



## 11. nslookup

### Contents:

[Is nslookup a Good Tool?](#)

[Interactive Versus Noninteractive](#)

[Option Settings](#)

[Avoiding the Search List](#)

[Common Tasks](#)

[Less Common Tasks](#)

[Troubleshooting nslookup Problems](#)

[Best of the Net](#)

*"Don't stand chattering to yourself like that," Humpty Dumpty said, looking at her for the first time, "but tell me your name and your business."*

*"My name is Alice, but – "*

*"It's a stupid name enough!" Humpty Dumpty interrupted impatiently. "What does it mean?"*

*"Must a name mean something?" Alice asked doubtfully.*

*"Of course it must," Humpty Dumpty said with a short laugh...*

To be proficient at troubleshooting name server problems, you'll need a special tool to make DNS queries, one that gives you complete control. We'll cover **nslookup** in this chapter because it's distributed with BIND and with many vendors' systems. If you're the explorer type, you might also check out **dig**; it provides similar functionality, and some people like its user interface better. You can pick up source for **dig** from the *tools* directory (BIND 4) or *src/bin* directory (BIND 8) of the BIND distribution.

Note that this chapter isn't comprehensive; there are aspects of **nslookup** – mostly obscure and seldom used – that we won't cover. You can always consult the manual pages for those.

### 11.1 Is nslookup a Good Tool?

Much of the time you'll use **nslookup** to make queries, in the same way the resolver makes them. Sometimes, though, you'll use **nslookup** to query other name servers as a name server would, instead. Which one you emulate will depend on the problem you're trying to debug. You might wonder, "How accurately does **nslookup** emulate a resolver or a name server? Does **nslookup** actually use the BIND resolver library routines?" No, **nslookup** uses its own routines for querying name servers, but those routines are based on the resolver routines. Consequently, **nslookup**'s behavior is very similar to the resolver's behavior, but it does differ slightly. We'll point out some of those differences. As for emulating

name server behavior, **nslookup** allows us to query another server with the same query packet that a name server would use, but the retransmission scheme is quite different. Like a name server, though, **nslookup** can pull a copy of the zone data. So **nslookup** does not exactly emulate either the resolver or the name server, but it does emulate them well enough to make a good troubleshooting tool. Let's delve into those differences we alluded to.

### 11.1.1 Multiple Servers

**nslookup** only talks to one name server at a time. This is the biggest difference between **nslookup**'s behavior and the resolver's behavior. The resolver makes use of each *nameserver* entry in *resolv.conf*. If there are two *nameserver* lines in *resolv.conf*, the resolver tries the first name server, then the second, then the first, then the second, until it receives a response or it gives up. The resolver does this for every query. On the other hand, **nslookup** tries the first name server in *resolv.conf* and keeps retrying until it finally gives up on the first name server and tries the second. Once it gets a response, it locks onto that server and doesn't try the other. But, you *want* your troubleshooting tool to talk only with one name server, so you can reduce the number of variables when analyzing a problem. If **nslookup** used more than one name server, you wouldn't have as much control over your troubleshooting session. So, talking to only one server is the right thing for a troubleshooting tool to do.

### 11.1.2 Timeouts

The **nslookup** timeouts match the resolver timeouts when the resolver is only querying one name server. A name server's timeouts, however, are based on how quickly the remote server answered the last query, a dynamic measure. **nslookup** will never match name server timeouts, but that's not a problem either. When you're querying remote name servers with **nslookup**, you probably only care *what* the response was, not how long it took.

### 11.1.3 Domain Searches

**nslookup** implements the search list just as the resolver code does. Name servers don't implement search lists, so, to act like a name server, the **nslookup** search function must be turned off – more on that later.

### 11.1.4 Zone Transfers

**nslookup** will do zone transfers just like a name server. Unlike the name server, **nslookup** does not check SOA serial numbers before pulling the zone data; you'll have to do that manually, if you want to.

### 11.1.5 Using NIS and */etc/hosts*

This last point doesn't compare **nslookup** to the resolver or name server but to ways of looking up names in general. **nslookup**, as distributed from the Internet Software Consortium, only uses DNS; it won't use NIS or */etc/hosts*. Most applications will use DNS, NIS, or */etc/hosts*. Don't look to **nslookup** to help you find your lookup problem unless your host is really configured to use name servers.

---

10.12 Load Sharing  
Between Mirrored Servers

11.2 Interactive Versus  
Noninteractive

---

[ [Library Home](#) | [DNS & BIND](#) | [TCP/IP](#) | [sendmail](#) | [sendmail Reference](#) | [Firewalls](#) | [Practical Security](#) ]




---

## Chapter 11 nslookup

---

### 11.2 Interactive Versus Noninteractive

Let's start our tutorial on **nslookup** by looking at how to start it and how to exit from it. **nslookup** can be run either interactively or noninteractively. If you only want to look up one piece of data, use the noninteractive form. If you plan on doing something more extensive, like changing servers or options, then use an interactive session.

To start an interactive session, just type **nslookup**:

```
% nslookup
Default Server:  terminator.movie.edu
Address:  0.0.0.0

> ^D
```

If you need help, type **?** or **help**. When you want to exit, type **^D** (control-D). If you try to exit from **nslookup** by interrupting it, with **^C** (or whatever your interrupt character is), you won't get very far. **nslookup** catches the interrupt, stops whatever it is doing (like a zone transfer), and gives you the **>** prompt.

