

NAME

ntp.conf — Network Time Protocol (NTP) daemon configuration file format

SYNOPSIS

ntp.conf [**--option-name**] [**--option-name** *value*]

All arguments must be options.

DESCRIPTION

The **ntp.conf** configuration file is read at initial startup by the `ntpd(8)` daemon in order to specify the synchronization sources, modes and other related information. Usually, it is installed in the `/etc` directory, but could be installed elsewhere (see the daemon's `-c` command line option).

The file format is similar to other UNIX configuration files. Comments begin with a '#' character and extend to the end of the line; blank lines are ignored. Configuration commands consist of an initial keyword followed by a list of arguments, some of which may be optional, separated by whitespace. Commands may not be continued over multiple lines. Arguments may be host names, host addresses written in numeric, dotted-quad form, integers, floating point numbers (when specifying times in seconds) and text strings.

The rest of this page describes the configuration and control options. The "Notes on Configuring NTP and Setting up an NTP Subnet" page (available as part of the HTML documentation provided in `/usr/share/doc/ntp`) contains an extended discussion of these options. In addition to the discussion of general **Configuration Options**, there are sections describing the following supported functionality and the options used to control it:

- **Authentication Support**
- **Monitoring Support**
- **Access Control Support**
- **Automatic NTP Configuration Options**
- **Reference Clock Support**
- **Miscellaneous Options**

Following these is a section describing **Miscellaneous Options**. While there is a rich set of options available, the only required option is one or more **pool**, **server**, **peer**, **broadcast** or **manycastclient** commands.

Configuration Support

Following is a description of the configuration commands in NTPv4. These commands have the same basic functions as in NTPv3 and in some cases new functions and new arguments. There are two classes of commands, configuration commands that configure a persistent association with a remote server or peer or reference clock, and auxiliary commands that specify environmental variables that control various related operations.

Configuration Commands

The various modes are determined by the command keyword and the type of the required IP address. Addresses are classed by type as (s) a remote server or peer (IPv4 class A, B and C), (b) the broadcast address of a local interface, (m) a multicast address (IPv4 class D), or (r) a reference clock address (127.127.x.x). Note that only those options applicable to each command are listed below. Use of options not listed may not be caught as an error, but may result in some weird and even destructive behavior.

If the Basic Socket Interface Extensions for IPv6 (RFC-2553) is detected, support for the IPv6 address family is generated in addition to the default support of the IPv4 address family. In a few cases, including the **reslist** billboard generated by `ntpq(1)` or `ntpd(1)`, IPv6 addresses are automatically generated. IPv6

addresses can be identified by the presence of colons “:” in the address field. IPv6 addresses can be used almost everywhere where IPv4 addresses can be used, with the exception of reference clock addresses, which are always IPv4.

Note that in contexts where a host name is expected, a **-4** qualifier preceding the host name forces DNS resolution to the IPv4 namespace, while a **-6** qualifier forces DNS resolution to the IPv6 namespace. See IPv6 references for the equivalent classes for that address family.

```
pool address [burst] [iburst] [version version] [prefer] [minpoll minpoll]
    [maxpoll maxpoll]
```

```
server address [key key | autokey] [burst] [iburst] [version version] [prefer]
    [minpoll minpoll] [maxpoll maxpoll] [true]
```

```
peer address [key key | autokey] [version version] [prefer] [minpoll minpoll]
    [maxpoll maxpoll] [true] [xleave]
```

```
broadcast address [key key | autokey] [version version] [prefer] [minpoll
    minpoll] [ttl ttl] [xleave]
```

```
manycastclient address [key key | autokey] [version version] [prefer]
    [minpoll minpoll] [maxpoll maxpoll] [ttl ttl]
```

These five commands specify the time server name or address to be used and the mode in which to operate. The *address* can be either a DNS name or an IP address in dotted-quad notation. Additional information on association behavior can be found in the "Association Management" page (available as part of the HTML documentation provided in `/usr/share/doc/ntp`).

pool For type *s* addresses, this command mobilizes a persistent client mode association with a number of remote servers. In this mode the local clock can be synchronized to the remote server, but the remote server can never be synchronized to the local clock.

server For type *s* and *r* addresses, this command mobilizes a persistent client mode association with the specified remote server or local radio clock. In this mode the local clock can be synchronized to the remote server, but the remote server can never be synchronized to the local clock. This command should *not* be used for type *b* or *m* addresses.

peer For type *s* addresses (only), this command mobilizes a persistent symmetric-active mode association with the specified remote peer. In this mode the local clock can be synchronized to the remote peer or the remote peer can be synchronized to the local clock. This is useful in a network of servers where, depending on various failure scenarios, either the local or remote peer may be the better source of time. This command should NOT be used for type *b*, *m* or *r* addresses.

broadcast

For type *b* and *m* addresses (only), this command mobilizes a persistent broadcast mode association. Multiple commands can be used to specify multiple local broadcast interfaces (subnets) and/or multiple multicast groups. Note that local broadcast messages go only to the interface associated with the subnet specified, but multicast messages go to all interfaces. In broadcast mode the local server sends periodic broadcast messages to a client population at the *address* specified, which is usually the broadcast address on (one of) the local network(s) or a multicast address assigned to NTP. The IANA has assigned the multicast group address IPv4 224.0.1.1 and IPv6 ff05::101 (site local) exclusively to NTP, but other nonconflicting addresses can be used to contain the messages within administrative boundaries. Ordinarily, this specification applies only to the local server operating as a sender; for operation as a broadcast client, see the **broadcastclient** or **multicastclient** commands below.

manycastclient

For type m addresses (only), this command mobilizes a manycast client mode association for the multicast address specified. In this case a specific address must be supplied which matches the address used on the **manycastserver** command for the designated manycast servers. The NTP multicast address 224.0.1.1 assigned by the IANA should NOT be used, unless specific means are taken to avoid spraying large areas of the Internet with these messages and causing a possibly massive implosion of replies at the sender. The **manycastserver** command specifies that the local server is to operate in client mode with the remote servers that are discovered as the result of broadcast/multicast messages. The client broadcasts a request message to the group address associated with the specified *address* and specifically enabled servers respond to these messages. The client selects the servers providing the best time and continues as with the **server** command. The remaining servers are discarded as if never heard.

Options:

autokey

All packets sent to and received from the server or peer are to include authentication fields encrypted using the autokey scheme described in **Authentication Options**.

burst when the server is reachable, send a burst of eight packets instead of the usual one. The packet spacing is normally 2 s; however, the spacing between the first and second packets can be changed with the **calldelay** command to allow additional time for a modem or ISDN call to complete. This is designed to improve timekeeping quality with the **server** command and s addresses.

iburst When the server is unreachable, send a burst of eight packets instead of the usual one. The packet spacing is normally 2 s; however, the spacing between the first two packets can be changed with the **calldelay** command to allow additional time for a modem or ISDN call to complete. This is designed to speed the initial synchronization acquisition with the **server** command and s addresses and when ntpd(8) is started with the **-q** option.

key *key*

All packets sent to and received from the server or peer are to include authentication fields encrypted using the specified *key* identifier with values from 1 to 65535, inclusive. The default is to include no encryption field.

minpoll *minpoll***maxpoll** *maxpoll*

These options specify the minimum and maximum poll intervals for NTP messages, as a power of 2 in seconds. The maximum poll interval defaults to 10 (1,024 s), but can be increased by the **maxpoll** option to an upper limit of 17 (36.4 h). The minimum poll interval defaults to 6 (64 s), but can be decreased by the **minpoll** option to a lower limit of 4 (16 s).

noselect

Marks the server as unused, except for display purposes. The server is discarded by the selection algorithm.

preempt

Says the association can be preempted.

true Marks the server as a truechimer. Use this option only for testing.

prefer Marks the server as preferred. All other things being equal, this host will be chosen for synchronization among a set of correctly operating hosts. See the "Mitigation Rules and the prefer Keyword" page (available as part of the HTML documentation provided in /usr/share/doc/ntp) for further information.

true Forces the association to always survive the selection and clustering algorithms. This option should almost certainly *only* be used while testing an association.

ttl *ttl*

This option is used only with broadcast server and manycast client modes. It specifies the time-to-live *ttl* to use on broadcast server and multicast server and the maximum *ttl* for the expanding ring search with manycast client packets. Selection of the proper value, which defaults to 127, is something of a black art and should be coordinated with the network administrator.

version *version*

Specifies the version number to be used for outgoing NTP packets. Versions 1–4 are the choices, with version 4 the default.

xleave Valid in **peer** and **broadcast** modes only, this flag enables interleave mode.

Auxiliary Commands

broadcastclient

This command enables reception of broadcast server messages to any local interface (type b) address. Upon receiving a message for the first time, the broadcast client measures the nominal server propagation delay using a brief client/server exchange with the server, then enters the broadcast client mode, in which it synchronizes to succeeding broadcast messages. Note that, in order to avoid accidental or malicious disruption in this mode, both the server and client should operate using symmetric-key or public-key authentication as described in **Authentication Options**.

manycastserver *address . . .*

This command enables reception of manycast client messages to the multicast group address(es) (type m) specified. At least one address is required, but the NTP multicast address 224.0.1.1 assigned by the IANA should NOT be used, unless specific means are taken to limit the span of the reply and avoid a possibly massive implosion at the original sender. Note that, in order to avoid accidental or malicious disruption in this mode, both the server and client should operate using symmetric-key or public-key authentication as described in **Authentication Options**.

multicastclient *address . . .*

This command enables reception of multicast server messages to the multicast group address(es) (type m) specified. Upon receiving a message for the first time, the multicast client measures the nominal server propagation delay using a brief client/server exchange with the server, then enters the broadcast client mode, in which it synchronizes to succeeding multicast messages. Note that, in order to avoid accidental or malicious disruption in this mode, both the server and client should operate using symmetric-key or public-key authentication as described in **Authentication Options**.

mdntries *number*

If we are participating in mDNS, after we have synched for the first time we attempt to register with the mDNS system. If that registration attempt fails, we try again at one minute intervals for up to **mdntries** times. After all, **ntpd** may be starting before mDNS. The default value for **mdntries** is 5.

