DiG HOWTO

Paul Heinlein <*heinlein@madboa.com*>
Initial publication: August 31, 2004
Most recent revision: May 11, 2006

How to use **dig** to query DNS name servers.

Table of Contents

Introduction
Understanding the default output
What can I discover?
How do I ...

- Get a short answer?
- Get a not-quite-so-short answer?
- Get a long answer?
- Do a reverse lookup?
- Query a different nameserver?
- Use the `search` list in `/etc/resolv.conf`?
- Do bulk lookups?

Verifying DNS mappings
dig fun

- Roll your own `named.root` file
- Tracing dig's path
- Grabbing SOA information

Interpreting TTL numbers

Introduction

dig is a command-line tool for querying DNS name servers for information about host addresses, mail exchanges, name servers, and related information. The *dig(1)* man page is somewhat lacking when it comes to examples, a shortcoming this article tries to remedy.

The source code for **dig** is part of the larger ISC BIND distribution. Compiling and installing BIND are topics outside the scope of this document, but on Linux systems **dig** is usually part of a common package: bind-tools (Gentoo), bind-utils (Red Hat, Fedora), or dnsutils (Debian).

If you're looking for information on configuring the BIND name server, you might find my article *BIND for the Small LAN* more to your taste.

Understanding the default output

The most typical, simplest query is for a single host. By default, however, **dig** is pretty

verbose. You probably don't need all the information in the default output, but it's probably worth knowing what it is. Below is an annotated query.

```
$ dig www.isc.org
```

That's the command-line invocation of dig I used.

```
; <<>> DiG 9.2.3 <<>> www.isc.org
;; global options:  printcmd
```

The opening section of dig's output tells us a little about itself (version 9.2.3) and the global options that are set (in this case, `printcmd`). This part of the output can be quelled by using the `+nocmd` option, but only if it's the very first argument on the command line (even preceeding the host you're querying).

```
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 43071
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 3, ADDITIONAL: 3
```

Here, dig tells us some technical details about the answer received from the DNS server. This section of the output can be toggled using the `+[no]comments` option—but beware that disabling the comments also turns off many section headers.

```
;; QUESTION SECTION:
;www.isc.org.                   IN      A
```

In the question section, dig reminds us of our query. The default query is for an Internet address (`A`). You can turn this output on or off using the `+[no]question` option.

```
;; ANSWER SECTION:
www.isc.org.            600     IN      A       204.152.184.88
```

Finally, we get our answer: the address of `www.isc.org` is `204.152.184.88`. I don't know why you'd ever want to turn off the answer, but you can toggle this section of the output using the `+[no]answer` option.

```
;; AUTHORITY SECTION:
isc.org.                2351    IN      NS      ns-int.isc.org.
isc.org.                2351    IN      NS      ns1.gnac.com.
isc.org.                2351    IN      NS      ns-ext.isc.org.
```

The authority section tells us what DNS servers can provide an authoritative answer to our query. In this example, `isc.org` has three name servers. You can toggle this section of the output using the `+[no]authority` option.

```
;; ADDITIONAL SECTION:
ns1.gnac.com.           171551  IN      A       209.182.216.75
ns-int.isc.org.         2351    IN      A       204.152.184.65
ns-int.isc.org.         2351    IN      AAAA    2001:4f8:0:2::15
```

The additional section typically includes the IP addresses of the DNS servers listed in the authority section. This section of the output can be toggled with the `+[no]additional` option.

```
;; Query time: 2046 msec
;; SERVER: 127.0.0.1#53(127.0.0.1)
```

```
;; WHEN: Fri Aug 27 08:22:26 2004
;; MSG SIZE  rcvd: 173
```

The final section of the default output contains statistics about the query; it can be toggled with the `+[no]stats` option.

What can I discover?

dig will let you perform any valid DNS query, the most common of which are `A` (the IP address), `TXT` (text annotations), `MX` (mail exchanges), `NS` name servers, or the omnibus `ANY`.

```
# get the address(es) for yahoo.com
dig yahoo.com A +noall +answer

# get a list of yahoo's mail servers
dig yahoo.com MX +noall +answer

# get a list of DNS servers authoritative for yahoo.com
dig yahoo.com NS +noall +answer

# get all of the above
dig yahoo.com ANY +noall +answer
```

More obscurely, for the present anyway, you can also poll for a host's IPv6 address using the `AAAA` option.

```
dig www.isc.org AAAA +short
```

If the domain you want to query allows DNS transfers, you can get those, too. The reality of life on the Internet, however, is that very few domains allow unrestricted transfers these days.

```
dig yourdomain.com AXFR
```

How do I ...