For a noninteractive lookup, include the name you are looking up on the command line:

```
% nslookup carrie
Server:  terminator.movie.edu
Address:  0.0.0.0

Name:    carrie.movie.edu
Address: 192.253.253.4
```

---

11.1 Is nslookup a Good  
Tool?

11.3 Option Settings

---



## Chapter 11 nslookup

### 11.3 Option Settings

**nslookup** has its own set of dials and knobs, called option settings. All of the option settings can be changed. We'll discuss here what each of the options means. We'll use the rest of the chapter to show you how to use them.

```
% nslookup
Default Server:  bladerunner.fx.movie.edu
Address:  0.0.0.0

> set all
Default Server:  bladerunner.fx.movie.edu
Address:  0.0.0.0

Set options:
  nodebug          defname          search           recurse
  nod2             novc             noignoretc      port=53
  querytype=A     class=IN        timeout=5       retry=4
  root=a.root-servers.net.
  domain=fx.movie.edu
  srchlist=fx.movie.edu

> ^D
```

Before we get into the options, we need to cover the introductory lines. The default name server is *bladerunner.fx.movie.edu*. This means that every query sent by **nslookup** is going to be sent to *bladerunner*. The address 0.0.0.0 means "this host." When **nslookup** is using address 0.0.0.0 or 127.0.0.1 as its server, it is using the name server running on the local system – in this case, *bladerunner*.

The options come in two flavors: *Boolean* and *value*. The options that do not have an equals sign after them are Boolean options. They have the interesting property of being either "on" or "off." The value options can take on different, well, values. How can we tell which Boolean options are on and which are off? The option is *off* when a "no" precedes the option's name. *nodebug* means that debugging is off. As you might guess, the option *search* is on.

How you change Boolean or value options depends on whether you are using **nslookup** interactively or not. In an interactive session, you change an option with the **set** command, as in **set debug** or **set domain=classics.movie.edu**. From the command line, you omit the word **set** and precede the option with a hyphen, as in **nslookup -debug** or **nslookup -domain=classics.movie.edu**. The options can be abbreviated to their shortest unique string – e.g., *nodeb* for *nodebug*. In addition to its abbreviation, the *querytype* option can also be called simply *type*.

Let's go through each of the options:

## DNS & BIND

### *[no]debug*

Debugging is turned off by default. If it is turned on, the name server shows timeouts and displays the response packets. See *[no]d2* for a discussion of debug level 2.

### *[no]defname*

By default, **nslookup** adds the default domain name to names without a dot in them. Before search lists existed, the BIND resolver code would only add the default domain to names without *any* dots in them; this option reflects that behavior. **nslookup** can implement the pre-search list behavior (with *search* off and *defname* on), or it can implement the search list behavior (with *search* on).

### *[no]search*

The search option "overshadows" the default domain name (*defname*) option. That is, *defname* only applies if *search* is turned off. By default, **nslookup** appends the domains in the search list (*srchlist*) to names that don't end in a dot.

### *[no]recurse*

**nslookup** requests recursive service by default. This turns on the recursion-desired bit in query packets. The BIND resolver sends recursive queries in the same way. Name servers, however, send out nonrecursive queries to other name servers.

### *[no]d2*

Debugging at level 2 is turned off by default. If it is turned on, you see the query packets sent out in addition to the regular debugging output. Turning on *d2* also turns on *debug*. Turning off *d2* turns off *d2* only; *debug* is left on. Turning off *debug* turns off both *debug* and *d2*.

### *[no]vc*

By default, **nslookup** makes queries using UDP packets instead of over a virtual circuit (TCP). Most BIND resolver queries are made with UDP, so the default **nslookup** behavior matches the resolver. As the resolver can be instructed to use TCP, so can **nslookup**.

### *[no]ignoretc*

By default, **nslookup** *doesn't* ignore truncated packets. If a packet is received that has the "truncated" bit set – indicating that the name server couldn't fit all the important information in the UDP response packet – **nslookup** doesn't ignore it; it retries the query using a TCP connection instead of UDP. Again, this matches the BIND resolver behavior. The reason for retrying the query using a TCP connection is that TCP responses can be twice as large as UDP responses. TCP responses *could* be many times the size of a UDP response (a TCP connection can carry much more data than a single UDP packet), but the buffers BIND uses for a TCP query are only twice as large as the UDP buffers.

### *port=53*

The DNS service is on port 53. You can start a name server on another port – for debugging purposes, for example – and **nslookup** can be directed to use that port.

*querytype=A*

By default, **nslookup** looks up A (address) resource record types. In addition, if you type in an IP address (and the **nslookup** query type is address or pointer), then **nslookup** will invert the address, append *in-addr.arpa*, and look up PTR (pointer) data instead.

*class=IN*

The only class that matters is *Internet*. Well, there is the *Hesiod* (HS) class too, if you are an MITer or run Ultrix.

*timeout=5*

If the name server doesn't respond within 5 seconds, **nslookup** resends the query and doubles the timeout (to 10, 20, and then 40 seconds). The BIND resolver uses the same timeouts when querying a single name server.

*retry=4*

Send the query four times before giving up. After each retry, the timeout value is doubled. Again, this matches the BIND resolver behavior.

