forms:

- simple file name
  if the desired remote system is the current "default-host"

- path string - host:filename
  giving the name of the remote system (host) and the desired file name

If the second form is used, the variable "default-host" will be set to the name specified as host in this command.

quit

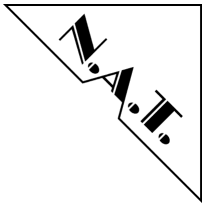
Terminate tftp. An <EOF> (End of File) character also terminates tftp.

rexmt *retransmission_timeout*

Set the retransmission timeout time (in seconds) for a single packet.

status

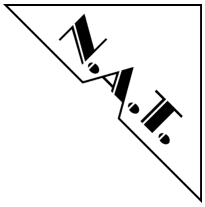
Display the current status of tftp.



timeout *total_transmission_timeout*
Set the timeout time (in seconds) for a completed file transfer.

trace
Toggle the "Tracing" of packets on/off.

verbose
Toggle the display of messages on/off.



2.5 ping

NAME

ping - send ICMP ECHO_REQUEST packets to network hosts

SYNOPSIS

ping [*-r*] [*-v*] *host* [*packetsize*] [*count*]

DESCRIPTION

The DARPA Internet is a large and complex aggregation of network hardware, connected together by gateways. Tracking a single-point hardware or software failure can often be difficult. Ping utilizes the ICMP protocol's mandatory ECHO_REQUEST datagram to elicit an ICMP ECHO_RESPONSE from a host or gateway. ECHO_REQUEST datagrams ("pings") have an IP and ICMP header, followed by a struct timeval, and then an arbitrary number of "pad" bytes used to fill out the packet. Default datagram length is 64 bytes, but this may be changed using the command-line option. Other options are:

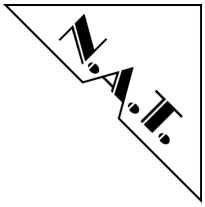
-r Bypass the normal routing tables and send directly to a host on an attached network. If the host is not on a directly-attached network, an error is returned. This option can be used to ping a local host through an interface that has no route through it (e.g., after the interface was dropped by routed).

-v Verbose output. ICMP packets other than ECHO_RESPONSE that are received are listed.

When using ping for fault isolation, it should first be run on the local host, to verify that the local network interface is up and running. Then, hosts and gateways further and further away should be "pinged". Ping sends one datagram per second, and prints one line of output for every ECHO_RESPONSE returned. No output is produced if there is no response. If an optional count is given, only that number of requests is sent. Round-trip times and packet loss statistics are computed. When all responses have been received or the program times out (with a count specified), or if the program is terminated with a SIGINT, a brief summary is displayed.
This program is intended for use in network testing, measurement and management. It should be used primarily for manual fault isolation. Because of the load it could impose on the network, it is unwise to use ping during normal operations or from automated scripts.

SEE ALSO

netstat, ifconfig



2.6 ifconfig

NAME

ifconfig - configure network interface parameters

SYNOPSIS

ifconfig *interface address_family* [*address* [*dest_address*]] [*parameters*]
ifconfig *interface* [*protocol_family*]

DESCRIPTION

ifconfig is used to assign an address to a network interface and/or configure network interface parameters. ifconfig must be used at boot time to define the network address of each interface present on a machine; it may also be used at a later time to redefine an interface's address or other operating parameters. The interface parameter is a string of the form "name unit", e.g. "nat0".
Since an interface may receive transmissions in differing protocols, each of which may require separate naming schemes, it is necessary to specify the address_family, which may change the interpretation of the remaining parameters. The only address family currently supported is "inet" and thus the DARPA Internet protocol is supported.
For the DARPA-Internet family, the address is either a host name present in the host name data base, hosts, or a DARPA Internet address expressed in the Internet standard "dot notation".

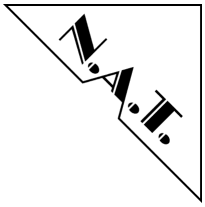
The following parameters may be set with ifconfig:

up

Mark an interface "up". This may be used to enable an interface after an ``ifconfig down." It happens automatically when setting the first address on an interface. If the interface was reset when previously marked down, the hardware will be reinitialized.

down

Mark an interface "down". When an interface is marked "down", the system will not attempt to transmit messages through that interface. If possible, the interface will be reset to disable reception as well. This action does not automatically disable routes using the interface.



trailers
Request the use of a "trailer" link level encapsulation when sending (default). If a network interface supports trailers, the system will, when possible, encapsulate outgoing messages in a manner which minimizes the number of memory to memory copy operations performed by the receiver. On networks that support the Address Resolution Protocol; currently, only 10 Mb/s Ethernet), this flag indicates that the system should request that other systems use trailers when sending to this host. Similarly, trailer encapsulations will be sent to other hosts that have made such requests. Currently used by Internet protocols only.

-trailers
Disable the use of a ``trailer" link level encapsulation.

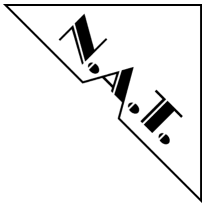
arp
Enable the use of the Address Resolution Protocol in mapping between network level addresses and link level addresses (default). This is currently implemented for mapping between DARPA Internet addresses and 10Mb/s Ethernet addresses.

-arp
Disable the use of the Address Resolution Protocol.

metric n
Set the routing metric of the interface to n, default 0. The routing metric is used by the routing protocol (routed). Higher metrics have the effect of making a route less favorable; metrics are counted as addition hops to the destination network or host.

debug
Enable driver dependent debugging code; usually, this turns on extra console error logging.