Authentication Support

Authentication support allows the NTP client to verify that the server is in fact known and trusted and not an intruder intending accidentally or on purpose to masquerade as that server. The NTPv3 specification RFC-1305 defines a scheme which provides cryptographic authentication of received NTP packets. Originally, this was done using the Data Encryption Standard (DES) algorithm operating in Cipher Block Chaining (CBC) mode, commonly called DES-CBC. Subsequently, this was replaced by the RSA Message Digest 5 (MD5) algorithm using a private key, commonly called keyed-MD5. Either algorithm computes a message digest, or one-way hash, which can be used to verify the server has the correct private key and key

identifier.

NTPv4 retains the NTPv3 scheme, properly described as symmetric key cryptography and, in addition, provides a new Autokey scheme based on public key cryptography. Public key cryptography is generally considered more secure than symmetric key cryptography, since the security is based on a private value which is generated by each server and never revealed. With Autokey all key distribution and management functions involve only public values, which considerably simplifies key distribution and storage. Public key management is based on X.509 certificates, which can be provided by commercial services or produced by utility programs in the OpenSSL software library or the NTPv4 distribution.

While the algorithms for symmetric key cryptography are included in the NTPv4 distribution, public key cryptography requires the OpenSSL software library to be installed before building the NTP distribution. Directions for doing that are on the Building and Installing the Distribution page.

Authentication is configured separately for each association using the **key** or **autokey** subcommand on the **peer**, **server**, **broadcast** and **manycastclient** configuration commands as described in **Configuration Options** page. The authentication options described below specify the locations of the key files, if other than default, which symmetric keys are trusted and the interval between various operations, if other than default.

Authentication is always enabled, although ineffective if not configured as described below. If a NTP packet arrives including a message authentication code (MAC), it is accepted only if it passes all cryptographic checks. The checks require correct key ID, key value and message digest. If the packet has been modified in any way or replayed by an intruder, it will fail one or more of these checks and be discarded. Furthermore, the Autokey scheme requires a preliminary protocol exchange to obtain the server certificate, verify its credentials and initialize the protocol

The **auth** flag controls whether new associations or remote configuration commands require cryptographic authentication. This flag can be set or reset by the **enable** and **disable** commands and also by remote configuration commands sent by a `ntpd(1)` program running on another machine. If this flag is enabled, which is the default case, new broadcast client and symmetric passive associations and remote configuration commands must be cryptographically authenticated using either symmetric key or public key cryptography. If this flag is disabled, these operations are effective even if not cryptographic authenticated. It should be understood that operating with the **auth** flag disabled invites a significant vulnerability where a rogue hacker can masquerade as a falseticker and seriously disrupt system timekeeping. It is important to note that this flag has no purpose other than to allow or disallow a new association in response to new broadcast and symmetric active messages and remote configuration commands and, in particular, the flag has no effect on the authentication process itself.

An attractive alternative where multicast support is available is manycast mode, in which clients periodically troll for servers as described in the **Automatic NTP Configuration Options** page. Either symmetric key or public key cryptographic authentication can be used in this mode. The principle advantage of manycast mode is that potential servers need not be configured in advance, since the client finds them during regular operation, and the configuration files for all clients can be identical.

The security model and protocol schemes for both symmetric key and public key cryptography are summarized below; further details are in the briefings, papers and reports at the NTP project page linked from <http://www.ntp.org/>.

Symmetric-Key Cryptography

The original RFC-1305 specification allows any one of possibly 65,535 keys, each distinguished by a 32-bit key identifier, to authenticate an association. The servers and clients involved must agree on the key and key identifier to authenticate NTP packets. Keys and related information are specified in a key file, usually called `ntp.keys`, which must be distributed and stored using secure means beyond the scope of the NTP protocol itself. Besides the keys used for ordinary NTP associations, additional keys can be used as passwords for the

`ntpq(1)` and `ntpd(1)` utility programs.

When `ntpd(8)` is first started, it reads the key file specified in the **keys** configuration command and installs the keys in the key cache. However, individual keys must be activated with the **trusted** command before use. This allows, for instance, the installation of possibly several batches of keys and then activating or deactivating each batch remotely using `ntpd(1)`. This also provides a revocation capability that can be used if a key becomes compromised. The **requestkey** command selects the key used as the password for the `ntpd(1)` utility, while the **controlkey** command selects the key used as the password for the `ntpq(1)` utility.

Public Key Cryptography

NTPv4 supports the original NTPv3 symmetric key scheme described in RFC-1305 and in addition the Autokey protocol, which is based on public key cryptography. The Autokey Version 2 protocol described on the Autokey Protocol page verifies packet integrity using MD5 message digests and verifies the source with digital signatures and any of several digest/signature schemes. Optional identity schemes described on the Identity Schemes page and based on cryptographic challenge/response algorithms are also available. Using all of these schemes provides strong security against replay with or without modification, spoofing, masquerade and most forms of clogging attacks.

The Autokey protocol has several modes of operation corresponding to the various NTP modes supported. Most modes use a special cookie which can be computed independently by the client and server, but encrypted in transmission. All modes use in addition a variant of the S-KEY scheme, in which a pseudo-random key list is generated and used in reverse order. These schemes are described along with an executive summary, current status, briefing slides and reading list on the **Autonomous Authentication** page.

The specific cryptographic environment used by Autokey servers and clients is determined by a set of files and soft links generated by the `ntp-keygen(1)` program. This includes a required host key file, required certificate file and optional sign key file, leapsecond file and identity scheme files. The digest/signature scheme is specified in the X.509 certificate along with the matching sign key. There are several schemes available in the OpenSSL software library, each identified by a specific string such as **md5WithRSAEncryption**, which stands for the MD5 message digest with RSA encryption scheme. The current NTP distribution supports all the schemes in the OpenSSL library, including those based on RSA and DSA digital signatures.

NTP secure groups can be used to define cryptographic compartments and security hierarchies. It is important that every host in the group be able to construct a certificate trail to one or more trusted hosts in the same group. Each group host runs the Autokey protocol to obtain the certificates for all hosts along the trail to one or more trusted hosts. This requires the configuration file in all hosts to be engineered so that, even under anticipated failure conditions, the NTP subnet will form such that every group host can find a trail to at least one trusted host.

Naming and Addressing

It is important to note that Autokey does not use DNS to resolve addresses, since DNS can't be completely trusted until the name servers have synchronized clocks. The cryptographic name used by Autokey to bind the host identity credentials and cryptographic values must be independent of interface, network and any other naming convention. The name appears in the host certificate in either or both the subject and issuer fields, so protection against DNS compromise is essential.

By convention, the name of an Autokey host is the name returned by the Unix `gethostname(2)` system call or equivalent in other systems. By the system design model, there are no provisions to allow alternate names or aliases. However, this is not to say that DNS aliases, different names for each interface, etc., are constrained in any way.

It is also important to note that Autokey verifies authenticity using the host name, network address and public keys, all of which are bound together by the protocol specifically to deflect masquerade attacks. For this reason Autokey includes the source and destination IP addresses in message digest computations and so the same addresses must be available at both the server and client. For this reason operation with network address translation schemes is not possible. This reflects the intended robust security model where government and corporate NTP servers are operated outside firewall perimeters.

Operation

A specific combination of authentication scheme (none, symmetric key, public key) and identity scheme is called a cryptotype, although not all combinations are compatible. There may be management configurations where the clients, servers and peers may not all support the same cryptotypes. A secure NTPv4 subnet can be configured in many ways while keeping in mind the principles explained above and in this section. Note however that some cryptotype combinations may successfully interoperate with each other, but may not represent good security practice.

The cryptotype of an association is determined at the time of mobilization, either at configuration time or some time later when a message of appropriate cryptotype arrives. When mobilized by a **server** or **peer** configuration command and no **key** or **autokey** subcommands are present, the association is not authenticated; if the **key** subcommand is present, the association is authenticated using the symmetric key ID specified; if the **autokey** subcommand is present, the association is authenticated using Autokey.

When multiple identity schemes are supported in the Autokey protocol, the first message exchange determines which one is used. The client request message contains bits corresponding to which schemes it has available. The server response message contains bits corresponding to which schemes it has available. Both server and client match the received bits with their own and select a common scheme.

Following the principle that time is a public value, a server responds to any client packet that matches its cryptotype capabilities. Thus, a server receiving an unauthenticated packet will respond with an unauthenticated packet, while the same server receiving a packet of a cryptotype it supports will respond with packets of that cryptotype. However, unconfigured broadcast or manycast client associations or symmetric passive associations will not be mobilized unless the server supports a cryptotype compatible with the first packet received. By default, unauthenticated associations will not be mobilized unless overridden in a decidedly dangerous way.

Some examples may help to reduce confusion. Client Alice has no specific cryptotype selected. Server Bob has both a symmetric key file and minimal Autokey files. Alice's unauthenticated messages arrive at Bob, who replies with unauthenticated messages. Cathy has a copy of Bob's symmetric key file and has selected key ID 4 in messages to Bob. Bob verifies the message with his key ID 4. If it's the same key and the message is verified, Bob sends Cathy a reply authenticated with that key. If verification fails, Bob sends Cathy a thing called a crypto-NAK, which tells her something broke. She can see the evidence using the `ntpqq(1)` program.

