NAME

ntpq - standard NTP query program

SYNOPSIS

ntpq [-inp] [-c *command*] [*host*] [...]

DESCRIPTION

The **ntpq** utility program is used to query NTP servers which implement the recommended NTP mode 6 control message format about current state and to request changes in that state. The program may be run either in interactive mode or controlled using command line arguments. Requests to read and write arbitrary variables can be assembled, with raw and pretty-printed output options being available. **ntpq** can also obtain and print a list of peers in a common format by sending multiple queries to the server.

If one or more request options is included on the command line when **ntpq** is executed, each of the requests will be sent to the NTP servers running on each of the hosts given as command line arguments, or on localhost by default. If no request options are given, **ntpq** will attempt to read commands from the standard input and execute these on the NTP server running on the first host given on the command line, again defaulting to localhost when no other host is specified. **ntpq** will prompt for commands if the standard input is a terminal device.

ntpq uses NTP mode 6 packets to communicate with the NTP server, and hence can be used to query any compatible server on the network which permits it. Note that since NTP is a UDP protocol this communication will be somewhat unreliable, especially over large distances in terms of network topology. **ntpq** makes one attempt to retransmit requests, and will time requests out if the remote host is not heard from within a suitable timeout time.

Command line options are described following. Specifying a command line option other than **-i** or **-n** will cause the specified query (queries) to be sent to the indicated host(s) immediately. Otherwise, **ntpq** will attempt to read interactive format commands from the standard input.

-c The following argument is interpreted as an interactive format command and is added to the list of commands to be executed on the specified host(s). Multiple **-c** options may be given.

-i Force **ntpq** to operate in interactive mode. Prompts will be written to the standard output and commands read from the standard input.

-n Output all host addresses in dotted-quad numeric format rather than converting to the canonical host names.

-p Print a list of the peers known to the server as well as a summary of their state. This is equivalent to the **peers** interactive command.

INTERNAL COMMANDS

Interactive format commands consist of a keyword followed by zero to four arguments. Only enough characters of the full keyword to uniquely identify the command need be typed. The output of a command is normally sent to the standard output, but optionally the output of individual commands may be sent to a file by appending a < , followed by a file name, to the command line. A number of interactive format commands are executed entirely within the **ntpq** program itself and do not result in NTP mode 6 requests being sent to a server. These are described following.

? [*command_keyword*]

helpl [*command_keyword*] A **?** by itself will print a list of all the command keywords known to this incarnation of **ntpq** . A **?** followed by a command keyword will print function and usage information about the command. This command is probably a better source of information about **ntpq** than this manual page.

addvars *variable_name* [= *value*] [...]

rmvars *variable_name* [...]

clearvars The data carried by NTP mode 6 messages consists of a list of items of the form *variable_name* = *value* , where the **=** *value* is ignored, and can be omitted, in requests to the server to read variables. **ntpq** maintains an internal list in which data to be included in control messages can be assembled, and sent using the **readlist** and **writelist** commands described below. The **addvars** command allows variables and their optional values to be added to the list. If more than one variable is to be added, the list should be comma-separated and not contain white space. The **rmvars** command can be used to remove individual variables from the list, while the **clearlist** command removes all variables from the list.

authenticate yes | no
Normally **ntpq** does not authenticate requests unless they are write requests. The command **authenticate yes** causes **ntpq** to send authentication with all requests it makes. Authenticated requests causes some servers to handle requests slightly differently, and can occasionally melt the CPU in fuzzballs if you turn authentication on before doing a **peer** display. [I didn't know that - Ed.]

cooked Causes output from query commands to be "cooked", so that variables which are recognized by **ntpq** will have their values reformatted for human consumption. Variables which **ntpq** thinks should have a decodable value but didn't are marked with a trailing **?** .

debug more | less | off
Turns internal query program debugging on and off.

delay *milliseconds* Specify a time interval to be added to timestamps included in requests which require authentication. This is used to enable (unreliable) server reconfiguration over long delay network paths or between machines whose clocks are unsynchronized. Actually the server does not now require timestamps in authenticated requests, so this command may be obsolete.

host *hostname* Set the host to which future queries will be sent. Hostname may be either a host name or a numeric address.

hostnames [yes | no]
If **yes** is specified, host names are printed in information displays. If **no** is specified, numeric addresses are printed instead. The default is **yes** , unless modified using the command line **-n** switch.

keyid *keyid* This command allows the specification of a key number to be used to authenticate configuration requests. This must correspond to a key number the server has been configured to use for this purpose.

ntpversion 1 | 2 | 3 | 4
Sets the NTP version number which **ntpq** claims in packets. Defaults to 3, Note that mode 6 control messages (and modes, for that matter) didn't exist in NTP version 1. There appear to be no servers left which demand version 1.

quit Exit **ntpq** .

passwd This command prompts you to type in a password (which will not be echoed) which will be used to authenticate configuration requests. The password must correspond to the key configured for use by the NTP server for this purpose if such requests are to be successful.

raw Causes all output from query commands is printed as received from the remote server. The only formating/interpretation done on the data is to transform nonascii data into a printable (but barely understandable) form.

timeout *millseconds* Specify a timeout period for responses to server queries. The default is about 5000 milliseconds. Note that since **ntpq** retries each query once after a timeout, the total waiting time for a timeout will be twice the timeout value set.