*root=a.root-servers.net.*

There is a convenience command called **root**, which switches your default server to the server named here. Executing the **root** command from a modern **nslookup**'s prompt is equivalent to executing **server a.root-servers.net**. Older versions use *nic.ddn.mil* (old) or even *sri-nic.arpa* (ancient) as the default root name server. You can change the default "root" server with **set root=server**.

*domain=fx.movie.edu*

This is the default domain appended if the *defname* option is on.

*srchlist=fx.movie.edu*

If *search* is on, these are the domains appended to names that do not end in a dot. The domains are listed in the order that they are tried, separated by a slash. (The 4.8.3 search list defaulted to *fx.movie.edu/movie.edu*. At 4.9.3, you have to explicitly set the search list in */etc/resolv.conf* to get both *fx.movie.edu* and *movie.edu*.)

### 11.3.1 The .nslookuprc File

You can set up new default **nslookup** options in an *.nslookuprc* file. **nslookup** will look for an *.nslookuprc* file in your home directory when it starts up, in both interactive and noninteractive modes. The *.nslookuprc* file can contain any legal **set** commands, one per line. This is useful, for example, if your old **nslookup** still thinks *sri-nic.arpa* is a root name server. You can set the default root name server to a real root with a line like this in your *.nslookuprc* file:

```
set root=a.root-servers.net.
```

## DNS & BIND

You might also use *.nslookuprc* to set your search list to something other than your host's default search list, or to change the timeouts **nslookup** uses.

---

11.2 Interactive Versus  
Noninteractive

11.4 Avoiding the Search  
List

---

[ [Library Home](#) | [DNS & BIND](#) | [TCP/IP](#) | [sendmail](#) | [sendmail Reference](#) | [Firewalls](#) | [Practical Security](#) ]



---

## Chapter 11 nslookup

---

### 11.4 Avoiding the Search List

**nslookup** implements the search list, as the resolver does. When you are debugging, the search list can get in your way. Either you need to turn the search list off completely (**set nosearch**), or you need to add a trailing dot to the fully qualified domain name you are looking up. We prefer the latter, as you'll see in our examples.

---

[11.3 Option Settings](#)

[11.5 Common Tasks](#)

---

[ [Library Home](#) | [DNS & BIND](#) | [TCP/IP](#) | [sendmail](#) | [sendmail Reference](#) | [Firewalls](#) | [Practical Security](#) ]



## Chapter 11 nslookup

### 11.5 Common Tasks

There are little chores you'll come to use **nslookup** for almost every day: finding the IP address or MX records for a given domain name, or querying a particular name server for data. We'll cover these first, before moving on to the more occasional stuff.

#### 11.5.1 Looking Up Different Data Types

By default, **nslookup** looks up the address for a name, or the name for an address. You can look up any data type by changing the *querytype*, as we will show in this example:

```
% nslookup
Default Server:  terminator.movie.edu
Address:  0.0.0.0

> misery                - Look up address
Server:  terminator.movie.edu
Address:  0.0.0.0

Name:    misery.movie.edu
Address: 192.253.253.2

> 192.253.253.2        - Look up name
Server:  terminator.movie.edu
Address: 0.0.0.0

Name:    misery.movie.edu
Address: 192.253.253.2

> set q=mx             - Look up MX data
> wormhole
Server:  terminator.movie.edu
Address: 0.0.0.0

wormhole.movie.edu    preference = 10, mail exchanger = wormhole.movie.edu
wormhole.movie.edu    internet address = 192.249.249.1
wormhole.movie.edu    internet address = 192.253.253.1

> set q=any           - Look up data of any type
> diehard
Server:  terminator.movie.edu
Address: 0.0.0.0

diehard.movie.edu     internet address = 192.249.249.4
diehard.movie.edu     preference = 10, mail exchanger = diehard.movie.edu
diehard.movie.edu     internet address = 192.249.249.4
```

These are only a few of the valid DNS data types, of course. For the complete list, see [Appendix A. DNS Message Format and Resource Records](#).

## 11.5.2 Authoritative Versus Nonauthoritative Answers

If you've used **nslookup** before, you might have noticed something peculiar – the first time you look up a remote name, the answer is authoritative, but the second time you look up the same name it is nonauthoritative. Here's an example:

```
% nslookup
Default Server:  relay.hp.com
Address:  15.255.152.2

> slate.mines.colorado.edu.
Server:  relay.hp.com
Address:  15.255.152.2

Name:    slate.mines.colorado.edu
Address:  138.67.1.3

> slate.mines.colorado.edu.
Server:  relay.hp.com
Address:  15.255.152.2

Non-authoritative answer:
Name:    slate.mines.colorado.edu
Address:  138.67.1.3
```

While this looks odd, it really isn't. What is happening is that the first time the local name server looks up *slate*, it contacts the name server for *mines.colorado.edu*, and the *mines.colorado.edu* server responds with an authoritative answer. The local name server, in effect, passes the authoritative response directly back to **nslookup**. It also caches the response. The second time you look up *slate*, the name server answers out of its cache, which results in the answer "nonauthoritative."

Notice that we ended the domain name with a trailing dot each time we looked it up. The response would have been the same had we left the trailing dot off. There are times when it is critical that you use the trailing dot while debugging, and times when it is not. Rather than stopping to decide if *this* name needs a trailing dot, we always add one if we know the name is fully qualified, except, of course, for the example where we turn off the search list.