Denise has rolled her own host key and certificate. She also uses one of the identity schemes as Bob. She sends the first Autokey message to Bob and they both dance the protocol authentication and identity steps. If all comes out okay, Denise and Bob continue as described above.

It should be clear from the above that Bob can support all the girls at the same time, as long as he has compatible authentication and identity credentials. Now, Bob can act just like the girls in his own choice of servers; he can run multiple configured associations with multiple different servers (or the same server, although that might not be useful). But, wise security policy might preclude some cryptotype combinations; for instance, running an identity scheme with one server and no authentication with another might not be wise.

Key Management

The cryptographic values used by the Autokey protocol are incorporated as a set of files generated by the `ntp-keygen(1)` utility program, including symmetric key, host key and public certificate files, as well as sign key, identity parameters and leapseconds files. Alternatively, host and sign keys and certificate files can be generated by the OpenSSL utilities and certificates can be imported from public certificate authorities. Note that symmetric keys are necessary for the `ntpqc(1)` and `ntpd(1)` utility programs. The remaining files are necessary only for the Autokey protocol.

Certificates imported from OpenSSL or public certificate authorities have certain limitations. The certificate should be in ASN.1 syntax, X.509 Version 3 format and encoded in PEM, which is the same format used by OpenSSL. The overall length of the certificate encoded in ASN.1 must not exceed 1024 bytes. The subject distinguished name field (CN) is the fully qualified name of the host on which it is used; the remaining subject fields are ignored. The certificate extension fields must not contain either a subject key identifier or a issuer key identifier field; however, an extended key usage field for a trusted host must contain the value **trustRoot**; Other extension fields are ignored.

Authentication Commands

autokey [*logsec*]

Specifies the interval between regenerations of the session key list used with the Autokey protocol. Note that the size of the key list for each association depends on this interval and the current poll interval. The default value is 12 (4096 s or about 1.1 hours). For poll intervals above the specified interval, a session key list with a single entry will be regenerated for every message sent.

controlkey *key*

Specifies the key identifier to use with the `ntpqc(1)` utility, which uses the standard protocol defined in RFC-1305. The *key* argument is the key identifier for a trusted key, where the value can be in the range 1 to 65,535, inclusive.

crypto [**cert** *file*] [**leap** *file*] [**randfile** *file*] [**host** *file*] [**sign** *file*] [**gq** *file*] [**gqpar** *file*] [**iffpar** *file*] [**mvpar** *file*] [**pw** *password*]

This command requires the OpenSSL library. It activates public key cryptography, selects the message digest and signature encryption scheme and loads the required private and public values described above. If one or more files are left unspecified, the default names are used as described above. Unless the complete path and name of the file are specified, the location of a file is relative to the keys directory specified in the **keysdir** command or default `/usr/local/etc`. Following are the subcommands:

cert *file*

Specifies the location of the required host public certificate file. This overrides the link `ntpkey_cert_hostname` in the keys directory.

gqpar *file*

Specifies the location of the optional GQ parameters file. This overrides the link `ntpkey_gq_hostname` in the keys directory.

host *file*

Specifies the location of the required host key file. This overrides the link `ntpkey_key_hostname` in the keys directory.

iffpar *file*

Specifies the location of the optional IFF parameters file. This overrides the link `ntpkey_iff_hostname` in the keys directory.

leap *file*

Specifies the location of the optional leapsecond file. This overrides the link `ntpkey_leap` in the keys directory.

mvpar *file*

Specifies the location of the optional MV parameters file. This overrides the link `ntpkey_mv_hostname` in the keys directory.

pw *password*

Specifies the password to decrypt files containing private keys and identity parameters. This is required only if these files have been encrypted.

randfile *file*

Specifies the location of the random seed file used by the OpenSSL library. The defaults are described in the main text above.

sign *file*

Specifies the location of the optional sign key file. This overrides the link `ntpkey_sign_hostname` in the keys directory. If this file is not found, the host key is also the sign key.

keys *keyfile*

Specifies the complete path and location of the MD5 key file containing the keys and key identifiers used by `ntpd(8)`, `ntpq(1)` and `ntpd(1)` when operating with symmetric key cryptography. This is the same operation as the `-k` command line option.

keysdir *path*

This command specifies the default directory path for cryptographic keys, parameters and certificates. The default is `/usr/local/etc/`.

requestkey *key*

Specifies the key identifier to use with the `ntpd(1)` utility program, which uses a proprietary protocol specific to this implementation of `ntpd(8)`. The *key* argument is a key identifier for the trusted key, where the value can be in the range 1 to 65,535, inclusive.

revoke *logsec*

Specifies the interval between re-randomization of certain cryptographic values used by the Autokey scheme, as a power of 2 in seconds. These values need to be updated frequently in order to deflect brute-force attacks on the algorithms of the scheme; however, updating some values is a relatively expensive operation. The default interval is 16 (65,536 s or about 18 hours). For poll intervals above the specified interval, the values will be updated for every message sent.

trustedkey *key . . .*

Specifies the key identifiers which are trusted for the purposes of authenticating peers with symmetric key cryptography, as well as keys used by the `ntpq(1)` and `ntpd(1)` programs. The authentication procedures require that both the local and remote servers share the same key and key identifier for this purpose, although different keys can be used with different servers. The *key* arguments are 32-bit unsigned integers with values from 1 to 65,535.

Error Codes

The following error codes are reported via the NTP control and monitoring protocol trap mechanism.

- | | |
|-----|--|
| 101 | (bad field format or length) The packet has invalid version, length or format. |
| 102 | (bad timestamp) The packet timestamp is the same or older than the most recent received. This could be due to a replay or a server clock time step. |
| 103 | (bad filestamp) The packet filestamp is the same or older than the most recent received. This could be due to a replay or a key file generation error. |

- 104 (bad or missing public key) The public key is missing, has incorrect format or is an unsupported type.
- 105 (unsupported digest type) The server requires an unsupported digest/signature scheme.
- 106 (mismatched digest types) Not used.
- 107 (bad signature length) The signature length does not match the current public key.
- 108 (signature not verified) The message fails the signature check. It could be bogus or signed by a different private key.
- 109 (certificate not verified) The certificate is invalid or signed with the wrong key.
- 110 (certificate not verified) The certificate is not yet valid or has expired or the signature could not be verified.
- 111 (bad or missing cookie) The cookie is missing, corrupted or bogus.
- 112 (bad or missing leapseconds table) The leapseconds table is missing, corrupted or bogus.
- 113 (bad or missing certificate) The certificate is missing, corrupted or bogus.
- 114 (bad or missing identity) The identity key is missing, corrupt or bogus.

Monitoring Support

`ntpd(8)` includes a comprehensive monitoring facility suitable for continuous, long term recording of server and client timekeeping performance. See the **statistics** command below for a listing and example of each type of statistics currently supported. Statistic files are managed using file generation sets and scripts in the `./scripts` directory of the source code distribution. Using these facilities and UNIX `cron(8)` jobs, the data can be automatically summarized and archived for retrospective analysis.

Monitoring Commands

statistics *name* . . .

Enables writing of statistics records. Currently, eight kinds of *name* statistics are supported.

clockstats

Enables recording of clock driver statistics information. Each update received from a clock driver appends a line of the following form to the file generation set named **clockstats**:

```
49213 525.624 127.127.4.1 93 226 00:08:29.606 D
```

The first two fields show the date (Modified Julian Day) and time (seconds and fraction past UTC midnight). The next field shows the clock address in dotted-quad notation. The final field shows the last timecode received from the clock in decoded ASCII format, where meaningful. In some clock drivers a good deal of additional information can be gathered and displayed as well. See information specific to each clock for further details.

cryptostats

This option requires the OpenSSL cryptographic software library. It enables recording of cryptographic public key protocol information. Each message received by the protocol module appends a line of the following form to the file generation set named **cryptostats**:

```
49213 525.624 127.127.4.1 message
```

The first two fields show the date (Modified Julian Day) and time (seconds and fraction past UTC midnight). The next field shows the peer address in dotted–quad notation. The final message field includes the message type and certain ancillary information. See the **Authentication Options** section for further information.

loopstats

Enables recording of loop filter statistics information. Each update of the local clock outputs a line of the following form to the file generation set named **loopstats**:

```
50935 75440.031 0.000006019 13.778190 0.000351733 0.0133806
```

The first two fields show the date (Modified Julian Day) and time (seconds and fraction past UTC midnight). The next five fields show time offset (seconds), frequency offset (parts per million – PPM), RMS jitter (seconds), Allan deviation (PPM) and clock discipline time constant.

peerstats

Enables recording of peer statistics information. This includes statistics records of all peers of a NTP server and of special signals, where present and configured. Each valid update appends a line of the following form to the current element of a file generation set named **peerstats**:

```
48773 10847.650 127.127.4.1 9714 -0.001605376 0.000000000 0.001424877 0
```

The first two fields show the date (Modified Julian Day) and time (seconds and fraction past UTC midnight). The next two fields show the peer address in dotted–quad notation and status, respectively. The status field is encoded in hex in the format described in Appendix A of the NTP specification RFC 1305. The final four fields show the offset, delay, dispersion and RMS jitter, all in seconds.

rawstats

Enables recording of raw–timestamp statistics information. This includes statistics records of all peers of a NTP server and of special signals, where present and configured. Each NTP message received from a peer or clock driver appends a line of the following form to the file generation set named **rawstats**:

```
50928 2132.543 128.4.1.1 128.4.1.20 3102453281.584327000 3102453281.5862
```

The first two fields show the date (Modified Julian Day) and time (seconds and fraction past UTC midnight). The next two fields show the remote peer or clock address followed by the local address in dotted–quad notation. The final four fields show the originate, receive, transmit and final NTP timestamps in order. The timestamp values are as received and before processing by the various data smoothing and mitigation algorithms.

sysstats

Enables recording of ntpd statistics counters on a periodic basis. Each hour a line of the following form is appended to the file generation set named **sysstats**:

```
50928 2132.543 36000 81965 0 9546 56 71793 512 540 10 147
```

The first two fields show the date (Modified Julian Day) and time (seconds and fraction past UTC midnight). The remaining ten fields show the statistics counter values accumulated since the last generated line.