-debug
Disable driver dependent debugging code.



netmask *mask*

(Internet only) Specify how much of the address to reserve for subdividing networks into sub-networks. The mask includes the network part of the local address and the subnet part, which is taken from the host field of the address. The mask can be specified as a single hexadecimal number with a leading 0x, with a dot-notation Internet address, or with a pseudo-network name listed in the network table networks. The mask contains 1's for the bit positions in the 32-bit address which are to be used for the network and subnet parts, and 0's for the host part. The mask should contain at least the standard network portion, and the subnet field should be contiguous with the network portion.

dstaddr

Specify the address of the correspondent on the other end of a point to point link.

broadcast

(Internet only) Specify the address to use to represent broadcasts to the network. The default broadcast address is the address with a host part of all 1's.

ifconfig displays the current configuration for a network interface when no optional parameters are supplied. If a protocol family is specified, ifconfig will report only the details specific to that protocol family.

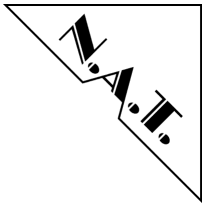
Only the super-user may modify the configuration of a network interface.

DIAGNOSTICS

Messages indicating the specified interface does not exit, the requested address is unknown, or the user is not privileged and tried to alter an interface's configuration.

SEE ALSO

arp, hosts, netstat, networks



2.7 netstat

NAME

netstat - show network status

SYNOPSIS

netstat [*-Aan*] [*-f address_family*]
netstat [*-mnrs*] [*-f address_family*] [*-p protocol*]
netstat [*-n*] [*-I interface*] *interval*

DESCRIPTION

The netstat command symbolically displays the contents of various network-related data structures. There are a number of output formats, depending on the options for the information presented. The first form of the command displays a list of active sockets for each protocol. The second form presents the contents of one of the other network data structures according to the option selected. Using the third form, with an interval specified, netstat will continuously display the information regarding packet traffic on the configured network interfaces. The fourth form displays statistics about the named protocol.

The options have the following meaning:

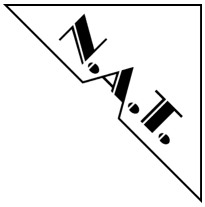
-A With the default display, show the address of any protocol control blocks associated with sockets; used for debugging.

-a With the default display, show the state of all sockets; normally sockets used by server processes are not shown. With either interface display (option -i or an interval, as described below), show the number of dropped packets.

-f *address_family* Limit statistics or address control block reports to those of the specified address family. Currently, only inet for the AF_INET address family is recognized.

-I *interface* Show information only about this interface; used with an interval as described below.

-m Show statistics recorded by the memory management routines (the network manages a private pool of memory buffers).



-n Show network addresses as numbers (normally netstat interprets addresses and attempts to display them symbolically). This option may be used with any of the display formats.

-p *protocol* Show statistics about protocol, which is either a well-known name for a protocol or an alias for it. Some protocol names and aliases are listed in the file /etc/protocols. A null response typically means that there are no interesting numbers to report. The program will complain if protocol is unknown or if there is no statistics routine for it.

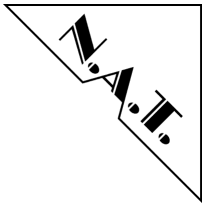
-r Show the routing tables. When -s is also present, show routing statistics instead.

-s Show protocol statistics for IP, ICMP, TCP, and UDP.

The default display, for active sockets, shows the local and remote addresses, send and receive queue sizes (in bytes), protocol, and the internal state of the protocol. Address formats are of the form "host.port" or "network.port" if a socket's address specifies a network but no specific host address. When known the host and network addresses are displayed symbolically according to the data bases /etc/hosts and /etc/networks, respectively. If a symbolic name for an address is unknown, or if the -n option is specified, the address is printed numerically, according to the address family. For more information regarding the Internet "dot format," refer to section 2.1, "Internet Addresses in Dot Notation". Unspecified, or "wildcard", addresses and ports appear as "*".

The interface display provides a table of cumulative statistics regarding packets transferred, errors, and collisions. The network addresses of the interface and the maximum transmission unit (``MTU") are also displayed.

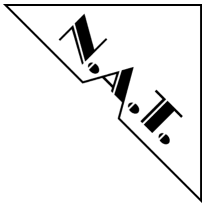
The routing table display indicates the available routes and their status. Each route consists of a destination host or network and a gateway to use in forwarding packets. The flags field shows the state of the route ("U" if "up"), whether the route is to a gateway ("G"), whether the route was created dynamically by a redirect ("D"), and whether the route has been modified by a redirect ("M"). Direct routes are created for each interface attached to the local host; the gateway field for such entries shows the address of the outgoing interface. The refcnt field gives the current number of active uses of the route. Connection oriented protocols normally hold on to a single route for the duration of a connection while connectionless protocols obtain a route while sending to the same destination. The use field provides a count of the number of packets sent using that route. The interface entry indicates the network interface utilized for the route.



When netstat is invoked with an interval argument, it displays a running count of statistics related to network interfaces. This display consists of a column for the primary interface (the first interface found during auto-configuration) and a column summarizing information for all interfaces. The primary interface may be replaced with another interface with the -I option. The first line of each screen of information contains a summary since the system was last rebooted. Subsequent lines of output show values accumulated over the preceding interval.

SEE ALSO

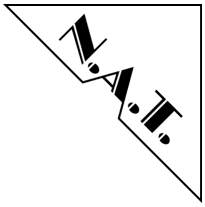
hosts, networks, protocols, services



3 Server Programs

This chapter covers the currently available server programs in detail.

bootpd	Boot Protocol Server
inetd	Internet "Super-Server"
ftpd	File Transfer Protocol Server
telnetd	TELNET Protocol Server
tftpd	Trivial File Transfer Protocol Server



3.1 bootpd

NAME

bootpd - Internet Boot Protocol Server

SYNOPSIS

bootpd [*-s -ttimeout -d*] [*configfile* [*dumpfile*]]

DESCRIPTION

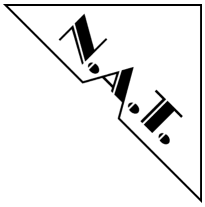
bootpd is the server process for the Internet boot protocol. It is normally started by the Internet "Super-Server", inetd. To enable this function, make the following entry in the file /DD/ETC/inetd.conf:

bootps dgram upd wait root /DD/TCPIP/CMDS/bootpd bootpd

Where /DD/TCPIP/CMDS is the directory in which the executable server program bootpd is located. If bootpd is located in a different directory on your system make the corresponding change to this entry.