## 11.5.3 Switching Servers

Sometimes you want to query another name server directly – you may think it is misbehaving, for example. You can switch servers with **nslookup** by using the **server** or **lserver** command. The difference between **server** and **lserver** is that **lserver** queries your "local" server – the one you started out with – to get the address of the server you want to switch to; **server** uses the default server instead of the local server. This difference is important to know because the server you just switched to may not be responding, as we'll show in this example:

```
% nslookup
Default Server:  relay.hp.com
Address:  15.255.152.2
```

When we start up, our first server, *relay.hp.com*, becomes our **lserver**. This will matter later on in this session.

## DNS & BIND

```
> server galt.cs.purdue.edu.
Default Server:  galt.cs.purdue.edu
Address:  128.10.2.39

> cs.purdue.edu.
Server:  galt.cs.purdue.edu
Address:  128.10.2.39

*** galt.cs.purdue.edu can't find cs.purdue.edu.: No response from server
```

At this point we try to switch back to our original name server. But there is no name server running on *galt* to look up *relay*'s address.

```
> server relay.hp.com.

*** Can't find address for server relay.hp.com.: No response from server
```

Instead of being stuck, though, we use the **lserver** command to have our local server look up *relay*'s address:

```
> lserver relay.hp.com.
Default Server:  relay.hp.com
Address:  15.255.152.2

> ^D
```

Since the server on *galt* did not respond – it's not even running a name server – it wasn't possible to look up the address of *relay* to switch back to using *relay*'s name server. Here's where **lserver** comes to the rescue: the local name server, *relay*, was still responding, so we used it. Instead of using **lserver**, we could have recovered by using *relay*'s IP address directly – **server 15.255.152.2**.

You can even change servers on a per–query basis. To specify that you'd like **nslookup** to query a particular server for information about a given domain name, you can specify the server as the second argument on the line, after the domain name to look up – like so:

```
% nslookup
Default Server:  relay.hp.com
Address:  15.255.152.2

> saturn.sun.com. ns.sun.com.
Name Server:  ns.sun.com
Address:  192.9.9.3

Name:  saturn.sun.com
Addresses:  192.9.25.2

> ^D
```

And, of course, you can change servers from the command line. You can specify the server to query as the argument after the domain name to look up, like this:

```
% nslookup -type=mx fisherking.movie.edu. terminator.movie.edu.
```

This instructs **nslookup** to query *terminator.movie.edu* for MX records for *fisherking.movie.edu*.

Finally, to specify an alternate default server and enter interactive mode, you can use a hyphen in place of the domain name to look up:

## DNS & BIND

```
% nslookup - terminator.movie.edu.
```

---

11.4 Avoiding the Search  
List

11.6 Less Common Tasks

---

[ [Library Home](#) | [DNS & BIND](#) | [TCP/IP](#) | [sendmail](#) | [sendmail Reference](#) | [Firewalls](#) | [Practical Security](#) ]




---

## Chapter 11 nslookup

---

### 11.6 Less Common Tasks

These are tricks you'll probably have to use less often, but which are very handy to have in your repertoire. Most of these will be helpful when you're trying to troubleshoot a DNS or BIND problem; they'll enable you to grub around in the packets the resolver sees, and mimic a BIND name server querying another name server or transferring zone data.

#### 11.6.1 Seeing the Query and Response Packets

If you need to, you can direct **nslookup** to show you the queries it sends out and the responses it receives. Turning on *debug* shows you the responses. Turning on *d2* shows you the queries as well. When you want to turn off debugging completely, you have to use **set nodebug**, since **set nod2** only turns off level 2 debugging. After the following trace, we'll explain some parts of the packet output. If you want, you can pull out your copy of RFC 1035, turn to page 25, and read along with our explanation.

```
% nslookup
Default Server: terminator.movie.edu
Address: 0.0.0.0

> set debug
> wormhole
Server: terminator.movie.edu
Address: 0.0.0.0

-----
Got answer:
HEADER:
    opcode = QUERY, id = 6813, rcode = NOERROR
    header flags: response, auth. answer, want recursion,
    recursion avail. questions = 1, answers = 2,
    authority records = 2, additional = 3

QUESTIONS:
    wormhole.movie.edu, type = A, class = IN
ANSWERS:
-> wormhole.movie.edu
    internet address = 192.253.253.1
    ttl = 86400 (1D)
-> wormhole.movie.edu
    internet address = 192.249.249.1
    ttl = 86400 (1D)
AUTHORITY RECORDS:
-> movie.edu
    nameserver = terminator.movie.edu
    ttl = 86400 (1D)
```

## DNS & BIND

```
-> movie.edu
    nameserver = wormhole.movie.edu
    ttl = 86400 (1D)
ADDITIONAL RECORDS:
-> terminator.movie.edu
    internet address = 192.249.249.3
    ttl = 86400 (1D)
-> wormhole.movie.edu
    internet address = 192.253.253.1
    ttl = 86400 (1D)
-> wormhole.movie.edu
    internet address = 192.249.249.1
    ttl = 86400 (1D)

-----
Name:      wormhole.movie.edu
Addresses: 192.253.253.1, 192.249.249.1

> set d2
> wormhole
Server:    terminator.movie.edu
Address:   0.0.0.0

This time the query is also shown.

-----
SendRequest(), len 36
  HEADER:
    opcode = QUERY, id = 6814, rcode = NOERROR
    header flags: query, want recursion
    questions = 1, answers = 0, authority records = 0,
    additional = 0

  QUESTIONS:
    wormhole.movie.edu, type = A, class = IN

-----
-----
Got answer (164 bytes):

The answer is the same as above.
```