Time since restart **36000**

Time in hours since the system was last rebooted.

Packets received **81965**
 Total number of packets received.

Packets processed **0**
 Number of packets received in response to previous packets sent

Current version **9546**
 Number of packets matching the current NTP version.

Previous version **56**
 Number of packets matching the previous NTP version.

Bad version **71793**
 Number of packets matching neither NTP version.

Access denied **512**
 Number of packets denied access for any reason.

Bad length or format **540**
 Number of packets with invalid length, format or port number.

Bad authentication **10**
 Number of packets not verified as authentic.

Rate exceeded **147**
 Number of packets discarded due to rate limitation.

statsdir *directory_path*

Indicates the full path of a directory where statistics files should be created (see below). This keyword allows the (otherwise constant) **filegen** filename prefix to be modified for file generation sets, which is useful for handling statistics logs.

filegen *name* [**file** *filename*] [**type** *typename*] [**link** | **nolink**] [**enable** | **disable**]

Configures setting of generation file set name. Generation file sets provide a means for handling files that are continuously growing during the lifetime of a server. Server statistics are a typical example for such files. Generation file sets provide access to a set of files used to store the actual data. At any time at most one element of the set is being written to. The type given specifies when and how data will be directed to a new element of the set. This way, information stored in elements of a file set that are currently unused are available for administrative operations without the risk of disturbing the operation of ntpd. (Most important: they can be removed to free space for new data produced.)

Note that this command can be sent from the ntpdc(1) program running at a remote location.

name This is the type of the statistics records, as shown in the **statistics** command.

file *filename*

This is the file name for the statistics records. Filenames of set members are built from three concatenated elements *file* . . . **prefix**, *file* . . . **filename** and *file* . . . **suffix**:

prefix This is a constant filename path. It is not subject to modifications via the *filegen* option. It is defined by the server, usually specified as a compile-time constant. It may, however, be configurable for individual file generation sets via other commands. For exam-

ple, the prefix used with *loopstats* and *peerstats* generation can be configured using the *statsdir* option explained above.

filename

This string is directly concatenated to the prefix mentioned above (no intervening '/'). This can be modified using the file argument to the *filegen* statement. No .. elements are allowed in this component to prevent filenames referring to parts outside the filesystem hierarchy denoted by *prefix*.

suffix This part reflects individual elements of a file set. It is generated according to the type of a file set.

type *typename*

A file generation set is characterized by its type. The following types are supported:

- none** The file set is actually a single plain file.
- pid** One element of file set is used per incarnation of a ntpd server. This type does not perform any changes to file set members during runtime, however it provides an easy way of separating files belonging to different ntpd(8) server incarnations. The set member filename is built by appending a '.' to concatenated *prefix* and *filename* strings, and appending the decimal representation of the process ID of the ntpd(8) server process.
- day** One file generation set element is created per day. A day is defined as the period between 00:00 and 24:00 UTC. The file set member suffix consists of a '.' and a day specification in the form **YYYYMMdd**. **YYYY** is a 4-digit year number (e.g., 1992). **MM** is a two digit month number. **dd** is a two digit day number. Thus, all information written at 10 December 1992 would end up in a file named *prefix filename.19921210*.
- week** Any file set member contains data related to a certain week of a year. The term week is defined by computing day-of-year modulo 7. Elements of such a file generation set are distinguished by appending the following suffix to the file set filename base: A dot, a 4-digit year number, the letter **W**, and a 2-digit week number. For example, information from January, 10th 1992 would end up in a file with suffix *.1992W1*.
- month** One generation file set element is generated per month. The file name suffix consists of a dot, a 4-digit year number, and a 2-digit month.
- year** One generation file element is generated per year. The filename suffix consists of a dot and a 4 digit year number.
- age** This type of file generation sets changes to a new element of the file set every 24 hours of server operation. The filename suffix consists of a dot, the letter **a**, and an 8-digit number. This number is taken to be the number of seconds the server is running at the start of the corresponding 24-hour period. Information is only written to a file generation by specifying **enable**; output is prevented by specifying **disable**.

link | nolinek

It is convenient to be able to access the current element of a file generation set by a fixed name. This feature is enabled by specifying **link** and disabled using **nolinek**. If **link** is specified, a hard link from the current file set element to a file without suffix is created. When there is already a file with this name and the number of links of this file is one, it is renamed appending a dot, the letter **C**, and the pid of the `ntpd(8)` server process. When the number of links is greater than one, the file is unlinked. This allows the current file to be accessed by a constant name.

enable | disable

Enables or disables the recording function.

Access Control Support

The `ntpd(8)` daemon implements a general purpose address/mask based restriction list. The list contains address/match entries sorted first by increasing address values and then by increasing mask values. A match occurs when the bitwise AND of the mask and the packet source address is equal to the bitwise AND of the mask and address in the list. The list is searched in order with the last match found defining the restriction flags associated with the entry. Additional information and examples can be found in the "Notes on Configuring NTP and Setting up a NTP Subnet" page (available as part of the HTML documentation provided in `/usr/share/doc/ntp`).

The restriction facility was implemented in conformance with the access policies for the original NSFnet backbone time servers. Later the facility was expanded to deflect cryptographic and clogging attacks. While this facility may be useful for keeping unwanted or broken or malicious clients from congesting innocent servers, it should not be considered an alternative to the NTP authentication facilities. Source address based restrictions are easily circumvented by a determined cracker.

Clients can be denied service because they are explicitly included in the restrict list created by the **restrict** command or implicitly as the result of cryptographic or rate limit violations. Cryptographic violations include certificate or identity verification failure; rate limit violations generally result from defective NTP implementations that send packets at abusive rates. Some violations cause denied service only for the offending packet, others cause denied service for a timed period and others cause the denied service for an indefinite period. When a client or network is denied access for an indefinite period, the only way at present to remove the restrictions is by restarting the server.

The Kiss-of-Death Packet

Ordinarily, packets denied service are simply dropped with no further action except incrementing statistics counters. Sometimes a more proactive response is needed, such as a server message that explicitly requests the client to stop sending and leave a message for the system operator. A special packet format has been created for this purpose called the "kiss-of-death" (KoD) packet. KoD packets have the leap bits set unsynchronized and stratum set to zero and the reference identifier field set to a four-byte ASCII code. If the **noserve** or **notrust** flag of the matching restrict list entry is set, the code is "DENY"; if the **limited** flag is set and the rate limit is exceeded, the code is "RATE". Finally, if a cryptographic violation occurs, the code is "CRYP".

A client receiving a KoD performs a set of sanity checks to minimize security exposure, then updates the stratum and reference identifier peer variables, sets the access denied (TEST4) bit in the peer flash variable and sends a message to the log. As long as the TEST4 bit is set, the client will send no further packets to the server. The only way at present to recover from this condition is to restart the protocol at both the client and server. This happens automatically at the client when the association times out. It will happen at the server only if the server operator cooperates.

Access Control Commands

discard [**average** *avg*] [**minimum** *min*] [**monitor** *prob*]

Set the parameters of the **limited** facility which protects the server from client abuse. The **average** subcommand specifies the minimum average packet spacing, while the **minimum** subcommand specifies the minimum packet spacing. Packets that violate these minima are discarded and a kiss-o'-death packet returned if enabled. The default minimum average and minimum are 5 and 2, respectively. The **monitor** subcommand specifies the probability of discard for packets that overflow the rate-control window.

restrict address [**mask** *mask*] [**ippeerlimit** *int*] [*flag . . .*]

The *address* argument expressed in dotted-quad form is the address of a host or network. Alternatively, the *address* argument can be a valid host DNS name. The *mask* argument expressed in dotted-quad form defaults to **255.255.255.255**, meaning that the *address* is treated as the address of an individual host. A default entry (address **0.0.0.0**, mask **0.0.0.0**) is always included and is always the first entry in the list. Note that text string **default**, with no mask option, may be used to indicate the default entry. The **ippeerlimit** directive limits the number of peer requests for each IP to *int*, where a value of **-1** means "unlimited", the current default. A value of **0** means "none". There would usually be at most 1 peering request per IP, but if the remote peering requests are behind a proxy there could well be more than 1 per IP. In the current implementation, **flag** always restricts access, i.e., an entry with no flags indicates that free access to the server is to be given. The flags are not orthogonal, in that more restrictive flags will often make less restrictive ones redundant. The flags can generally be classed into two categories, those which restrict time service and those which restrict informational queries and attempts to do run-time reconfiguration of the server. One or more of the following flags may be specified:

ignore Deny packets of all kinds, including `ntpq(1)` and `ntpd(1)` queries.

kod If this flag is set when an access violation occurs, a kiss-o'-death (KoD) packet is sent. KoD packets are rate limited to no more than one per second. If another KoD packet occurs within one second after the last one, the packet is dropped.

