Next Previous Contents

5. A *simple* domain.

How to set up your own domain.

5.1 But first some dry theory

First of all: you read all the stuff before here right? You have to.

Before we *really* start this section I'm going to serve you some theory on and an example of how DNS works. And you're going to read it because it's good for you. If you don't want to you should at least skim it very quickly. Stop skimming when you get to what should go in your `named.conf` file.

DNS is a hierarchical, tree structured system. The top is written `.' and pronounced `root', as is usual for tree data-structures. Under `.` there are a number of Top Level Domains (TLDs); the best known ones are `ORG`, `COM`, `EDU` and `NET`, but there are many more. Just like a tree it has a root and it branches out. If you have any computer science background you will recognize DNS as a search tree, and you will be able to find nodes, leaf nodes and edges. The dots are nodes, the edges are on the names.

When looking for a machine the query proceeds recursively into the hierarchy starting at the root. If you want to find the address of `prep.ai.mit.edu.`, your nameserver has to start asking somewhere. It starts by looking it its cache. If it knows the answer, having cached it before, it will answer right away as we saw in the last section. If it does not know it will see how closely it can match the requested name and use whatever information it has cached. In the worst case there is no match but the `.' (root) of the name, and the root servers have to be consulted. It will remove the leftmost parts one at a time, checking if it knows anything about `ai.mit.edu.`, then `mit.edu.`, then `edu.`, and if not that it does know about `.` because that was in the hints file. It will then ask a `.` server about `prep.ai.mit.edu`. This `.` server will not know the answer, but it will help your server on its way by giving a referral, telling it where to look instead. These referrals will eventually lead your server to a nameserver that knows the answer. I will illustrate that now. `+norec` means that dig is asking non-recursive questions so that we get to do the recursion ourselves. The other options are to reduce the amount of dig produces so this won't go on for too many pages:

```
$ ;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 980
;; flags: qr ra; QUERY: 1, ANSWER: 0, AUTHORITY: 13, ADDITIONAL: 0

;; AUTHORITY SECTION:
.                       518400  IN      NS      J.ROOT-SERVERS.NET.
.                       518400  IN      NS      K.ROOT-SERVERS.NET.
.                       518400  IN      NS      L.ROOT-SERVERS.NET.
.                       518400  IN      NS      M.ROOT-SERVERS.NET.
.                       518400  IN      NS      A.ROOT-SERVERS.NET.
.                       518400  IN      NS      B.ROOT-SERVERS.NET.
.                       518400  IN      NS      C.ROOT-SERVERS.NET.
.                       518400  IN      NS      D.ROOT-SERVERS.NET.
.                       518400  IN      NS      E.ROOT-SERVERS.NET.
.                       518400  IN      NS      F.ROOT-SERVERS.NET.
.                       518400  IN      NS      G.ROOT-SERVERS.NET.
.                       518400  IN      NS      H.ROOT-SERVERS.NET.
.                       518400  IN      NS      I.ROOT-SERVERS.NET.
```

This is a referral. It is giving us an "Authority section" only, no "Answer section". Our own nameserver refers us to a nameserver. Pick one at random:

```
$ dig +norec +noques +nostats +nocmd prep.ai.mit.edu. @D.ROOT-SERVERS.NET.
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 58260
;; flags: qr; QUERY: 1, ANSWER: 0, AUTHORITY: 3, ADDITIONAL: 3

;; AUTHORITY SECTION:
mit.edu.                172800  IN      NS      BITSY.mit.edu.
mit.edu.                172800  IN      NS      STRAWB.mit.edu.
mit.edu.                172800  IN      NS      W20NS.mit.edu.

;; ADDITIONAL SECTION:
BITSY.mit.edu.          172800  IN      A       18.72.0.3
STRAWB.mit.edu.         172800  IN      A       18.71.0.151
W20NS.mit.edu.          172800  IN      A       18.70.0.160
```