The server bootpd will now be started whenever a request for it is received from a remote system. If bootpd does not receive another request for its services within 15 minutes following the last request, it will be removed from the running processes to save system resources. The option -t can be used to set the timeout period in minutes (e.g., -t20 to set the timeout to 20 minutes). Setting the timout period to 0 results in bootpd running continuously (it will never be killed).

bootpd can also be run in a stand-alone configuration by entering the option -s (for example in the startup file). This is a good solution in larger networks where many systems are accessing bootpd. When the option -s has been used, the timeout (-t) is disabled and bootpd is never terminated.
Each occurrence of the option -d in the command line raises the level for the debug output from bootpd.



When bootpd is started it first reads its configuration file /DD/ETC/bootpdab and begins listening for a BOOTREQUEST packet. The configuration file has a format similar to that of termcap. The configuration file contains of a number of parameter declarations consisting of 2 character symbols (so-called tags) and their associated values. The tag labels are case sensitive. All parameter declarations are separated by a colon (:). The general format is:

host_name:tg=value ... :tg=value ... :tg=value ...

Where host_name is the actual name of a bootp client and
tg is a tag symbol.

As shown above, most tags must be followed by a equals sign and the associated value. There are, however, tags which are boolean toggles and are therefore entered without an associated value. In the case of a boolean tag, only the tag symbol is found between the colons. The currently used tags are:

bf	boot file
bs	size of the boot file in 512 octet blocks
cs	address list of the cookie servers
ds	address list of the domain name servers
gw	address list of the gateways
ha	hardware address of the hosts
hd	home directory for the boot file
hn	boolean tag - send host name
ht	host´s hardware type
im	address list of the impress servers
ip	IP addresses of the hosts
lg	address list of the log servers
lp	address list of the LPR servers
ns	address list of the IEN-116-Name servers
rl	address list of resource location protocol servers
sm	host´s subnet mask
tc	table continuation (points to similar "template" host entry)
to	time offset in seconds from UTC
ts	address list of the time servers
vm	selector for the "vendor magic cookie"

The tags cs, ds, gw, im, lg, lp, ns, rl, and ts all take a list of IP addresses in which the individual entries are separated by a tab or spaces.
The tags ip and sm each take a single IP address.

All IP addresses must be entered in the Internet "Dot Notation" (see section 2.1, "Internet Addresses in Dot Notation").

The tag ht contains the hardware type code. This can be an unsigned decimal, octal, or hexadecimal value or one of the following symbolic names:

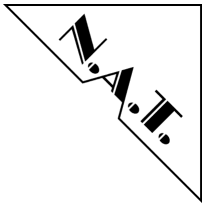
ethernet or **ether**	for 10MB Ethernet
ethernet3 or **ether3**	for 3MB Experimental Ethernet
ieee802, **tr**, or **token-ring**	for IEEE802 networks
pronet	for Proteon ProNET token ring
chaos	for Chaos networks
arcnet	for ARCNET networks
ax.25	for the amateur radio network

The value for the hardware address tag ha must be given in hexadecimal. An optional period or a leading "0x" may be added for better readability. The ha tag must precede an explicit or implicit ht tag (see below - tc).
The host name, the name of the boot file´s home directory, and the name of the boot file itself are ASCII strings and may (optionally) be enclosed within quotation marks (").
What the server will write into the boot file field of a reply is determined by the client´s query and the value of the tags hd and bf.

- If the client´s query contains an absolute pathname and
  - this file exists, this pathname will be repeated in the reply.
  - If the file is not found, no reply is sent.

- If the client´s query contains a relative pathname, a complete pathname will be created from this relative pathname and the value of the tag hd.
  - If the tag hd does not exist or
  - the file cannot be found, no reply will be sent.
  - If the client´s query doesn't contain a boot file name, the server will always reply. The contents of this reply depends on the contents of the tags hd and bf.
  - If the tag bf contains a complete pathname, the server will write this name in the reply.
  - Otherwise, if bf and hd combine to make a complete pathname to an accessible file, this name will be written by the server in the reply.
  - If it is not possible to create a complete pathname or the file cannot be found, the reply will be sent with an empty field for the boot file name.

In all of the above cases, "file exists" means not only that the file exists but also that the access rights have been set to public read so that tftpd is permitted to transfer the file.



By all files, the program will attempt to create a file name in the form filename.hostname. If this is not successful, the simple file name will be used. Thus, the use of individual host boot files is supported.

The time offset (to) tag can take either:

- a signed integer value for the offset (in seconds) of client´s time-zone as compared to the UTC, or
- the keyword auto - where the time-zone offset of the server is used. If to is used without an explicit value (boolean), the tag defaults to auto.

The size of the boot file (tag bs) can be given as:

- the integer value of the file size (in 512 octet blocks) and may be in decimal, octal, or hexadecimal, or
- the keyword auto - where the file size is recalculated by every client query. As by to, if bs is used without an explicit value (boolean), the tag defaults to auto.

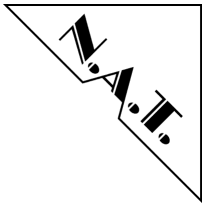
The tag vm can take either of 2 keywords:

- rfc1048 for replies in the RFC1048 style
- cmu for replies in the CMU style

The boolean tag hn, when present, signifies that the host name should be sent to RFC1048 clients. The server bootpd always attempts to send a complete host name. If this does not fit into the reply packet, it will attempt to send just the host part of the name (up to the first period, if present). If this also doesn't fit in the reply, no name will be sent. NO artificial abbreviation of the name will be made by the program. Please note: The tag hn is boolean and is not permitted a value assignment (=xxx).