limited

Deny service if the packet spacing violates the lower limits specified in the **discard** command. A history of clients is kept using the monitoring capability of `ntpd(8)`. Thus, monitoring is always active as long as there is a restriction entry with the **limited** flag.

lowpriotrap

Declare traps set by matching hosts to be low priority. The number of traps a server can maintain is limited (the current limit is 3). Traps are usually assigned on a first come, first served basis, with later trap requestors being denied service. This flag modifies the assignment algorithm by allowing low priority traps to be overridden by later requests for normal priority traps.

noepeer

Deny ephemeral peer requests, even if they come from an authenticated source. Note that the ability to use a symmetric key for authentication may be restricted to one or more IPs or subnets via the third field of the `ntp.keys` file. This restriction is not enabled by default, to maintain backward compatibility. Expect **noepeer** to become the default in `ntp-4.4`.

nomodify

Deny `ntpq(1)` and `ntpd(1)` queries which attempt to modify the state of the server (i.e., run time reconfiguration). Queries which return information are permitted.

noquery

Deny `ntpq(1)` and `ntpd(1)` queries. Time service is not affected.

nopeer Deny unauthenticated packets which would result in mobilizing a new association. This includes broadcast and symmetric active packets when a configured association does not exist. It also includes **pool** associations, so if you want to use servers from a **pool** directive and also want to use **nopeer** by default, you'll want a **restrict source** . . . line as well that does *not* include the **nopeer** directive.

noserve

Deny all packets except `ntpq(1)` and `ntpd(1)` queries.

notrap Decline to provide mode 6 control message trap service to matching hosts. The trap service is a subsystem of the `ntpq(1)` control message protocol which is intended for use by remote event logging programs.

notrust

Deny service unless the packet is cryptographically authenticated.

ntpport

This is actually a match algorithm modifier, rather than a restriction flag. Its presence causes the restriction entry to be matched only if the source port in the packet is the standard NTP UDP port (123). Both **ntpport** and **non-ntpport** may be specified. The **ntpport** is considered more specific and is sorted later in the list.

version

Deny packets that do not match the current NTP version.

Default restriction list entries with the flags `ignore`, `interface`, `ntpport`, for each of the local host's interface addresses are inserted into the table at startup to prevent the server from attempting to synchronize to its own time. A default entry is also always present, though if it is otherwise unconfigured; no flags are associated with the default entry (i.e., everything besides your own NTP server is unrestricted).

Automatic NTP Configuration Options**Manycasting**

Manycasting is a automatic discovery and configuration paradigm new to NTPv4. It is intended as a means for a multicast client to troll the nearby network neighborhood to find cooperating manycast servers, validate them using cryptographic means and evaluate their time values with respect to other servers that might be lurking in the vicinity. The intended result is that each manycast client mobilizes client associations with some number of the "best" of the nearby manycast servers, yet automatically reconfigures to sustain this number of servers should one or another fail.

Note that the manycasting paradigm does not coincide with the anycast paradigm described in RFC-1546, which is designed to find a single server from a clique of servers providing the same service. The manycast paradigm is designed to find a plurality of redundant servers satisfying defined optimality criteria.

Manycasting can be used with either symmetric key or public key cryptography. The public key infrastructure (PKI) offers the best protection against compromised keys and is generally considered stronger, at least with relatively large key sizes. It is implemented using the Autokey protocol and the OpenSSL cryptographic library available from <http://www.openssl.org/>. The library can also be used with other NTPv4 modes as well and is highly recommended, especially for broadcast modes.

A persistent manycast client association is configured using the **manycastclient** command, which is similar to the **server** command but with a multicast (IPv4 class **D** or IPv6 prefix **FF**) group address. The IANA has designated IPv4 address 224.1.1.1 and IPv6 address FF05::101 (site local) for NTP. When more servers are needed, it broadcasts manycast client messages to this address at the minimum feasible rate and

minimum feasible time-to-live (TTL) hops, depending on how many servers have already been found. There can be as many multicast client associations as different group address, each one serving as a template for a future ephemeral unicast client/server association.

Multicast servers configured with the **multicastserver** command listen on the specified group address for multicast client messages. Note the distinction between multicast client, which actively broadcasts messages, and multicast server, which passively responds to them. If a multicast server is in scope of the current TTL and is itself synchronized to a valid source and operating at a stratum level equal to or lower than the multicast client, it replies to the multicast client message with an ordinary unicast server message.

The multicast client receiving this message mobilizes an ephemeral client/server association according to the matching multicast client template, but only if cryptographically authenticated and the server stratum is less than or equal to the client stratum. Authentication is explicitly required and either symmetric key or public key (Autokey) can be used. Then, the client polls the server at its unicast address in burst mode in order to reliably set the host clock and validate the source. This normally results in a volley of eight client/server at 2-s intervals during which both the synchronization and cryptographic protocols run concurrently. Following the volley, the client runs the NTP intersection and clustering algorithms, which act to discard all but the "best" associations according to stratum and synchronization distance. The surviving associations then continue in ordinary client/server mode.

The multicast client polling strategy is designed to reduce as much as possible the volume of multicast client messages and the effects of implosion due to near-simultaneous arrival of multicast server messages. The strategy is determined by the **multicastclient**, **tos** and **ttd** configuration commands. The multicast poll interval is normally eight times the system poll interval, which starts out at the **minpoll** value specified in the **multicastclient**, command and, under normal circumstances, increments to the **maxpoll** value specified in this command. Initially, the TTL is set at the minimum hops specified by the **ttd** command. At each retransmission the TTL is increased until reaching the maximum hops specified by this command or a sufficient number client associations have been found. Further retransmissions use the same TTL.

The quality and reliability of the suite of associations discovered by the multicast client is determined by the NTP mitigation algorithms and the **minclock** and **minsane** values specified in the **tos** configuration command. At least **minsane** candidate servers must be available and the mitigation algorithms produce at least **minclock** survivors in order to synchronize the clock. Byzantine agreement principles require at least four candidates in order to correctly discard a single falseticker. For legacy purposes, **minsane** defaults to 1 and **minclock** defaults to 3. For multicast service **minsane** should be explicitly set to 4, assuming at least that number of servers are available.

If at least **minclock** servers are found, the multicast poll interval is immediately set to eight times **maxpoll**. If less than **minclock** servers are found when the TTL has reached the maximum hops, the multicast poll interval is doubled. For each transmission after that, the poll interval is doubled again until reaching the maximum of eight times **maxpoll**. Further transmissions use the same poll interval and TTL values. Note that while all this is going on, each client/server association found is operating normally it the system poll interval.

Administratively scoped multicast boundaries are normally specified by the network router configuration and, in the case of IPv6, the link/site scope prefix. By default, the increment for TTL hops is 32 starting from 31; however, the **ttd** configuration command can be used to modify the values to match the scope rules.

It is often useful to narrow the range of acceptable servers which can be found by multicast client associations. Because multicast servers respond only when the client stratum is equal to or greater than the server stratum, primary (stratum 1) servers will find only primary servers in TTL range, which is probably the most common objective. However, unless configured otherwise, all multicast clients in TTL range will eventually find all primary servers in TTL range, which is probably not the most common objective in large networks. The **tos** command can be used to modify this behavior. Servers with stratum below **floor** or above

ceiling specified in the **tos** command are strongly discouraged during the selection process; however, these servers may be temporally accepted if the number of servers within TTL range is less than **minclock**.

The above actions occur for each manycast client message, which repeats at the designated poll interval. However, once the ephemeral client association is mobilized, subsequent manycast server replies are discarded, since that would result in a duplicate association. If during a poll interval the number of client associations falls below **minclock**, all manycast client prototype associations are reset to the initial poll interval and TTL hops and operation resumes from the beginning. It is important to avoid frequent manycast client messages, since each one requires all manycast servers in TTL range to respond. The result could well be an implosion, either minor or major, depending on the number of servers in range. The recommended value for **maxpoll** is 12 (4,096 s).

It is possible and frequently useful to configure a host as both manycast client and manycast server. A number of hosts configured this way and sharing a common group address will automatically organize themselves in an optimum configuration based on stratum and synchronization distance. For example, consider an NTP subnet of two primary servers and a hundred or more dependent clients. With two exceptions, all servers and clients have identical configuration files including both **multicastclient** and **multicastserver** commands using, for instance, multicast group address 239.1.1.1. The only exception is that each primary server configuration file must include commands for the primary reference source such as a GPS receiver.

The remaining configuration files for all secondary servers and clients have the same contents, except for the **tos** command, which is specific for each stratum level. For stratum 1 and stratum 2 servers, that command is not necessary. For stratum 3 and above servers the **floor** value is set to the intended stratum number. Thus, all stratum 3 configuration files are identical, all stratum 4 files are identical and so forth.

Once operations have stabilized in this scenario, the primary servers will find the primary reference source and each other, since they both operate at the same stratum (1), but not with any secondary server or client, since these operate at a higher stratum. The secondary servers will find the servers at the same stratum level. If one of the primary servers loses its GPS receiver, it will continue to operate as a client and other clients will time out the corresponding association and re-associate accordingly.

Some administrators prefer to avoid running `ntpd(8)` continuously and run either `sntp(1)` or `ntpd(8) -q` as a cron job. In either case the servers must be configured in advance and the program fails if none are available when the cron job runs. A really slick application of manycast is with `ntpd(8) -q`. The program wakes up, scans the local landscape looking for the usual suspects, selects the best from among the rascals, sets the clock and then departs. Servers do not have to be configured in advance and all clients throughout the network can have the same configuration file.