Get a short answer?

When all you want is a quick answer, the `+short` option is your friend:

```
$ dig www.isc.org +short
204.152.184.88
```

Get a not-quite-so-short answer?

Note that a short answer is different from only an answer. The way to get a detailed answer, but without any auxiliary information, is to turn off all the results (`+noall`) and then turn on only those sections you want.

Here's a short answer followed by only an answer; the latter includes all the configuration information, including time-to-live (TTL) data, displayed in a format compatible with BIND configuration files.

```
$ dig fsf.org mx +short
20 mx20.gnu.org.
30 mx30.gnu.org.
10 mx10.gnu.org.

$ dig +nocmd fsf.org mx +noall +answer
fsf.org.                3583    IN      MX      30 mx30.gnu.org.
fsf.org.                3583    IN      MX      10 mx10.gnu.org.
fsf.org.                3583    IN      MX      20 mx20.gnu.org.
```

Get a long answer?

According to its man page, the `+multiline` option will give you an answer with "the SOA records in a verbose multi-line format with human-readable comments." In general, the answers retrieved using the `+multiline` option will appear more like BIND config files than they will without it.

```
$ dig +nocmd ogi.edu any +multiline +noall +answer
ogi.edu.   14267 IN A 129.95.59.31
ogi.edu.   14267 IN MX 5 cse.ogi.edu.
ogi.edu.   14267 IN MX 15 hermes.admin.ogi.edu.
ogi.edu.   14267 IN SOA zeal.admin.ogi.edu. hostmaster.admin.ogi.edu. (
                  200408230  ; serial
                  14400      ; refresh (4 hours)
                  900        ; retry (15 minutes)
                  3600000    ; expire (5 weeks 6 days 16 hours)
                  14400      ; minimum (4 hours)
                  )
ogi.edu.   14267 IN NS zeal.admin.ogi.edu.
ogi.edu.   14267 IN NS cse.ogi.edu.
ogi.edu.   14267 IN NS fork.admin.ogi.edu.
```

Do a reverse lookup?

Use the `-x` option to lookup the main hostname associated with an IP address.

```
$ dig -x 204.152.184.167 +short
mx-1.isc.org.
```

In a loop, this is a slick way to map the names in a given subnet:

```
#!/bin/bash
NET=18.7.22
for n in $(seq 1 254); do
  ADDR=${NET}.${n}
  echo -e "${ADDR}\t$(dig -x ${ADDR} +short)"
done
```

Query a different nameserver?

Just specify it on the command line:

```
dig @ns1.google.com www.google.com
```

Use the `search` list in `/etc/resolv.conf`?

The **host** utility will automatically use the `search` list in your `/etc/resolv.conf` file.

```
$ host www
www.madboa.com has address 65.102.49.170
```

By default, however, **dig** doesn't—which may produce some unexpected results. If you want to use local hostnames instead of fully qualified domain names, use the `+search` option.

```
dig www +search
```

Do bulk lookups?

If you want to look up a large number of hostnames, you can put them in a file (one name per line) and use the `-f` option to query each one in turn.

```
# do full lookups for a number of hostnames
dig -f /path/to/host-list.txt

# the same, with more focused output
dig -f /path/to/host-list.txt +noall +answer
```

As far as I can tell, **dig** versions up to and including 9.2.3 are unable to do reverse lookups using the `-f` option.

Verifying DNS mappings

An improperly configured DNS setup can be really annoying. You want to make sure that your mappings work both ways:

1. Each hostname should resolve to an address, and that address ought to resolve back to the proper hostname.
2. If an address on your subnet(s) has been assigned a reverse pointer to a hostname, that hostname ought to point back to the original address.

There are exceptions to those two rules, of course. A CNAME will resolve to another hostname first, and only then to an address. Sometimes multiple hostnames will point to the same address, but that address will have only one reverse pointer.

Still, it's good to know that your basic mappings work as expected.