CONTROL MESSAGE COMMANDS

Each peer known to an NTP server has a 16 bit integer association identifi er assigned to it. NTP control messages which carry peer variables must identify the peer the values correspond to by including its association ID. An association ID of 0 is special, and indicates the variables are system variables, whose names are drawn from a separate name space.

Control message commands result in one or more NTP mode 6 messages being sent to the server, and cause the data returned to be printed in some format. Most commands currently implemented send a single message and expect a single response. The current exceptions are the peers command, which will send a preprogrammed series of messages to obtain the data it needs, and the mreadlist and mreadvar commands, which will iterate over a range of associations.

associations
: Obtains and prints a list of association identifi ers and peer statuses for in-spec peers of the server being queried. The list is printed in columns. The fi rst of these is an index numbering the associations from 1 for internal use, the second the actual association identifi er returned by the server and the third the status word for the peer. This is followed by a number of columns containing data decoded from the status word See the peers command for a decode of the **condition** fi eld. Note that the data returned by the **associations"** command is cached internally in **ntpq** . The index is then of use when dealing with stupid servers which use association identifi ers which are hard for humans to type, in that for any subsequent commands which require an association identifi er as an argument, the form and index may be used as an alternative.

clockvar [
: *assocID*] [*variable_name* [= *value* [...]] [...]

cv [*assocID*] [*variable_name* [= *value* [...]][...] Requests that a list of the server's clock variables be sent. Servers which have a radio clock or other external synchronization will respond positively to this. If the association identifi er is omitted or zero the request is for the variables of the **system clock** and will generally get a positive response from all servers with a clock. If the server treats clocks as pseudo-peers, and hence can possibly have more than one clock connected at once, referencing the appropriate peer association ID will show the variables of a particular clock. Omitting the variable list will cause the server to return a default variable display.

lassocations
: Obtains and prints a list of association identifi ers and peer statuses for all associations for which the server is maintaining state. This command differs from the **associations** command only for servers which retain state for out-of-spec client associations (i.e., fuzzballs). Such associations are normally omitted from the display when the **associations** command is used, but are included in the output of **lassociations** .

lpassociations
: Print data for all associations, including out-of-spec client associations, from the internally cached list of associations. This command differs from **passociations** only when dealing with fuzzballs.

lpeers Like R peers, except a summary of all associations for which the server is maintaining state is printed. This can produce a much longer list of peers from fuzzball servers.

mreadlist
: *assocID assocID*

mrl *assocID assocID* Like the **readlist** command, except the query is done for each of a range of (nonzero) association IDs. This range is determined from the association list cached by the most recent **associations** command.

mreadvar

assocID assocID [*variable_name* [= *value* [...]

mrv *assocID assocID* [*variable_name* [= *value* [...] Like the **readvar** command, except the query is done for each of a range of (nonzero) association IDs. This range is determined from the association list cached by the most recent **associations** command.

opeers An old form of the **peers** command with the reference ID replaced by the local interface address.

passociations

Displays association data concerning in-spec peers from the internally cached list of associations. This command performs identically to the **associations** except that it displays the internally stored data rather than making a new query.

peers Obtains a current list peers of the server, along with a summary of each peer's state. Summary information includes the address of the remote peer, the reference ID (0.0.0.0 if this is unknown), the stratum of the remote peer, the type of the peer (local, unicast, multicast or broadcast), when the last packet was received, the polling interval, in seconds, the reachability register, in octal, and the current estimated delay, offset and dispersion of the peer, all in milliseconds. The character in the left margin indicates the fate of this peer in the clock selection process. Following is a list of these characters, the pigeon used in the **rv** command, and a short explanation of the condition revealed.

space reject

The peer is discarded as unreachable, synchronized to this server (synch loop) or outrageous synchronization distance.

x falsetick

The peer is discarded by the intersection algorithm as a falseticker.

The peer is discarded as not among the fi rst ten peers sorted

by synchronization distance and so is probably a poor candidate for further consideration.

- outlyer

The peer is discarded by the clustering algorithm as an outlyer.

+ candidat

The peer is a survivor and a candidate for the combining algorithm.

# selected

The peer is a survivor, but not among the fi rst six peers sorted by synchronization distance. If the assocation is ephemeral, it may be demobilized to conserve resources.

*** sys.peer**

The peer has been declared the system peer and lends its variables to the system variables.

o pps.peer

The peer has been declared the system peer and lends its variables to thesystem variables. However, the actual system synchronization is derived from a pulse-per-second (PPS) signal, either indirectly via the PPS reference clock driver or directly via kernel interface.