Generally, many of the host entries contain identical entries for several of the tags. The tc tag provides a mechanism for eliminating the need to repeat all these entries. A host entry (often a dummy entry) is made containing a complete specification. The following entries use the tag tc to take the values defined in this preceding entry and then modify it with their own tag entries. This mechanism functions in much the same way as that of termcap for similar terminals.

Please note: In contrast to termcap, the tc tag in bootpd need not stand at the end of the entry, but may be located anywhere in the entry. Explicit entries for a host overwrite the implicit values given by tc regardless of their position in the entry. The value of the tag symbol tc can be the host name or the IP address that occurred earlier in the configuration file.
Sometimes it is necessary to delete a special tag symbol that was inherited by the tc mechanism. This can be done using the construction tag@, which blocks the effect of tag. For example, to delete the complete specification of an IEN-116 name server, enter ":ns@:" at the appropriate position within a configuration entry. After deleting the tag value using @, the tag can be assigned a new value using the tc mechanism.
In the configuration file, blank lines and lines that begin with "#" are ignored. Host entries are separated by a carriage return. A single host entry may be extended over several lines by terminating each line except for the last with a backslash (\). A line may exceed 80 characters. With two exceptions, the tags may be given in any order within an entry: one, the host name must be the first field within an entry; and two, the hardware type tag ht must precede the hardware address tag ha.



Sample **bootptab** file:

```
# Sample bootptab file
default1:\
        :hd=/DD/USR/BOOT:bf=null:\
        :ds=128.2.35.50 128.2.13.21:\
        :ns=0x80020b4d 0x80020ffd:\
        :ts=0x80020b4d 0x80020ffd:\
        :sm=255.255.0.0:gw=0x8002fe24:\
        :hn:vm=auto:to=-18000:

carnegie:ht=6:ha=7ff8100000af:ip=128.2.11.1:tc=default1:
baldwin:ht=1:ha=0800200159c3:ip=128.2.11.10:tc=default1:
wylie:ht=1:ha=00dd00cadf00:ip=128.2.11.100:tc=default1:
arnold:ht=1:ha=0800200102ad:ip=128.2.11.102:tc=default1:
baird:ht=1:ha=08002b02a2f9:ip=128.2.11.103:tc=default1:
baker:ht=1:ha=08002b0287c8:ip=128.2.11.104:tc=default1:

# Special domain name server for next host
butler:ht=1:ha=08002001560d:ip=128.2.11.108:\
        :ds=128.2.13.42:tc=default1
gaston:ht=6:ha=7fff81000a47:ip=128.2.11.115:tc=default1:
hahn:ht=6:ha=7fff81000434:ip=128.2.11.117:tc=default1:
hick:ht=6:ha=7fff810001ba:ip=128.2.11.118:tc=default1:
lowber:ht=1:ha=00dd00caf000:ip=128.2.11.121:tc=default1:
oliver:ht=1:ha=00dd00fel600:ip=128.2.11.122:tc=default1:
```

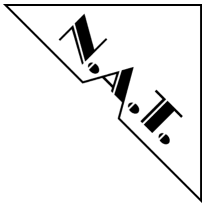
The server bootpd searches the file /DD/ETC/services for the portnumber it requires. It uses the entry bootps for its own server port and bootpc for the port number for transfers to the clients. If these entries cannot be found, it will use the default values server port 67 and client port 68.
If bootpd receives a hangup signal (SIGHUP) or a bootp query packet and discovers that configuration file was updated, it will reread the entire file. Thus, it is possible add, delete, or modify the host entries while in operation.

LIMITATIONS

A single host entry may not exceed 1024 characters.

SEE ALSO

inetd;
DARPA Internet Request For Comments RFC951, RFC1048, RFC1084



3.2 inetd

NAME

inetd - internet "super-server"

SYNOPSIS

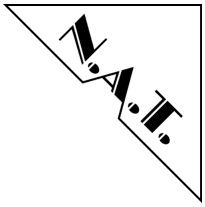
inetd [*-d*] [*configuration file*]

DESCRIPTION

inetd should be run at boot time by the startup file startup.tcp. It then listens for connections on certain internet sockets. When a connection is found on one of its sockets, it decides what service the socket corresponds to, and invokes a program to service the request. After the program is finished, it continues to listen on the socket (except in some cases which will be described below). Essentially, inetd allows running one daemon to invoke several others, reducing load on the system.

Upon execution, inetd reads its configuration information from a configuration file which, by default, is /etc/inetd.conf. There must be an entry for each field of the configuration file, with entries for each field separated by a tab or a space. Comments are denoted by a ``#" at the beginning of a line. There must be an entry for each field. The fields of the configuration file are as follows:

service name	The service name entry is the name of a valid service in the file /DD/ETC/services. For "internal" services (discussed below), the service name must be the official name of the service (that is, the first entry in /DD/ETC/services).
socket type	The socket type should be one of "stream", "dgram", or "raw", depending on whether the socket is a SOCK_STREAM, SOCK_DGRAM, or SOCK_RAW socket.
protocol	The protocol must be a valid protocol as given in the /DD/ETC/protocols. Examples might be "tcp" or "udp".



wait/nowait The wait/nowait entry is only applicable to datagram sockets (other sockets should have a "nowait" entry in this space). If a datagram server connects to its peer, freeing the socket so inetd can received further messages on the socket, it is said to be a ``multi-threaded" server, and should use the ``nowait" entry. For datagram servers which process all incoming datagrams on a socket and eventually time out, the server is said to be ``single-threaded" and should use a ``wait" entry. The ``talk" service is an example of the latter type of datagram server. Tftpd is an exception; it is a datagram server that establishes pseudo-connections. It must be listed as ``wait" in order to avoid a race; the server reads the first packet, creates a new socket, and then forks and exits to allow inetd to check for new service requests to spawn new servers.

user The user entry should contain the user name of the user as whom the server should run. This allows for servers to be given less permission than root.