You can script such a test if you build a file containing your known hostnames. The example script below is pretty simple; it will break if fed a CNAME, and it'll report a failure somewhere if multiple hostnames point to the same address. Let's assume the file containing your hostnames is named `named-hosts`.

```
#!/bin/bash
#
# test DNS forward- and reverse-mapping
#
```

```
# edit this variable to reflect local class C subnet(s)
NETS="192.168.1 192.168.2"

# Test name to address to name validity
echo
echo -e "\tname -> address -> name"
echo '---------------------------------'
while read H; do
  ADDR=$(dig $H +short)
  if test -n "$ADDR"; then
    HOST=$(dig -x $ADDR +short)
    if test "$H" = "$HOST"; then
      echo -e "ok\t$H -> $ADDR -> $HOST"
    elif test -n "$HOST"; then
      echo -e "fail\t$H -> $ADDR -> $HOST"
    else
      echo -e "fail\t$H -> $ADDR -> [unassigned]"
    fi
  else
    echo -e "fail\t$H -> [unassigned]"
  fi
done < named-hosts

# Test address to name to address validity
echo
echo -e "\taddress -> name -> address"
echo '-----------------------------------'
for NET in $NETS; do
  for n in $(seq 1 254); do
    A=${NET}.${n}
    HOST=$(dig -x $A +short)
    if test -n "$HOST"; then
      ADDR=$(dig $HOST +short)
      if test "$A" = "$ADDR"; then
        echo -e "ok\t$A -> $HOST -> $ADDR"
      elif test -n "$ADDR"; then
        echo -e "fail\t$A -> $HOST -> $ADDR"
      else
        echo -e "fail\t$A -> $HOST -> [unassigned]"
      fi
    fi
  done
done
```

dig fun

Roll your own `named.root` file

Any DNS server connected to the Internet is likely to have a copy of the InterNIC's `named.root` file that lists the root name servers for the entire Internet. You can always download that file in the boring way from the InterNIC's ftp server. Or, in a true build-it-yourself fashion, you can build it with **dig**.

```
# compare with ftp://ftp.internic.net/domain/named.root
dig +nocmd . NS +noall +answer +additional
```

Your TTL values might be a little on the short side, but otherwise, it's as up-to-date a list as you'll find!

Tracing dig's path

Perhaps you're a devotee of **traceroute** and like to watch how to get from point A to point B. You can do a similar thing with dig's `+trace` option.

```
dig gentoo.de +trace
```

You'll see dig head to the root name servers, then to the servers responsible for all the *.de domains, and finally to the name servers responsible for `gentoo.de`.

Grabbing SOA information

As a DNS administrator, I sometimes make changes and want to see if any of my name servers are still pushing the old data. The `+nssearch` provides a clear accounting of your public servers.

```
# the unvarnished truth
dig cse.ogi.edu +nssearch

# the same, displaying only serial number and hostname
dig cse.ogi.edu +nssearch | cut -d' ' -f4,11
```

Interpreting TTL numbers

I love Google for many reasons, one of which is that it provides referrer strings in my web logs that make it easy to figure out what sort of queries lead people to pages on this site.

Somewhat unexpectedly, I've seen a lot of queries asking for information about TTL (time-to-live) numbers. I would have never guessed that TTL would be a topic of interest, but you learn something new every day. So, by popular demand, here's a brief intro to TTL.

If you ask your local DNS server for an Internet address, the server figures out where to find an authoritative answer and then asks for it. Once the server receives an answer, it will keep the answer in a local cache so that if you ask for the same address again a short time later, it can give you the answer quickly rather than searching the Internet for it all over again.

When domain administrators configure their DNS records, they decide how long the records should remain in remote caches. This is the TTL number (usually expressed in number of seconds).

Typically, a remote server will only cache those records for the length of time specified by the TTL. After that, the remote server will flush its local cache and ask again for an authoritative answer.

When you use dig to query a DNS server concerning a certain record, the server will tell

dig the time (in seconds) that record will remain in cache.

For example, as of this writing, the TTL for the MX records for the `gmail.com` domain is 300 seconds. The `gmail.com` admins are asking that remote servers cache their MX records for no more than five minutes. So when you first ask for that record set, dig will report a TTL of 300.

```
$ dig +nocmd gmail.com MX +noall +answer
gmail.com.          300     IN      MX      20 gsmtp57.google.com.
gmail.com.          300     IN      MX      10 gsmtp171.google.com.
```

If you ask a few seconds later, you'll see the TTL number reduced by approximately the number of seconds you waited to ask again.

```
$ dig +nocmd gmail.com MX +noall +answer
gmail.com.          280     IN      MX      10 gsmtp171.google.com.
gmail.com.          280     IN      MX      20 gsmtp57.google.com.
```

If your timing is good, you can catch the record at the very end of its life.

```
$ dig +nocmd gmail.com MX +noall +answer
gmail.com.          1       IN      MX      10 gsmtp171.google.com.
gmail.com.          1       IN      MX      20 gsmtp57.google.com.
```

After that, the DNS server you're querying will "forget" the answer to that question, so the whole cycle will start over again (in this example, at 300 seconds) the next time you perform that query.

return to technical writings
home - tech - praise - paul - books - about
printer-friendly layout