The **flash** variable is a valuable debugging aid. It displays the results of the original sanity checks defi ned in the NTP specifi cation RFC-1305 and additional ones added in NTP Version 4. There are eleven tests called **TEST1** through **TEST11** . The tests are performed in a certain order designed to gain maximum diagnostic information while protecting against accidental or malicious errors. The **flash** variable is fi rst initialized to zero. If after each set of tests one or more bits are set, the packet is discarded. Tests **TEST4** and **TEST5** check the access permissions and cryptographic message digest. If any bits are set after that, the packet is discarded. Tests **TEST10** and **TEST11** check the authentication state using Autokey public-key cryptography, as described in the Authentication Options page. If any bits are set and the association has previously been marked reachable, the packet is discarded; otherwise, the originate and receive timestamps are saved, as required by the NTP protocol, and processing continues.

Tests **TEST1** through **TEST3** check the packet timestamps from which the offset and delay are calculated. If any bits are set, the packet is discarded; otherwise, the packet header variables are saved. Tests **TEST6** through **TEST8** check the health of the server. If any bits are set, the packet is discarded; otherwise, the offset and delay relative to the server are calculated and saved. Test **TEST9** checks the health of the association itself. If any bits are set, the packet is discarded; otherwise, the saved variables are passed to the clock filter and mitigation algorithms.

The **flash** bits for each test read in increasing order from the least significant bit are defined as follows.

TEST1 Duplicate packet. The packet is at best a casual retransmission and at worst a malicious replay.

TEST2 Bogus packet. The packet is not a reply to a message previously sent. This can happen when the NTP daemon is restarted and before somebody else notices.

TEST3 Unsynchronized. One or more timestamp fields are invalid. This normally happens when the first packet from a peer is received.

TEST4 Access is denied. See the Access Control Options page.

TEST5 Cryptographic authentication fails. See the Authentication Options page.

TEST6 The server is unsynchronized. Wind up its clock first.

TEST7 The server stratum is at the maximum than 15. It is probably unsynchronized and its clock needs to be wound up.

TEST8 Either the root delay or dispersion is greater than one second, which is highly unlikely unless the peer is synchronized to Mars.

TEST9 Either the peer delay or dispersion is greater than one second, which is higly unlikely unless the peer is on Mars.

TEST10
The autokey protocol has detected an authentication failure. See the Authentication Options page.

TEST11
The autokey protocol has not verified the server or peer is authentic and has valid public key credentials. See the Authentication Options page.

support include the following:

certificate
filestamp Shows the NTP seconds when the certificate file was created.

hostname
host Shows the name of the host as returned by the Unix **gethostname()** library function.

flags *hex* Shows the current flag bits, where the *hex* bits are interpreted as follows:

0x01 autokey enabled

0x02 RSA public/private key files present

0x04 PKI certificate file present

0x08 Diffie-Hellman parameters file present

0x10 NIST leapseconds table file present

leapseconds
filestamp Shows the NTP seconds when the NIST leapseconds table fi le was created.

params *filestamp* Shows the NTP seconds when the Diffi e-Hellman agreement parameter fi le was created.

publickey
filestamp Shows the NTP seconds when the RSA public/private key fi les were created.

refresh *timestamp* Shows the NTP seconds when the public cryptographic values were refreshed and signed.

tai *offset* Shows the TAI-UTC offset in seconds obtained from the NIST leapseconds table.

support include the following:

certificate
filestamp Shows the NTP seconds when the certifi cate fi le was created.

flags *hex* Shows the current flag bits, where the *hex* bits are interpreted as in the system variable of the same name. The bits are set in the fi rst autokey message received from the server and then reset as the associated data are obtained from the server and stored.

hcookie *hex* Shows the host cookie used in the key agreement algorithm.

initkey *key* Shows the initial key used by the key list generator in the autokey protocol.

initsequence
index Shows the initial index used by the key list generator in the autokey protocol.

pcookie *hex* Specifi es the peer cookie used in the key agreement algorithm.

timestamp
time Shows the NTP seconds when the last autokey key list was generated and signed.

pstatus *assocID* Sends a read status request to the server for the given association. The names and values of the peer variables returned will be printed. Note that the status word from the header is displayed preceding the variables, both in hexidecimal and in pidgeon English.

readlist [
assocID]

rl [*assocID*] Requests that the values of the variables in the internal variable list be returned by the server. If the association ID is omitted or is 0 the variables are assumed to be system variables. Otherwise they are treated as peer variables. If the internal variable list is empty a request is sent without data, which should induce the remote server to return a default display.

readvar *assocID variable_name* [= *value*] [...]

rv *assocID* [*variable_name* [= *value*] [Requests that the values of the specifi ed variables be returned by the server by sending a read variables request. If the association ID is omitted or is given as zero the variables are system variables, otherwise they are peer variables and the values returned will be those of the corresponding peer. Omitting the variable list will send a request with no data which should induce the server to return a default display.

writevar
assocID variable_name [= *value* [...] Like the readvar request, except the specifi ed variables are written instead of read.

writelist [
assocID] Like the readlist request, except the internal list variables are written instead of read.

BUGS

The peers command is non-atomic and may occasionally result in spurious error messages about invalid associations occurring and terminating the command. The timeout time is a fixed constant, which means you wait a long time for timeouts since it assumes sort of a worst case. The program should improve the timeout estimate as it sends queries to a particular host, but doesn't.

AUTHOR

David L. Mills <mills@udel.edu>