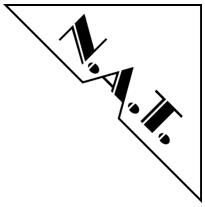
server program The server program entry should contain the pathname of the program which is to be executed by inetd when a request is found on its socket. If inetd provides this service internally, this entry should be "internal".

server program arguments The arguments to the server program should be just as they normally are, starting with argv[0], which is the name of the program. If the service is provided internally, the word ``internal" should take the place of this entry.

Inetd provides several ``trivial" services internally by use of routines within itself. These services are ``echo", ``discard", ``chargen" (character generator), ``daytime" (human readable time), and ``time" (machine readable time, in the form of the number of seconds since midnight, January 1, 1900). All of these services are tcp based.

SEE ALSO

ftpd, rexecd, rlogind, rshd, telnetd, tftpd



3.3 ftpd

NAME

ftpd - DARPA Internet File Transfer Protocol Server

SYNOPSIS

ftpd [*-ttimeout*]

DESCRIPTION

ftpd is the DARPA Internet File Transfer Protocol server process. The server uses the TCP protocol and listens at the port specified in the "ftp" service specification; see services.

The ftp server will timeout an inactive session after 15 minutes. If the -t option is specified, the inactivity timeout period will be set to timeout.

The ftp server currently supports the following ftp requests; it is not case sensitive.

Request	Description
ABOR	abort previous command
ACCT	specify account (ignored)
ALLO	allocate storage (vacuously)
APPE	append to a file
CDUP	change to parent of current working directory
CWD	change working directory
DELE	delete a file
HELP	give help information
LIST	give list files in a directory (``ls -lg")
MKD	make a directory
MODE	specify data transfer mode
NLST	give name list of files in directory (``ls")
NOOP	do nothing
PASS	specify password
PASV	prepare for server-to-server transfer
PORT	specify data connection port
PWD	print the current working directory
QUIT	terminate session
RCMD	start a process on the FTP server
RETR	retrieve a file
RMD	remove a directory
RNFR	specify rename-from file name
RNTO	specify rename-to file name
STOR	store a file
Request	**Description**

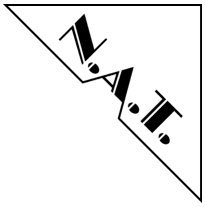
STOU	store a file with a unique name
STRU	specify data transfer structure
TYPE	specify data transfer type
USER	specify user name
XCUP	change to parent of current working directory
XCWD	change working directory
XMKD	make a directory
XPWD	print the current working directory
XRMD	remove a directory

The remaining ftp requests specified in Internet RFC 959 are recognized, but not implemented.

The ftp server will abort an active file transfer only when the ABOR command is preceded by a Telnet "Interrupt Process" (IP) signal and a Telnet "Synch" signal in the command Telnet stream, as described in Internet RFC 959.

ftpd interprets file names according to the OS-9 conventions. This allows users to utilize the metacharacters ``*?".

ftpd authenticates users.
The user name must be in the password data base, /etc/passwd, and may not be a null password. In this case a password must be provided by the client before any file operations may be performed.



3.4 telnetd

NAME

telnetd - TELNET Protocol Server

SYNOPSIS

telnetd

DESCRIPTION

The server telnetd supports the DARPA standard virtual terminal protocol TELNET. It is called by the Internet server inetd whenever a request is made for the TELNET port as it given in the file /DD/ETC/services.

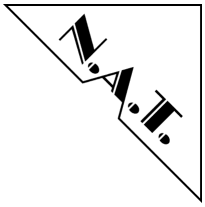
The server telnetd reserves a pseudo-terminal-device for his client, then calls a login process, and passes the slave side of the pseudo-terminal as stdin, stdout, and stderr to this login process. Telnetd retains the master side of the pseudo-terminal and exchanges characters between the login process and the remote client.

SEE ALSO

pty, telnet

Note:

A few of the TELNET commands are only partially implemented. The TELNET protocol permits the exchange of line and column numbers on the user terminal, but telnetd does not support this option.



3.5 snmpd

NAME

snmpd - simple network management protocol

SYNOPSIS

snmpd [*options*]

OPTIONS

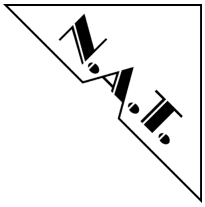
-c=<contact>	System Administrators Name
-l=<location>	Location of the System
-n=<name>	Name of the System

DESCRIPTION

snmpd is a daemon process which is based on the simple network management protocol and allows any SNMP Agent to request information about the node on which the snmpd runs.
snmpd requires N.AT. TCP/IP to be running.

Currently the following management information bases (MIB) are supported:

- MIB II:
- System Group (mib-21)
- Interfaces Group (mib-22)
- Address-Translation Group (mib-23)
- IP Group (mib-24)
- ICMP Group (mib-25)
- TCP Group (mib-26)
- UDP Group (mib-27)



3.6 tftpd

NAME

tftpd - DARPA Trivial File Transfer Protocol Server

SYNOPSIS

tftpd

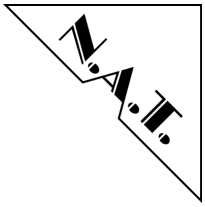
DESCRIPTION

The server process tftpd is used for the DARPA Internet Trivial File Transfer Protocol. The tftpd server uses the port assigned in the tftpd service specification (see services). The server is normally started from inetd.

Neither account nor password on the remote system are required to use tftpd. Since there is no authentication of users, tftpd only permits access to public read files. A user is only allowed to write if the file already exists and it has the public write bit set to enable writes.

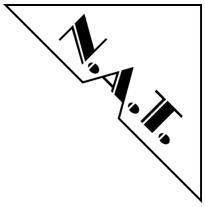
SEE ALSO

tftp, inetd



4 Appendix A

This appendix contains descriptions of the protocols of the internet protocol family, for example IP, TCP and UDP.
Note that several times functions of the socket library are referenced to. To get detailed information on these functions, please consider the "N.A.T. Socket Library Guide".