Manycast Interactions with Autokey

Each time a manycast client sends a client mode packet to a multicast group address, all manycast servers in scope generate a reply including the host name and status word. The manycast clients then run the Autokey protocol, which collects and verifies all certificates involved. Following the burst interval all but three survivors are cast off, but the certificates remain in the local cache. It often happens that several complete signing trails from the client to the primary servers are collected in this way.

About once an hour or less often if the poll interval exceeds this, the client regenerates the Autokey key list. This is in general transparent in client/server mode. However, about once per day the server private value used to generate cookies is refreshed along with all manycast client associations. In this case all cryptographic values including certificates is refreshed. If a new certificate has been generated since the last refresh epoch, it will automatically revoke all prior certificates that happen to be in the certificate cache. At the same time, the manycast scheme starts all over from the beginning and the expanding ring shrinks to the minimum and increments from there while collecting all servers in scope.

Broadcast Options

tos [**bcpollbstep** *gate*]

This command provides a way to delay, by the specified number of broadcast poll intervals, believing backward time steps from a broadcast server. Broadcast time networks are expected to be trusted. In the event a broadcast server's time is stepped backwards, there is clear benefit to having the clients notice this change as soon as possible. Attacks such as replay attacks can happen, however, and even though there are a number of protections built in to broadcast mode, attempts to perform a replay attack are possible. This value defaults to 0, but can be changed to any number of poll intervals between 0 and 4.

Manycast Options

tos [**ceiling** *ceiling* | **cohort** { 0 | 1 } | **floor** *floor* | **minclock** *minclock* | **minsane** *minsane*]

This command affects the clock selection and clustering algorithms. It can be used to select the quality and quantity of peers used to synchronize the system clock and is most useful in manycast mode. The variables operate as follows:

ceiling *ceiling*

Peers with strata above **ceiling** will be discarded if there are at least **minclock** peers remaining. This value defaults to 15, but can be changed to any number from 1 to 15.

cohort {0|1}

This is a binary flag which enables (0) or disables (1) manycast server replies to manycast clients with the same stratum level. This is useful to reduce implosions where large numbers of clients with the same stratum level are present. The default is to enable these replies.

floor *floor*

Peers with strata below **floor** will be discarded if there are at least **minclock** peers remaining. This value defaults to 1, but can be changed to any number from 1 to 15.

minclock *minclock*

The clustering algorithm repeatedly casts out outlier associations until no more than **minclock** associations remain. This value defaults to 3, but can be changed to any number from 1 to the number of configured sources.

minsane *minsane*

This is the minimum number of candidates available to the clock selection algorithm in order to produce one or more truechimers for the clustering algorithm. If fewer than this number are available, the clock is undisciplined and allowed to run free. The default is 1 for legacy purposes. However, according to principles of Byzantine agreement, **minsane** should be at least 4 in order to detect and discard a single falseticker.

t1 *hop* . . .

This command specifies a list of TTL values in increasing order, up to 8 values can be specified. In manycast mode these values are used in turn in an expanding-ring search. The default is eight multiples of 32 starting at 31.

Reference Clock Support

The NTP Version 4 daemon supports some three dozen different radio, satellite and modem reference clocks plus a special pseudo-clock used for backup or when no other clock source is available. Detailed descriptions of individual device drivers and options can be found in the "Reference Clock Drivers" page (available as part of the HTML documentation provided in `/usr/share/doc/ntp`). Additional information can be found in the pages linked there, including the "Debugging Hints for Reference Clock Drivers" and "How To

Write a Reference Clock Driver" pages (available as part of the HTML documentation provided in `/usr/share/doc/ntp`). In addition, support for a PPS signal is available as described in the "Pulse-per-second (PPS) Signal Interfacing" page (available as part of the HTML documentation provided in `/usr/share/doc/ntp`). Many drivers support special line discipline/streams modules which can significantly improve the accuracy using the driver. These are described in the "Line Disciplines and Streams Drivers" page (available as part of the HTML documentation provided in `/usr/share/doc/ntp`).

A reference clock will generally (though not always) be a radio timecode receiver which is synchronized to a source of standard time such as the services offered by the NRC in Canada and NIST and USNO in the US. The interface between the computer and the timecode receiver is device dependent, but is usually a serial port. A device driver specific to each reference clock must be selected and compiled in the distribution; however, most common radio, satellite and modem clocks are included by default. Note that an attempt to configure a reference clock when the driver has not been compiled or the hardware port has not been appropriately configured results in a scolding remark to the system log file, but is otherwise non hazardous.

For the purposes of configuration, `ntpd(8)` treats reference clocks in a manner analogous to normal NTP peers as much as possible. Reference clocks are identified by a syntactically correct but invalid IP address, in order to distinguish them from normal NTP peers. Reference clock addresses are of the form `127.127.t.u`, where *t* is an integer denoting the clock type and *u* indicates the unit number in the range 0–3. While it may seem overkill, it is in fact sometimes useful to configure multiple reference clocks of the same type, in which case the unit numbers must be unique.

The **server** command is used to configure a reference clock, where the *address* argument in that command is the clock address. The **key**, **version** and **ttl** options are not used for reference clock support. The **mode** option is added for reference clock support, as described below. The **prefer** option can be useful to persuade the server to cherish a reference clock with somewhat more enthusiasm than other reference clocks or peers. Further information on this option can be found in the "Mitigation Rules and the prefer Keyword" (available as part of the HTML documentation provided in `/usr/share/doc/ntp`) page. The **minpoll** and **maxpoll** options have meaning only for selected clock drivers. See the individual clock driver document pages for additional information.

The **fudge** command is used to provide additional information for individual clock drivers and normally follows immediately after the **server** command. The *address* argument specifies the clock address. The **refid** and **stratum** options can be used to override the defaults for the device. There are two optional device-dependent time offsets and four flags that can be included in the **fudge** command as well.

The stratum number of a reference clock is by default zero. Since the `ntpd(8)` daemon adds one to the stratum of each peer, a primary server ordinarily displays an external stratum of one. In order to provide engineered backups, it is often useful to specify the reference clock stratum as greater than zero. The **stratum** option is used for this purpose. Also, in cases involving both a reference clock and a pulse-per-second (PPS) discipline signal, it is useful to specify the reference clock identifier as other than the default, depending on the driver. The **refid** option is used for this purpose. Except where noted, these options apply to all clock drivers.

Reference Clock Commands

server *127.127.t.u* [**prefer**] [**mode** *int*] [**minpoll** *int*] [**maxpoll** *int*]

This command can be used to configure reference clocks in special ways. The options are interpreted as follows:

prefer Marks the reference clock as preferred. All other things being equal, this host will be chosen for synchronization among a set of correctly operating hosts. See the "Mitigation Rules and the prefer Keyword" page (available as part of the HTML documentation provided in `/usr/share/doc/ntp`) for further information.

mode *int*

Specifies a mode number which is interpreted in a device-specific fashion. For instance, it selects a dialing protocol in the ACTS driver and a device subtype in the parse drivers.

minpoll *int***maxpoll** *int*

These options specify the minimum and maximum polling interval for reference clock messages, as a power of 2 in seconds. For most directly connected reference clocks, both **minpoll** and **maxpoll** default to 6 (64 s). For modem reference clocks, **minpoll** defaults to 10 (17.1 m) and **maxpoll** defaults to 14 (4.5 h). The allowable range is 4 (16 s) to 17 (36.4 h) inclusive.

fudge 127.127.t.u [**time1** *sec*] [**time2** *sec*] [**stratum** *int*] [**refid** *string*] [**mode** *int*] [**flag1** 0 | 1] [**flag2** 0 | 1] [**flag3** 0 | 1] [**flag4** 0 | 1]

This command can be used to configure reference clocks in special ways. It must immediately follow the **server** command which configures the driver. Note that the same capability is possible at run time using the `ntpd(1)` program. The options are interpreted as follows:

time1 *sec*

Specifies a constant to be added to the time offset produced by the driver, a fixed-point decimal number in seconds. This is used as a calibration constant to adjust the nominal time offset of a particular clock to agree with an external standard, such as a precision PPS signal. It also provides a way to correct a systematic error or bias due to serial port or operating system latencies, different cable lengths or receiver internal delay. The specified offset is in addition to the propagation delay provided by other means, such as internal DIPswitches. Where a calibration for an individual system and driver is available, an approximate correction is noted in the driver documentation pages. Note: in order to facilitate calibration when more than one radio clock or PPS signal is supported, a special calibration feature is available. It takes the form of an argument to the **enable** command described in **Miscellaneous Options** page and operates as described in the "Reference Clock Drivers" page (available as part of the HTML documentation provided in `/usr/share/doc/ntp`).

time2 *secs*

Specifies a fixed-point decimal number in seconds, which is interpreted in a driver-dependent way. See the descriptions of specific drivers in the "Reference Clock Drivers" page (available as part of the HTML documentation provided in `/usr/share/doc/ntp`).

stratum *int*

Specifies the stratum number assigned to the driver, an integer between 0 and 15. This number overrides the default stratum number ordinarily assigned by the driver itself, usually zero.

refid *string*

Specifies an ASCII string of from one to four characters which defines the reference identifier used by the driver. This string overrides the default identifier ordinarily assigned by the driver itself.

mode *int*

Specifies a mode number which is interpreted in a device-specific fashion. For instance, it selects a dialing protocol in the ACTS driver and a device subtype in the parse drivers.

flag1 0 | 1

flag2 0 | 1

flag3 0 | 1

flag4 0 | 1

These four flags are used for customizing the clock driver. The interpretation of these values, and whether they are used at all, is a function of the particular clock driver. However, by convention **flag4** is used to enable recording monitoring data to the **clockstats** file configured with the **filegen** command. Further information on the **filegen** command can be found in **Monitoring Options**.