The text between the dashes is the query and response packets. As promised, we will go through the packet contents. DNS packets are composed of five sections:

1. Header section
2. Question section
3. Answer section
4. Authority section
- 5.

## DNS & BIND

### Additional section

#### *Header section*

The header section is present in every query and response. The operation code is always QUERY. The only other opcodes are inverse query (IQUERY) and status (STATUS), but those aren't used. The id is used to associate a response with a query and to detect duplicate queries or responses. You have to look in the header flags to see which packets are queries and which are responses. The string `want recursion` means that the querier wants the name server to do all the work. The flag is parroted in the response. The string `auth. answer` means that this response is *authoritative*. In other words, the response is from the name server's authoritative data, not from its cache data. The response code, `rcode`, can be one of no error, server failure, name error (also known as `nxdomain` or `nonexistent domain`), not implemented, or refused. The server failure, name error, not implemented, and refused response codes cause the **nslookup** "Server failed," "Nonexistent domain," "Not implemented," and "Query refused" errors, respectively. The last four entries in the header section are counters – they indicate how many resource records there are in each of the next four sections.

#### *Question section*

There is always *one* question in a DNS packet; it includes the name and the requested data type and class. There is never more than one question in a DNS packet. The capability of handling more than one question in a DNS packet would require a redesign of the packet format. For one thing, the single authority bit would have to be changed, because the answer section could contain a mix of authoritative answers and nonauthoritative answers. In the present design, setting the authoritative answer bit means that the name server is an authority for the domain name in the question section.

#### *Answer section*

This section contains the resource records that answer the question. There can be more than one resource record in the response. For example, if the host is multihomed, there will be more than one address resource record.

#### *Authority section*

The authority section is where name server records are returned. When a response refers the querier to some other name servers, those name servers are listed here.

#### *Additional section*

The additional records section adds information that may complete information included in other sections. For instance, if a name server is listed in the authority section, the name server's address is added to the additional records section. After all, to contact the name server, you need to have its address.

For you sticklers for detail, there *is* a time when the number of questions in a query packet isn't one: in an inverse query, when it's zero. In an inverse query, there is one answer in the query packet, and the question section is empty. The name server fills in the question. But, as we said, inverse queries are almost nonexistent.

## 11.6.2 Querying Like a BIND Name Server

You can make **nslookup** send out the same query packet a name server would. Name server query packets are not much different from resolver packets. The primary difference in the query packets is that resolvers request recursive services, and name servers seldom do. Recursion is the default with **nslookup**, so you have to explicitly turn it off. The difference in *operation* between a resolver and a name server is that the resolver implements the search list, and the name server doesn't. By default, **nslookup** implements the search list, so that, too, has to be turned off. Of course, judicious use of the trailing dot will have the same effect.

In raw **nslookup** terms, this means that to query like a resolver, you use **nslookup**'s default settings. To query like a name server, use **set norecurse** and **set nosearch**. On the command line, that's **nslookup -norecurse -nosearch**.

When a BIND name server gets a query, it looks for the answer in its cache. If it doesn't have the answer, and it is authoritative for the domain, the name server responds that the name doesn't exist or that there is no data for that type. If the name server doesn't have the answer, and it is *not* authoritative for the domain, it starts walking up the domain tree looking for NS records. There will always be NS records somewhere higher in the domain tree. As a last resort, it will use the NS records at the root domain, the highest level.

If the name server received a nonrecursive query, it would respond to the querier by giving the NS records that it had found. On the other hand, if the original query was a recursive query, the name server would then query the remote name servers in the NS records that it found. When the name server receives a response from one of the remote name servers, it caches the response, and repeats this process, if necessary. The remote server's response will either have the answer to the question or it will contain a list of name servers lower in the domain tree and closer to the answer.

Let's assume for our example that we are trying to satisfy a recursive query and that we didn't find any NS records until we checked the *gov* domain. That is, in fact, the case when we ask the name server on *relay.hp.com* about *www.whitehouse.gov* – it doesn't find any NS records until the *gov* domain. From there we switch servers to a *gov* name server and ask the same question. It directs us to the *whitehouse.gov* servers. We then switch to a *whitehouse.gov* name server and ask the same question:

```
% nslookup
Default Server: relay.hp.com
Address: 15.255.152.2

> set norec           - Query like a name server: turn off recursion
> set nosearch        - turn off the search list
> www.whitehouse.gov - We don't need to dot-terminate since we've turned
                    - search off

Server: relay.hp.com
Address: 15.255.152.2

Name: www.whitehouse.gov
Served by:
- H.ROOT-SERVERS.NET
  128.63.2.53
  gov
- B.ROOT-SERVERS.NET
  128.9.0.107
  gov
- C.ROOT-SERVERS.NET
  192.33.4.12
```

## DNS & BIND

```
gov
- D.ROOT-SERVERS.NET
  128.8.10.90
gov
- E.ROOT-SERVERS.NET
  192.203.230.10
gov
- I.ROOT-SERVERS.NET
  192.36.148.17
gov
- F.ROOT-SERVERS.NET
  192.5.5.241
gov
- G.ROOT-SERVERS.NET
  192.112.36.4
gov
- A.ROOT-SERVERS.NET
  198.41.0.4
gov
```