4.1 inet

NAME

inet - Internet Protocol Family

DESCRIPTION

The Internet protocol family is a collection of protocols layered atop the Internet Protocol (IP) transport layer, and utilizing the Internet address format. The Internet family provides protocol support for the SOCK_STREAM, SOCK_DGRAM, and SOCK_RAW socket types; the SOCK_RAW interface provides access to the IP protocol.

ADDRESSING

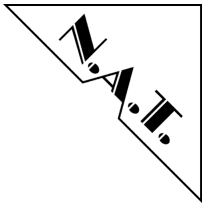
Internet addresses are four byte quantities, stored in network standard format (from left to right). The include file <in.h> defines this address as a discriminated union. For historical reasons, the structure of the address entries in the include file are in the following form:

```
struct in_addr {
        u_long s_addr;
};
```

Sockets bound to the Internet protocol family utilize the following addressing structure (also defined in the include file in.h),

```
struct sockaddr_in {
        short sin_family;
        u_short sin_port;
        struct in_addr sin_addr;
        char sin_zero[8];
};
```

Sockets may be created with the local address INADDR_ANY to effect wildcard matching on incoming messages. The address in a connect() or sendto() call may be given as INADDR_ANY to mean "this host." The distinguished address INADDR_BROADCAST is allowed as a shorthand for the broadcast address on the primary network if the first network configured supports broadcast.



PROTOCOLS

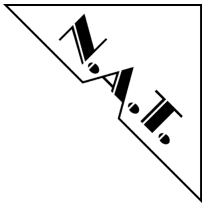
The Internet protocol family is comprised of the IP transport protocol, Internet Control Message Protocol (ICMP), Transmission Control Protocol (TCP), and User Datagram Protocol (UDP). TCP is used to support the SOCK_STREAM abstraction while UDP is used to support the SOCK_DGRAM abstraction. A raw interface to IP is available by creating an Internet socket of type SOCK_RAW. The ICMP message protocol is accessible from a raw socket.
The 32-bit Internet address contains both network and host parts. It is frequency-encoded; the most-significant bit is clear in Class A addresses, in which the high-order 8 bits are the network number. Class B addresses use the high-order 16 bits as the network field, and Class C addresses have a 24-bit network part.

00000001 00000000 00000000 00000000 (1.0.0.0) Class A
with up to 128 networks each with up to 16,777,216 host nodes
Class A is normally used for worldwide networks

10000000 00000000 00000000 00000000 (128.0.0.0) Class B
with up to 16,384 networks each with up to 65,536 host nodes

11000000 00000000 00000000 00000000 (192.0.0.0) Class C
with up to 2,097,152 networks each with up to 256 host nodes
Class C is normally used for local networks

11100000 00000000 00000000 00000000 (224.0.0.0) Class D (IP Multicasting)
This is the IP Multicasting address format. Note that the address 224.0.0.0 is invalid and 224.0.0.1 means all hosts.



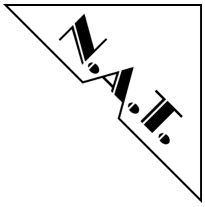
Sites with a cluster of local networks and a connection to the DARPA Internet may chose to use a single network number for the cluster; this is done by using subnet addressing. The local (host) portion of the address is further subdivided into subnet and host parts. Within a subnet, each subnet appears to be an individual network; externally, the entire cluster appears to be a single, uniform network requiring only a single routing entry. Subnet addressing is enabled and examined by the following ioctl() commands on a datagram socket in the Internet domain; they have the same form as the SIOCIFADDR command).

SIOCSIFNETMASK Set interface network mask. The network mask defines the network part of the address; if it contains more of the address than the address type would indicate, then subnets are in use.

SIOCGIFNETMASK Get interface network mask.

SEE ALSO

ioctl(), socket(), intro(), TCP, UDP, IP, ICMP



4.2 ARP

NAME

ARP - Address Resolution Protocol

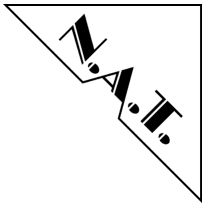
DESCRIPTION

ARP is a protocol used to dynamically map between DARPA Internet and 10Mb/s Ethernet addresses. It is used by all the 10Mb/s Ethernet interface drivers. It is not specific to Internet protocols or to 10Mb/s Ethernet, but this implementation currently supports only that combination.

ARP caches Internet-Ethernet address mappings. When an interface requests a mapping for an address not in the cache, ARP queues the message which requires the mapping and broadcasts a message on the associated network requesting the address mapping. If a response is provided, the new mapping is cached and any pending message is transmitted. ARP will queue at most one packet while waiting for a mapping request to be responded to; only the most recently "transmitted" packet is kept.
To facilitate communications with systems which do not use ARP, ioctls are provided to enter and delete entries in the Internet-to-Ethernet tables.

Usage:

```
#include        <sys/ioctl.h>
#include        <sys/socket.h>
#include        <net/if.h>
struct      arpreq arpreq;

ioctl(s, SIOCSARP, (caddr_t)&arpreq);
ioctl(s, SIOCGARP, (caddr_t)&arpreq);
ioctl(s, SIOCDARP, (caddr_t)&arpreq);
```



Each ioctl() takes the same structure as an argument. SIOCSARP sets an ARP entry, SIOCGARP gets an ARP entry, and SIOCDARP deletes an ARP entry. These ioctl() calls may be applied to any socket descriptor s, but only by the superuser. The arpreq structure contains:

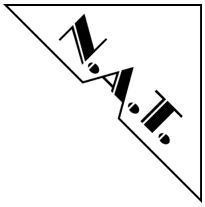
```
/*
 * ARP ioctl request
 */
struct arpreq {
        struct      sockaddr arp_pa;        /* protocol address */
        struct      sockaddr arp_ha;        /* hardware address */
        int         arp_flags;                  /* flags */
};

/* arp_flags field values */
#define ATF_COM         0x02        /* completed entry (arp_ha valid) */
#define ATF_PERM        0x04        /* permanent entry */
#define ATF_PUBL        0x08        /* publish (respond for other host) */
#define ATF_USETRAILERS     0x10        /* send trailer packets to host */
```

The address family for the arp_pa sockaddr must be AF_INET; for the arp_ha sockaddr it must be AF_UNSPEC. The only flag bits which may be written are ATF_PERM, ATF_PUBL and ATF_USETRAILERS. ATF_PERM causes the entry to be permanent if the ioctl call succeeds. The peculiar nature of the ARP tables may cause the ioctl to fail if more than 8 (permanent) Internet host addresses hash to the same slot. ATF_PUBL specifies that the ARP code should respond to ARP requests for the indicated host coming from other machines. This allows a host to act as a "ARP server," which may be useful in convincing an ARP only machine to talk to a non-ARP machine.