Miscellaneous Options

broadcastdelay *seconds*

The broadcast and multicast modes require a special calibration to determine the network delay between the local and remote servers. Ordinarily, this is done automatically by the initial protocol exchanges between the client and server. In some cases, the calibration procedure may fail due to network or server access controls, for example. This command specifies the default delay to be used under these circumstances. Typically (for Ethernet), a number between 0.003 and 0.007 seconds is appropriate. The default when this command is not used is 0.004 seconds.

calldelay *delay*

This option controls the delay in seconds between the first and second packets sent in burst or iburst mode to allow additional time for a modem or ISDN call to complete.

driftfile *driftfile*

This command specifies the complete path and name of the file used to record the frequency of the local clock oscillator. This is the same operation as the **-f** command line option. If the file exists, it is read at startup in order to set the initial frequency and then updated once per hour with the current frequency computed by the daemon. If the file name is specified, but the file itself does not exist, the starts with an initial frequency of zero and creates the file when writing it for the first time. If this command is not given, the daemon will always start with an initial frequency of zero.

The file format consists of a single line containing a single floating point number, which records the frequency offset measured in parts-per-million (PPM). The file is updated by first writing the current drift value into a temporary file and then renaming this file to replace the old version. This implies that `ntpd(8)` must have write permission for the directory the drift file is located in, and that file system links, symbolic or otherwise, should be avoided.

dscp *value*

This option specifies the Differentiated Services Control Point (DSCP) value, a 6-bit code. The default value is 46, signifying Expedited Forwarding.

enable [**auth** | **bclient** | **calibrate** | **kernel** | **mode7** | **monitor** | **ntp** | **stats** | **peer_clear_digest_early** | **unpeer_crypto_early** | **unpeer_crypto_nak_early** | **unpeer_digest_early**]

disable [**auth** | **bclient** | **calibrate** | **kernel** | **mode7** | **monitor** | **ntp** | **stats** | **peer_clear_digest_early** | **unpeer_crypto_early** | **unpeer_crypto_nak_early** | **unpeer_digest_early**]

Provides a way to enable or disable various server options. Flags not mentioned are unaffected. Note that all of these flags can be controlled remotely using the `ntpdc(1)` utility program.

auth Enables the server to synchronize with unconfigured peers only if the peer has been correctly authenticated using either public key or private key cryptography. The default for this flag is **enable**.

bclient

Enables the server to listen for a message from a broadcast or multicast server, as in the **multicastclient** command with default address. The default for this flag is **disable**.

calibrate

Enables the calibrate feature for reference clocks. The default for this flag is **disable**.

kernel Enables the kernel time discipline, if available. The default for this flag is **enable** if support is available, otherwise **disable**.

mode7 Enables processing of NTP mode 7 implementation-specific requests which are used by the deprecated `ntpd(1)` program. The default for this flag is **disable**. This flag is excluded from runtime configuration using `ntp(1)`. The `ntp(1)` program provides the same capabilities as `ntpd(1)` using standard mode 6 requests.

monitor

Enables the monitoring facility. See the `ntpd(1)` program and the **monlist** command or further information. The default for this flag is **enable**.

ntp Enables time and frequency discipline. In effect, this switch opens and closes the feedback loop, which is useful for testing. The default for this flag is **enable**.

peer_clear_digest_early

By default, if `ntpd(8)` is using autokey and it receives a crypto-NAK packet that passes the duplicate packet and origin timestamp checks the peer variables are immediately cleared. While this is generally a feature as it allows for quick recovery if a server key has changed, a properly forged and appropriately delivered crypto-NAK packet can be used in a DoS attack. If you have active noticeable problems with this type of DoS attack then you should consider disabling this option. You can check your **peerstats** file for evidence of any of these attacks. The default for this flag is **enable**.

stats Enables the statistics facility. See the **Monitoring Options** section for further information. The default for this flag is **disable**.

unpeer_crypto_early

By default, if `ntpd(8)` receives an autokey packet that fails TEST9, a crypto failure, the association is immediately cleared. This is almost certainly a feature, but if, in spite of the current recommendation of not using autokey, you are using autokey you are seeing this sort of DoS attack disabling this flag will delay tearing down the association until the reachability counter becomes zero. You can check your **peerstats** file for evidence of any of these attacks. The default for this flag is **enable**.

unpeer_crypto_nak_early

By default, if `ntpd(8)` receives a crypto-NAK packet that passes the duplicate packet and origin timestamp checks the association is immediately cleared. While this is generally a feature as it allows for quick recovery if a server key has changed, a properly forged and appropriately delivered crypto-NAK packet can be used in a DoS attack. If you have active noticeable problems with this type of DoS attack then you should consider disabling this option. You can check your **peerstats** file for evidence of any of these attacks. The default for this flag is **enable**.

unpeer_digest_early

By default, if `ntpd(8)` receives what should be an authenticated packet that passes other packet sanity checks but contains an invalid digest the association is immediately cleared. While this is generally a feature as it allows for quick recovery, if this type of packet is carefully forged and sent during an appropriate window it can be used for a

DoS attack. If you have active noticeable problems with this type of DoS attack then you should consider disabling this option. You can check your **peerstats** file for evidence of any of these attacks. The default for this flag is **enable**.

includefile *includefile*

This command allows additional configuration commands to be included from a separate file. Include files may be nested to a depth of five; upon reaching the end of any include file, command processing resumes in the previous configuration file. This option is useful for sites that run `ntpd(8)` on multiple hosts, with (mostly) common options (e.g., a restriction list).

interface [**listen** | **ignore** | **drop**] [**all** | **ipv4** | **ipv6** | **wildcard name** | *address* [/ *prefixlen*]]

The **interface** directive controls which network addresses `ntpd(8)` opens, and whether input is dropped without processing. The first parameter determines the action for addresses which match the second parameter. The second parameter specifies a class of addresses, or a specific interface name, or an address. In the address case, *prefixlen* determines how many bits must match for this rule to apply. **ignore** prevents opening matching addresses, **drop** causes `ntpd(8)` to open the address and drop all received packets without examination. Multiple **interface** directives can be used. The last rule which matches a particular address determines the action for it. **interface** directives are disabled if any **-I**, **--interface**, **-L**, or **--novirtualips** command-line options are specified in the configuration file, all available network addresses are opened. The **nic** directive is an alias for **interface**.

leapfile *leapfile*

This command loads the IERS leapseconds file and initializes the leapsecond values for the next leapsecond event, leapfile expiration time, and TAI offset. The file can be obtained directly from the

IERS	at
https://hpiers.obspm.fr/iers/bul/bulc/ntp/leap-seconds.list	or
ftp://hpiers.obspm.fr/iers/bul/bulc/ntp/leap-seconds.list	The

leapfile is scanned when `ntpd(8)` processes the **leapfile** directive or when `ntpd` detects that the *leapfile* has changed. `ntpd` checks once a day to see if the *leapfile* has changed. The `update-leap(1update_leapmdoc)` script can be run to see if the *leapfile* should be updated.

leapsmearinterval *seconds*

This EXPERIMENTAL option is only available if `ntpd(8)` was built with the **--enable-leap-smear** option to the **configure** script. It specifies the interval over which a leap second correction will be applied. Recommended values for this option are between 7200 (2 hours) and 86400 (24 hours). **DO NOT USE THIS OPTION ON PUBLIC-ACCESS SERVERS!** See <http://bugs.ntp.org/2855> for more information.

logconfig *configkeyword*

This command controls the amount and type of output written to the system `syslog(3)` facility or the alternate **logfile** log file. By default, all output is turned on. All *configkeyword* keywords can be prefixed with '=', '+', and '-', where '=' sets the `syslog(3)` priority mask, '+' adds and '-' removes messages. `syslog(3)` messages can be controlled in four classes (**clock**, **peer**, **sys** and **sync**). Within these classes four types of messages can be controlled: informational messages (**info**), event messages (**events**), statistics messages (**statistics**) and status messages (**status**).

Configuration keywords are formed by concatenating the message class with the event class. The **all** prefix can be used instead of a message class. A message class may also be followed by the **all** keyword to enable/disable all messages of the respective message class. Thus, a minimal log configuration could look like this:


```
logconfig =syncstatus +sysevents
```

This would just list the synchronizations state of `ntpd(8)` and the major system events. For a simple reference server, the following minimum message configuration could be useful:

```
logconfig =syncall +clockall
```

This configuration will list all clock information and synchronization information. All other events and messages about peers, system events and so on is suppressed.

logfile *logfile*

This command specifies the location of an alternate log file to be used instead of the default system `syslog(3)` facility. This is the same operation as the `-l` command line option.

mr [**maxdepth** *count* | **maxmem** *kilobytes* | **mindepth** *count* | **maxage** *seconds* | **initialloc** *count* | **initmem** *kilobytes* | **incalloc** *count* | **incmem** *kilobytes*]

Controls size limits of the monitoring facility's Most Recently Used (MRU) list of client addresses, which is also used by the rate control facility.

maxdepth *count*

maxmem *kilobytes*

Equivalent upper limits on the size of the MRU list, in terms of entries or kilobytes. The actual limit will be up to **incalloc** entries or **incmem** kilobytes larger. As with all of the **mr** options offered in units of entries or kilobytes, if both **maxdepth** and **maxmem** are used, the last one used controls. The default is 1024 kilobytes.

mindepth *count*

Lower limit on the MRU list size. When the MRU list has fewer than **mindepth** entries, existing entries are never removed to make room for newer ones, regardless of their age. The default is 600 entries.