Switch to a gov name server. You may have to turn recursion back on temporarily, if the name server doesn't have the address already cached.

```
> server e.root-servers.net
Default Server: e.root-servers.net
Address: 192.203.230.10
```

Ask the same question of the gov name server. It will refer us to name servers closer to our desired answer.

```
> www.whitehouse.gov.
Server: e.root-servers.net
Address: 192.203.230.10

Name: www.whitehouse.gov
Served by:
- SEC1.DNS.PSI.NET
  38.8.92.2
  WHITEHOUSE.GOV
- SEC2.DNS.PSI.NET
  38.8.93.2
  WHITEHOUSE.GOV
```

Switch to a whitehouse.gov name server – either of them will do.

```
> server sec1.dns.psi.net.
Default Server: sec1.dns.psi.net
Address: 38.8.92.2

> www.whitehouse.gov.
Server: sec1.dns.psi.net
Address: 38.8.92.2

Name: www.whitehouse.gov
Addresses: 198.137.240.91, 198.137.240.92
```

We hope this example gives you a feeling for how name servers look up names. If you need to refresh your understanding of what this looks like graphically, flip back to [Figure 2.10](#).

## DNS & BIND

Before we move on, notice that we asked each of the servers the very same question: "What's the address for *www.whitehouse.gov*?" What do you think would happen if the *gov* name server had already cached *www.whitehouse.gov*'s address itself? The *gov* name server would have answered the question out of its cache instead of referring you to the *whitehouse.gov* name servers. Why is this significant? Suppose you messed up a particular host's address in your zone. Someone points it out to you, and you clean up the problem. Even though your name server now has the correct data, some remote sites find the old, messed-up data when they look up the name. One of the name servers higher up in the domain tree, such as a root name server, has cached the incorrect data; when it receives a query for that host's address, it returns the incorrect data instead of referring the querier to your name servers. What makes this problem hard to track down is that only one of the "higher up" name servers has cached the incorrect data, so only some of the remote lookups get the wrong answer – the ones that use this server. Fun, huh? Eventually, though, the "higher up" name server will time out the old record. If you're pressed for time, you can contact the administrators of the remote name server and ask them to kill and restart **named** to flush the cache. Of course, if the remote name server is an important, much-used name server, they may tell you where to go with that suggestion.

### 11.6.3 Zone Transfers

**nslookup** can be used to transfer a whole zone using the **ls** command. This feature is useful for troubleshooting, for figuring out how to spell a remote host's name, or just for counting how many hosts are in some remote domain. Since the output can be substantial, **nslookup** allows you to redirect the output to a file. If you want to bail out in the middle of a transfer, you can interrupt it by typing your interrupt character.

Beware: some hosts won't let you pull a copy of their zone, either for security reasons or to limit the load on their name server host. The Internet is a friendly place, but administrators have to defend their turf.

Let's look at the *movie.edu* zone. As you will see in the output below, all the zone data is listed – the SOA record is listed twice, which is an artifact of how the data is exchanged during the zone transfer. (BIND 4's **nslookup** only shows you address and name server data, not all the data.)

```
% nslookup
Default Server: terminator.movie.edu
Address: 0.0.0.0

> ls movie.edu.
@                4D IN SOA      terminator root.terminator (
                  1997080605      ; serial
                  3H          ; refresh
                  1H          ; retry
                  4w2d       ; expiry
                  1D )       ; minimum

terminator       4D IN NS      terminator
terminator       4D IN A       192.249.249.3
terminator       4D IN MX      10 terminator
terminator       4D IN NS      wormhole
wormhole         4D IN A       192.249.249.1
wormhole         4D IN A       192.253.253.1
wormhole         4D IN MX      10 wormhole
robocop          4D IN A       192.249.249.2
robocop          4D IN MX      10 robocop
wh249            4D IN A       192.249.249.1
wh253            4D IN A       192.253.253.1
wh                4D IN CNAME   wormhole
shining          4D IN A       192.253.253.3
```

## DNS & BIND

```
localhost      4D IN MX      10 shining
localhost      4D IN A       127.0.0.1
bitg           4D IN CNAME   terminator
carrie         4D IN A       192.253.253.4
carrie         4D IN MX      10 carrie
dh             4D IN CNAME   diehard
diehard        4D IN A       192.249.249.4
diehard        4D IN MX      10 diehard
misery         4D IN A       192.253.253.2
misery         4D IN MX      10 misery
@              4D IN SOA     terminator root.terminator (
                1997080605    ; serial
                3H      ; refresh
                1H      ; retry
                4w2d   ; expiry
                1D )   ; minimum
```

Received 48 answers (0 records).

> **ls movie.edu > /tmp/movie** - List all data into /tmp/movie

[terminator.movie.edu]

Received 48 answers (0 records).

---

11.5 Common Tasks

11.7 Troubleshooting  
nslookup Problems

---

[ [Library Home](#) | [DNS & BIND](#) | [TCP/IP](#) | [sendmail](#) | [sendmail Reference](#) | [Firewalls](#) | [Practical Security](#) ]



## Chapter 11 nslookup

### 11.7 Troubleshooting nslookup Problems

The last thing you want is to have problems with your troubleshooting tool. Unfortunately, some types of failures render the troubleshooting tool mostly useless. Other types of **nslookup** failures are, at best, confusing because they don't give you any direct information to work with. While there may be a few problems with **nslookup** itself, most of the problems you encounter will be with name server configuration and operation. We'll cover a few odd problems here.