ARP is also used to negotiate the use of trailer IP encapsulations; trailers are an alternate encapsulation used to allow efficient packet alignment for large packets despite variable-sized headers. Hosts which wish to receive trailer encapsulations so indicate by sending gratuitous ARP translation replies along with replies to IP requests; they are also sent in reply to IP translation replies.
The negotiation is thus fully symmetrical, in that either or both hosts may request trailers. The ATF_USETRAILERS flag is used to record the receipt of such a reply, and enables the transmission of trailer packets to that host.
ARP watches passively for hosts impersonating the local host (i.e. a host which responds to an ARP mapping request for the local host's address).

SEE ALSO

inet, ifconfig



4.3 ICMP

NAME

ICMP - Internet Control Message Protocol

DESCRIPTION

ICMP is the error and control message protocol used by IP and the Internet protocol family. It may be accessed through a raw socket for network monitoring and diagnostic functions. The call to socket() has the following form:

```
#include <socket.h>
#include <in.h>

s = socket(AF_INET, SOCK_RAW, proto);
```

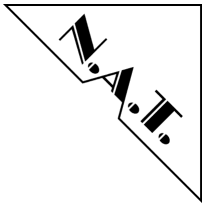
The proto parameter to the socket call to create an ICMP socket is obtained from getprotobyname(). ICMP sockets are connectionless, and are normally used with the sendto() and recvfrom() calls, though the connect()call may also be used to fix the destination for future packets (in which case the read() or recv() and write() or send() system calls may be used).
Outgoing packets automatically have an IP header prepended to them (based on the destination address). Incoming packets are received with the IP header and options intact.

A socket operation may fail with one of the following errors returned:

[EISCONN]	when trying to establish a connection on a socket which already has one, or when trying to send a datagram with the destination address specified and the socket is already connected;
[ENOTCONN]	when trying to send a datagram, but no destination address is specified, and the socket hasn't been connected;
[ENOBUFS]	when the system runs out of memory for an internal data structure;
[EADDRNOTAVAIL]	when an attempt is made to create a socket with a network address for which no network interface exists.

SEE ALSO

connect(), inet, IP, read(), recv(), recvfrom(), send(), sendto(), socket(), write()



4.4 IP

NAME

IP - Internet Protocol

DESCRIPTION

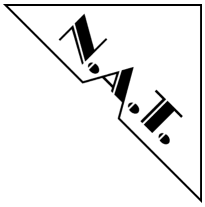
IP is the transport layer protocol used by the Internet protocol family. Options may be set at the IP level when using higher-level protocols that are based on IP (such as TCP and UDP). It may also be accessed through a raw socket when developing new protocols, or special purpose applications. The call to socket() has the following form:

```
#include <socket.h>
#include <in.h>

s = socket(AF_INET, SOCK_RAW, proto);
```

A single generic option is supported at the IP level, IP_OPTIONS, that may be used to provide IP options to be transmitted in the IP header of each outgoing packet. Options are set with setsockopt() and examined with getsockopt(). The format of IP options to be sent is that specified by the IP protocol specification, with one exception: the list of addresses for Source Route options must include the first-hop gateway at the beginning of the list of gateways. The first-hop gateway address will be extracted from the option list and the size adjusted accordingly before use. IP options may be used with any socket type in the Internet family.

Raw IP sockets are connectionless, and are normally used with the sendto() and recvfrom() calls, though the connect() call may also be used to fix the destination for future packets (in which case the read() or recv() and write() or send() system calls may be used).

If proto is 0, the default protocol IPPROTO_RAW is used for outgoing packets, and only incoming packets destined for that protocol are received. If proto is non-zero, that protocol number will be used on outgoing packets and to filter incoming packets.
Outgoing packets automatically have an IP header prepended to them (based on the destination address and the protocol number the socket is created with). Incoming packets are received with IP header and options intact.



DIAGNOSTICS

A socket operation may fail with one of the following errors returned:

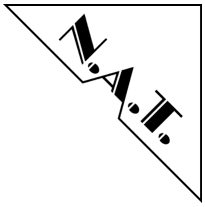
[EISCONN]	when trying to establish a connection on a socket which already has one, or when trying to send a datagram with the destination address specified and the socket is already connected;
[ENOTCONN]	when trying to send a datagram, but no destination address is specified, and the socket hasn't been connected;
[ENOBUFS]	when the system runs out of memory for an internal data structure;
[EADDRNOTAVAIL]	when an attempt is made to create a socket with a network address for which no network interface exists.

The following errors specific to IP may occur when setting or getting IP options:

[EINVAL]	An unknown socket option name was given.
[EINVAL]	The IP option field was improperly formed; an option field was shorter than the minimum value or longer than the option buffer provided.