maxage *seconds*

Once the MRU list has **mindepth** entries and an additional client is to be added to the list, if the oldest entry was updated more than **maxage** seconds ago, that entry is removed and its storage is reused. If the oldest entry was updated more recently the MRU list is grown, subject to **maxdepth** / **maxmem**. The default is 64 seconds.

initialloc *count*

initmem *kilobytes*

Initial memory allocation at the time the monitoring facility is first enabled, in terms of the number of entries or kilobytes. The default is 4 kilobytes.

incalloc *count*

incmem *kilobytes*

Size of additional memory allocations when growing the MRU list, in entries or kilobytes. The default is 4 kilobytes.

nonvolatile *threshold*

Specify the *threshold* delta in seconds before an hourly change to the **driftfile** (frequency file) will be written, with a default value of $1e-7$ (0.1 PPM). The frequency file is inspected each hour. If the difference between the current frequency and the last value written exceeds the threshold, the file is written and the **threshold** becomes the new threshold value. If the threshold is not exceeded, it is reduced by half. This is intended to reduce the number of file writes for embedded systems with nonvolatile memory.

phone *dial . . .*

This command is used in conjunction with the ACTS modem driver (type 18) or the JJY driver (type 40, mode 100 – 180). For the ACTS modem driver (type 18), the arguments consist of a maximum of 10 telephone numbers used to dial USNO, NIST, or European time service. For the JJY driver (type 40 mode 100 – 180), the argument is one telephone number used to dial the telephone JJY service. The Hayes command ATDT is normally prepended to the number. The number can contain other modem control codes as well.

reset [**allpeers**] [**auth**] [**ctl**] [**io**] [**mem**] [**sys**] [**timer**]

Reset one or more groups of counters maintained by **ntpd** and exposed by **ntpq** and **ntpd**.

rlimit [**memlock** *Nmegabytes* | **stacksize** *N4kPages* **filenum** *Nfiledescriptors*]**memlock** *Nmegabytes*

Specify the number of megabytes of memory that should be allocated and locked. Probably only available under Linux, this option may be useful when dropping root (the **-i** option). The default is 32 megabytes on non-Linux machines, and **-1** under Linux. **-1** means "do not lock the process into memory". **0** means "lock whatever memory the process wants into memory".

stacksize *N4kPages*

Specifies the maximum size of the process stack on systems with the **mlockall()** function. Defaults to 50 4k pages (200 4k pages in OpenBSD).

filenum *Nfiledescriptors*

Specifies the maximum number of file descriptors **ntpd** may have open at once. Defaults to the system default.

saveconfigdir *directory_path*

Specify the directory in which to write configuration snapshots requested with **ntpq -s saveconfig** command. If **saveconfigdir** does not appear in the configuration file, **saveconfig** requests are rejected by **ntpd**.

saveconfig *filename*

Write the current configuration, including any runtime modifications given with **:config** or **config-from-file** to the **ntpd** host's *filename* in the **saveconfigdir**. This command will be rejected unless the **saveconfigdir** directive appears in **ntpd -s** configuration file. *filename* can use **strftime(3)** format directives to substitute the current date and time, for example, **saveconfig ntp-%Y%m%d-%H%M%S.conf**. The filename used is stored in the system variable **savedconfig**. Authentication is required.

setvar *variable* [**default**]

This command adds an additional system variable. These variables can be used to distribute additional information such as the access policy. If the variable of the form *name=value* is followed by the **default** keyword, the variable will be listed as part of the default system variables (**ntpq(1) rv** command)). These additional variables serve informational purposes only. They are not related to the protocol other than they can be listed. The known protocol variables will always override any variables defined via the **setvar** mechanism. There are three special variables that contain the names of all variable of the same group. The *sys_var_list* holds the names of all system variables. The *peer_var_list* holds the names of all peer variables and the *clock_var_list* holds the names of the reference clock variables.

sysinfo

Display operational summary.

sysstats

Show statistics counters maintained in the protocol module.

tinker [**allan** *allan* | **dispersion** *dispersion* | **freq** *freq* | **huffpuff** *huffpuff* | **panic** *panic* | **step** *step* | **stepback** *stepback* | **stepfwd** *stepfwd* | **stepout** *stepout*]

This command can be used to alter several system variables in very exceptional circumstances. It should occur in the configuration file before any other configuration options. The default values of these variables have been carefully optimized for a wide range of network speeds and reliability expectations. In general, they interact in intricate ways that are hard to predict and some combinations can result in some very nasty behavior. Very rarely is it necessary to change the default values; but, some folks cannot resist twisting the knobs anyway and this command is for them. Emphasis added: twisters are on their own and can expect no help from the support group.

The variables operate as follows:

allan *allan*

The argument becomes the new value for the minimum Allan intercept, which is a parameter of the PLL/FLL clock discipline algorithm. The value in log2 seconds defaults to 7 (1024 s), which is also the lower limit.

dispersion *dispersion*

The argument becomes the new value for the dispersion increase rate, normally .000015 s/s.

freq *freq*

The argument becomes the initial value of the frequency offset in parts-per-million. This overrides the value in the frequency file, if present, and avoids the initial training state if it is not.

huffpuff *huffpuff*

The argument becomes the new value for the experimental huff-n'-puff filter span, which determines the most recent interval the algorithm will search for a minimum delay. The lower limit is 900 s (15 m), but a more reasonable value is 7200 (2 hours). There is no default, since the filter is not enabled unless this command is given.

panic *panic*

The argument is the panic threshold, normally 1000 s. If set to zero, the panic sanity check is disabled and a clock offset of any value will be accepted.

step *step*

The argument is the step threshold, which by default is 0.128 s. It can be set to any positive number in seconds. If set to zero, step adjustments will never occur. Note: The kernel time discipline is disabled if the step threshold is set to zero or greater than the default.

stepback *stepback*

The argument is the step threshold for the backward direction, which by default is 0.128 s. It can be set to any positive number in seconds. If both the forward and backward step thresholds are set to zero, step adjustments will never occur. Note: The kernel time discipline is disabled if each direction of step threshold are either set to zero or greater than .5 second.

stepfwd *stepfwd*

As for stepback, but for the forward direction.

stepout *stepout*

The argument is the stepout timeout, which by default is 900 s. It can be set to any positive number in seconds. If set to zero, the stepout pulses will not be suppressed.

writevar *assocID name = value [, ...]*

Write (create or update) the specified variables. If the **assocID** is zero, the variables are from the system variables name space, otherwise they are from the peer variables name space. The **assocID** is required, as the same name can occur in both name spaces.

trap *host_address [port port_number] [interface interface_address]*

This command configures a trap receiver at the given host address and port number for sending messages with the specified local interface address. If the port number is unspecified, a value of 18447 is used. If the interface address is not specified, the message is sent with a source address of the local interface the message is sent through. Note that on a multihomed host the interface used may vary from time to time with routing changes.

ttd *hop ...*

This command specifies a list of TTL values in increasing order. Up to 8 values can be specified. In **manycast** mode these values are used in turn in an expanding-ring search. The default is eight multiples of 32 starting at 31.

The trap receiver will generally log event messages and other information from the server in a log file. While such monitor programs may also request their own trap dynamically, configuring a trap receiver will ensure that no messages are lost when the server is started.

hop *...*

This command specifies a list of TTL values in increasing order, up to 8 values can be specified. In **manycast** mode these values are used in turn in an expanding-ring search. The default is eight multiples of 32 starting at 31.

OPTIONS

--help

Display usage information and exit.

--more-help

Pass the extended usage information through a pager.

--version [*v|c|n*]

Output version of program and exit. The default mode is 'v', a simple version. The 'c' mode will print copyright information and 'n' will print the full copyright notice.

OPTION PRESETS

Any option that is not marked as *not presettable* may be preset by loading values from environment variables named:

NTP_CONF_<option-name> or **NTP_CONF**

ENVIRONMENT

See **OPTION PRESETS** for configuration environment variables.

FILES

<code>/etc/ntp.conf</code>	the default name of the configuration file
<code>ntp.keys</code>	private MD5 keys
<code>ntpkey</code>	RSA private key

<code>ntpkey_host</code>	RSA public key
<code>ntp_dh</code>	Diffie–Hellman agreement parameters

EXIT STATUS

One of the following exit values will be returned:

0 (EXIT_SUCCESS)

Successful program execution.

1 (EXIT_FAILURE)

The operation failed or the command syntax was not valid.

70 (EX_SOFTWARE)

libopts had an internal operational error. Please report it to autogen-users@lists.sourceforge.net. Thank you.

SEE ALSO

`ntpd(8)`, `ntpdc(1)`, `ntpq(1)`

In addition to the manual pages provided, comprehensive documentation is available on the world wide web at <http://www.ntp.org/>. A snapshot of this documentation is available in HTML format in `/usr/share/doc/ntp`.

David L. Mills, *Network Time Protocol (Version 4)*, RFC5905.

AUTHORS

The University of Delaware and Network Time Foundation

COPYRIGHT

Copyright (C) 1992–2017 The University of Delaware and Network Time Foundation all rights reserved. This program is released under the terms of the NTP license, [<http://ntp.org/license>](http://ntp.org/license).

BUGS

The syntax checking is not picky; some combinations of ridiculous and even hilarious options and modes may not be detected.

The `ntpkey_host` files are really digital certificates. These should be obtained via secure directory services when they become universally available.

Please send bug reports to: <http://bugs.ntp.org>, bugs@ntp.org

NOTES

This document was derived from FreeBSD.

This manual page was *AutoGen*-erated from the `ntp.conf` option definitions.