#### 11.7.1 Looking Up the Right Data

This isn't really a problem, *per se*, but it can be awfully confusing. If you use **nslookup** to look up a type of data for a domain name, and the domain name exists, but no data of the type you're looking for exists, you'll get an error like this:

```
% nslookup
Default Server:  terminator.movie.edu
Address:  0.0.0.0

> movie.edu.

*** No address (A) records available for movie.edu.
```

So what types of records *do* exist? Just **set type=any** to find out:

```
> set type=any
> movie.edu.
Server:  terminator.movie.edu
Address:  0.0.0.0

movie.edu
  origin = terminator.movie.edu
  mail addr = al.robocop.movie.edu
  serial = 42
  refresh = 10800 (3H)
  retry = 3600 (1H)
  expire = 604800 (7D)
  minimum ttl = 86400 (1D)
movie.edu  nameserver = terminator.movie.edu
movie.edu  nameserver = wormhole.movie.edu
movie.edu  nameserver = zardozi.movie.edu
movie.edu  preference = 10, mail exchanger = postmanrings2x.movie.edu
postmanrings2x.movie.edu  internet address = 192.249.249.66
```

## 11.7.2 No Response from Server

What could have gone wrong if your server can't look up its own name?

```
% nslookup
Default Server: terminator.movie.edu
Address: 0.0.0.0

> terminator
Server: terminator.movie.edu
Address: 0.0.0.0

*** terminator.movie.edu can't find terminator: No response from server
```

The "no response from server" error message means exactly that: the name server didn't get back a response. **nslookup** doesn't necessarily look up anything when it starts up. If you see that the address of your server is 0.0.0.0, **nslookup** grabbed the system's host name (what the **hostname** command returns) for the server field and gave you its prompt. It is only when you try to look up something that you find out that there is no server responding. In this case, it is pretty obvious that there is no name server running – a name server ought to be able to look up its own name. If you are looking up some remote information, though, the name server could fail to respond because it is still trying to look up the item and **nslookup** gave up waiting. How can you tell the difference between a name server that isn't running and a name server that is running but didn't respond? Use the **ls** command to point out the difference:

```
% nslookup
Default Server: terminator.movie.edu
Address: 0.0.0.0

> ls foo.      - Try to list a nonexistent domain
*** Can't list domain foo.: No response from server
```

In this case, no name server is running. If the host couldn't be reached, the error would be "timed out." If a name server is running, you'll see the following error message:

```
% nslookup
Default Server: terminator.movie.edu
Address: 0.0.0.0

> ls foo.
[terminator.movie.edu]
*** Can't list domain foo.: No information
```

That is, unless there's a top-level *foo* domain in your world.

## 11.7.3 No PTR Data for Name Server's Address

Here is one of the most annoying problems: something went wrong, and **nslookup** exited on startup:

```
% nslookup

*** Can't find server name for address 192.249.249.3: Non-existent host/domain
*** Default servers are not available
```

The "nonexistent domain" means that the name *3.249.249.192.in-addr.arpa* doesn't exist. In other words, **nslookup** couldn't find the name for 192.249.249.3, its name server host. But didn't we just say that

**nslookup** doesn't look up anything when it starts up? In the configuration presented before, **nslookup** didn't look up anything, but that's not a rule. If you create a *resolv.conf* that includes *nameserver* lines, **nslookup** looks up the address in order to get the name server's name. In the preceding example, there is a name server running on 192.249.249.3, but it said there is no PTR data for the address 192.249.249.3. Obviously, this name server's data is messed up, at least for the *249.249.192.in-addr.arpa* zone.

The "default servers are not available" message in the example is misleading. After all, there is a name server there to say the address doesn't exist. More often, you'll see the error "no response from server" if the name server isn't running on the host or the host can't be reached. Only then does the "default servers are not available" message makes sense.

### 11.7.4 Query Refused

Refused queries can cause problems at startup, and they can cause lookup failures during a session. Here's what it looks like when **nslookup** exits on startup because of a refused query:

```
% nslookup
*** Can't find server name for address 192.249.249.3: Query refused
*** Default servers are not available
%
```

This one has two possible causes. Either your name server does not support inverse queries (older **nslookup**s only), or zone security is stopping the lookup.

Old versions of **nslookup** (pre-4.8.3) used an inverse query on startup. Inverse queries were never widely used – **nslookup** was one of the few applications that did use them. At 4.9.3, support for inverse queries was dropped, which broke old **nslookup**s. To accommodate these old clients, a configuration file statement was added.

In BIND 4, the statement looks like this:

```
options fake-iquery
```

In BIND 8, the statement looks like this:

```
options { fake-iquery yes; };
```

This statement causes your name server to respond to the inverse query with a "fake" response that is good enough for **nslookup** to continue.

Zone security features can also cause **nslookup** startup problems. When **nslookup** attempts to find the name of its server (using a PTR query, not an inverse query), the query can be refused. If you think the problem is zone security, make sure your BIND 4 *secure\_zone* TXT resource records or BIND 8 *allow-transfer* substatement include the network for the host running **nslookup**, and the address 127.0.0.1 if **nslookup** is running on the host also running the name server.