SEE ALSO

connect(), getsockopt(), ICMP, inet, read(), recv(), recvfrom(), send(), sendto(), setsockopt(), socket(), write()



4.5 TCP

NAME

TCP - Internet Transmission Control Protocol

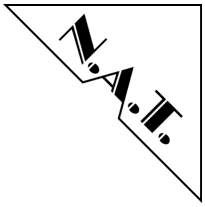
DESCRIPTION

The TCP protocol provides reliable, flow-controlled, two-way transmission of data. It is a byte-stream protocol used to support the SOCK_STREAM abstraction. TCP uses the standard Internet address format and, in addition, provides a per-host collection of port addresses. Thus, each address is composed of an Internet address specifying the host and network, with a specific TCP port on the host identifying the peer entity. The call of socket() to run TCP has the following form:

```
#include <socket.h>
#include <in.h>

s = socket(AF_INET, SOCK_STREAM, 0);
```

Sockets utilizing the TCP protocol are either active or passive. Active sockets initiate connections to passive sockets. By default TCP sockets are created active; to create a passive socket the listen() system call must be used after binding the socket with the bind() system call. Only passive sockets may use the accept() call to accept incoming connections. Only active sockets may use the connect() call to initiate connections.

Passive sockets may underspecify their location to match incoming connection requests from multiple networks. This technique, termed wildcard addressing, allows a single server to provide service to clients on multiple networks.
To create a socket which listens on all networks, the Internet address INADDR_ANY must be bound. The TCP port may still be specified at this time; if the port is not specified the system will assign one. Once a connection has been established the socket's address is fixed by the peer entity's location. The address assigned the socket is the address associated with the network interface through which packets are being transmitted and received. Normally this address corresponds to the peer entity's network.
TCP supports one socket option which is set with setsockopt() and tested with getsockopt(). Under most circumstances, TCP sends data when it is presented; when outstanding data has not yet been acknowledged, it gathers small amounts of output to be sent in a single packet once an acknowledgment is received. For a small number of clients, such as window systems that send a stream of mouse events which receive no replies, this packetization may cause significant delays. Therefore, TCP provides a boolean option, TCP_NODELAY (from <netinet/tcp.h>, to defeat this algorithm. The option level for the setsockopt call is the protocol number for TCP, available from getprotobyname().



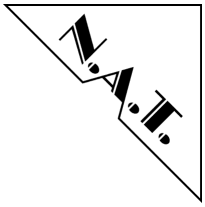
Options at the IP transport level may be used with TCP; see IP. Incoming connection requests that are source-routed are noted, and the reverse source route is used in responding.

A socket operation may fail with one of the following errors returned:

[EISCONN]	when trying to establish a connection on a socket which already has one;
[ENOBUFS]	when the system runs out of memory for an internal data structure;
[ETIMEDOUT]	when a connection was dropped due to excessive retransmissions;
[ECONNRESET]	when the remote peer forces the connection to be closed;
[ECONNREFUSED]	when the remote peer actively refuses connection establishment (usually because no process is listening to the port);
[EADDRINUSE]	when an attempt is made to create a socket with a port which has already been allocated;
[EADDRNOTAVAIL]	when an attempt is made to create a socket with a network address for which no network interface exists.

SEE ALSO

accept(), bind(), connect(), inet, IP, listen(), read(), recv(), recvfrom(), send(), sendto(), socket(), write()



4.6 UDP

NAME

UDP - Internet User Datagram Protocol

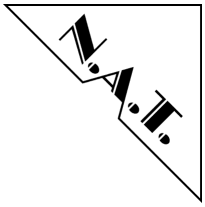
DESCRIPTION

UDP is a simple, unreliable datagram protocol which is used to support the SOCK_DGRAM abstraction for the Internet protocol family.
UDP sockets are connectionless, and are normally used with the sendto and recvfrom calls, though the connect() call may also be used to fix the destination for future packets (in which case the recv() or read() and send() or write() system calls may be used). Similar to the TCP sockets, they are defined with a Internet address and a port number. The Internet address defines a specific system in the network and the port identifies a particular application program that runs on this system. The call to socket() has the following form:

```
#include <sys/socket.h>
#include <netinet/in.h>

s = socket(AF_INET, SOCK_DGRAM, 0);
```

UDP address formats are identical to those used by TCP. In particular UDP provides a port identifier in addition to the normal Internet address format. Note that the UDP port space is separate from the TCP port space (i.e. a UDP port may not be connected to a TCP port). In addition broadcast packets may be sent (assuming the underlying network supports this) by using a reserved broadcast address; this address is network interface dependent.
Options at the IP transport level as well as those of TCP may be used with UDP; see IP.
A socket operation may fail with one of the following errors returned:

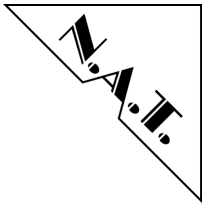
[EISCONN]	when trying to establish a connection on a socket which already has one, or when trying to send a datagram with the destination address specified and the socket is already connected;
[ENOTCONN]	when trying to send a datagram, but no destination address is specified, and the socket has not been connected;
[ENOBUFS]	when the system runs out of memory for an internal data structure;



[EADDRINUSE]	when an attempt is made to create a socket with a port which has already been allocated;
[EADDRNOTAVAIL]	when an attempt is made to create a socket with a network address for which no network interface exists.

SEE ALSO

connect(), getsockopt(), inet, IP, read(), recv(), recvfrom(), send(), sendto(), setsockopt(), socket(), write()



4.7 lo

NAME

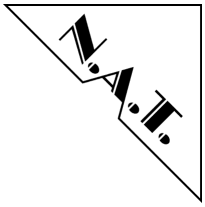
lo - Software Loopback Network Interface

DESCRIPTION

The lo interface is a software loopback mechanism that can be used for performance analysis, tests, and/or for local communications. As with all the other network interfaces, the loopback interface must be assigned a network address for every address family with which it will be used. These addresses can be set or modified using the ioctl() function´s command SIOCSIFADDR.

SEE ALSO

inet, Networking - An Introduction



4.8 nat

NAME

nat - N.A.T. Network Interface

DESCRIPTION

The nat interface can be used to initialize (e.g., assign an Internet address) hardware and software network products from N.A.T GmbH. The interface is accessed using the function ioctl().

SEE ALSO

ioctl(), inet, Networking - An Introduction