Zone security is not limited to causing **nslookup** to fail to start up. It can also cause lookups and zone transfers to fail in the middle of a session when you point **nslookup** to a remote name server. This is what you will see:

```
% nslookup
```

## DNS & BIND

```
Default Server: hp.com
Address: 15.255.152.4

> server terminator.movie.edu
Default Server: terminator.movie.edu
Address: 192.249.249.3

> carrie.movie.edu.
Server: terminator.movie.edu
Address: 192.249.249.3

*** terminator.movie.edu can't find carrie.movie.edu.: Query refused

> ls movie.edu - This attempts a zone transfer
[terminator.movie.edu]
*** Can't list domain movie.edu: Query refused
>
```

### 11.7.5 First resolv.conf Name Server Not Responding

Here is another twist on the last problem:

```
% nslookup

*** Can't find server name for address 192.249.249.3: No response from server
Default Server: wormhole.movie.edu
Address: 192.249.249.1
```

This time the first *nameserver* in *resolv.conf* did not respond. We had put a second *nameserver* line in *resolv.conf*, and the second server did respond. From now on, **nslookup** will send queries only to *wormhole*; it won't try the name server at 192.249.249.3 again.

### 11.7.6 Finding Out What Is Being Looked Up

We've been waving our hands in the last examples, saying that **nslookup** was looking up the name server's address, but we didn't prove it. Here is our proof. This time, when we started up **nslookup** we turned on *d2* debugging from the command line. This causes **nslookup** to print out the query packets it sent, as well as printing out when the query timed out and was retransmitted:

```
% nslookup -d2
-----
SendRequest(), len 44
  HEADER:
    opcode = QUERY, id = 1, rcode = NOERROR
    header flags: query, want recursion
    questions = 1, answers = 0, authority records = 0,
    additional = 0

    QUESTIONS:
      3.249.249.192.in-addr.arpa, type = PTR, class = IN

-----
timeout (5 secs)
timeout (10 secs)
timeout (20 secs)
timeout (40 secs)
SendRequest failed
```

```
*** Can't find server name for address 192.249.249.3: No response from server
*** Default servers are not available
```

As you can see by the timeouts, it took 75 seconds for **nslookup** to give up. Without the debugging output, you won't see anything printed to your screen for 75 seconds; it'll look as if **nslookup** has hung.

### 11.7.7 Unspecified Error

You can run into a rather unsettling problem called "unspecified error." We have an example of this error here. We've only included the tail end of the output, since we only want to talk about the error at this point. You'll find the whole **nslookup** session that produced this segment in [Chapter 13, Troubleshooting DNS and BIND](#).

```
Authoritative answers can be found from:
(root) nameserver = NS.NIC.DDN.MIL
(root) nameserver = B.ROOT-SERVERS.NET
(root) nameserver = E.ROOT-SERVERS.NET
(root) nameserver = D.ROOT-SERVERS.NET
(root) nameserver = F.ROOT-SERVERS.NET
(root) nameserver = C.ROOT-SERVERS.NET
(root) nameserver =
*** Error: record size incorrect (1050690 != 65519)

*** relay.hp.com can't find .: Unspecified error
```

What happened here is that there was too much data to fit into a UDP datagram. The name server stopped filling in the response when it ran out of room. The name server *didn't* set the truncation bit in the response packet, or **nslookup** would have retried the query over a TCP connection; the name server must have decided that enough of the "important" information fit. You won't see this kind of error very often. You'll see it if you create too many NS records for a domain, so don't create too many. (Advice like this makes you wonder why you bought this book, right?) How many is too many depends upon how well the names can be "compressed" in the packet, which, in turn, depends upon how many name servers share the same domain in their domain name. The root name servers were renamed to all be in the *root-servers.net* domain for this very reason – more names fit in DNS packets if they share a common domain, which allows more root name servers to support the Internet. As a rule of thumb, don't go over 10 NS records. As for what caused *this* error, you'll have to read [Chapter 13](#). Those of you who just read [Chapter 9, Parenting](#), may know already.

---

11.6 Less Common Tasks

11.8 Best of the Net

---

[ [Library Home](#) | [DNS & BIND](#) | [TCP/IP](#) | [sendmail](#) | [sendmail Reference](#) | [Firewalls](#) | [Practical Security](#) ]




---

## Chapter 11 nslookup

---

### 11.8 Best of the Net

System administrators have a thankless job. There are certain questions, usually quite simple ones, that they are asked over and over again. And sometimes, in a creative mood, they come up with a clever way to help their users. When the rest of us find out about their ingenuity, we can only sit back, smile admiringly, and wish we had thought of it ourselves. Here is one such case, where a system administrator found a way to communicate the solution to the sometimes perplexing puzzle of how to end an **nslookup** session:

```
% nslookup
Default Server:  envy.ugcs.caltech.edu
Address:  131.215.134.135

> quit
Server:  envy.ugcs.caltech.edu
Addresses:  131.215.134.135, 131.215.128.135

Name:  ugcs.caltech.edu
Addresses:  131.215.128.135, 131.215.134.135
Aliases:  quit.ugcs.caltech.edu
          use.exit.to.leave.nslookup.-.-.ugcs.caltech.edu

> exit
%
```

---

11.7 Troubleshooting  
nslookup Problems

12. Reading BIND  
Debugging Output

---

[ [Library Home](#) | [DNS & BIND](#) | [TCP/IP](#) | [sendmail](#) | [sendmail Reference](#) | [Firewalls](#) | [Practical Security](#) ]