It refers us to MIT.EDU servers at once. Again pick one at random:

```
$ dig +norec +noques +nostats +nocmd prep.ai.mit.edu. @BITSY.mit.edu.
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 29227
;; flags: qr ra; QUERY: 1, ANSWER: 1, AUTHORITY: 4, ADDITIONAL: 4

;; ANSWER SECTION:
prep.ai.mit.edu.        10562   IN      A       198.186.203.77

;; AUTHORITY SECTION:
ai.mit.edu.             21600   IN      NS      FEDEX.ai.mit.edu.
ai.mit.edu.             21600   IN      NS      LIFE.ai.mit.edu.
ai.mit.edu.             21600   IN      NS      ALPHA-BITS.ai.mit.edu.
ai.mit.edu.             21600   IN      NS      BEET-CHEX.ai.mit.edu.

;; ADDITIONAL SECTION:
FEDEX.ai.mit.edu.       21600   IN      A       192.148.252.43
LIFE.ai.mit.edu.        21600   IN      A       128.52.32.80
ALPHA-BITS.ai.mit.edu.  21600   IN      A       128.52.32.5
BEET-CHEX.ai.mit.edu.   21600   IN      A       128.52.32.22
```

This time we got a "ANSWER SECTION", and an answer for our question. The "AUTHORITY SECTION" contains information about which servers to ask about `ai.mit.edu` the next time. So you can ask them directly the next time you wonder about `ai.mit.edu` names. Named also gathered information about `mit.edu`, so of `www.mit.edu` is requested it is much closer to being able to answer the question.

So starting at `.` we found the successive name servers for each level in the domain name by referral. If you had used your own DNS server instead of using all those other servers, your named would of-course cache all the information it found while digging this out for you, and it would not have to ask again for a while.

In the tree analogue each ``.'' in the name is a branching point. And each part between the ``.''s are the names of individual branches in the tree. One climbs the tree by taking the name we want (`prep.ai.mit.edu`) asking the root (`.`) or whatever servers father from the root toward `prep.ai.mit.edu` we have information about in the cache. Once the cache limits are reached the recursive resolver goes out asking servers, pursuing referrals (edges) further into the name.

A much less talked about, but just as important domain is `in-addr.arpa`. It too is nested like the `normal' domains. `in-addr.arpa` allows us to get the host's name when we have its address. A important thing to note here is that the IP addresses are written in reverse order in the `in-addr.arpa` domain. If you have the address of a machine: `198.186.203.77` named proceeds to find the named 77.203.168.198.in-addr.arpa/ just like it did for `prep.ai.mit.edu`. Example: Finding no cache entry for any match but `.', ask a root server, `m.root-servers.net` refers you to some other root servers. `b.root-servers.net` refers you directly to bitsy.mit.edu/. You should be able to take it from there.

5.2 Our own domain

Now to define our own domain. We're going to make the domain `linux.bogus` and define machines in it. I use a totally bogus domain name to make sure we disturb no-one Out There.

One more thing before we start: Not all characters are allowed in host names. We're restricted to the characters of the English alphabet: a-z, and numbers 0-9 and the character '-' (dash). Keep to those characters (BIND 9 will not bug you if you break this rule, BIND 8 will). Upper and lower-case characters are the same for DNS, so `pat.uio.no` is identical to `Pat.UiO.No`.

We've already started this part with this line in `named.conf`:

```
zone "0.0.127.in-addr.arpa" {
        type master;
        file "pz/127.0.0";
};
```

Please note the lack of `.' at the end of the domain names in this file. This says that now we will define the zone `0.0.127.in-addr.arpa`, that we're the master server for it and that it is stored in a file called `pz/127.0.0`. We've already set up this file, it reads:

```
$TTL 3D
@               IN      SOA     ns.linux.bogus. hostmaster.linux.bogus. (
                                1       ; Serial
                                8H      ; Refresh
                                2H      ; Retry
                                4W      ; Expire
                                1D)     ; Minimum TTL
                        NS      ns.linux.bogus.
1                       PTR     localhost.
```

Please note the `.' at the end of all the full domain names in this file, in contrast to the `named.conf` file above. Some people like to start each zone file with a `$ORIGIN` directive, but this is superfluous. The origin (where in the DNS hierarchy it belongs) of a zone file is specified in the zone section of the `named.conf` file; in this case it's `0.0.127.in-addr.arpa`.

This `zone file' contains 3 `resource records' (RRs): A SOA RR. A NS RR and a PTR RR. SOA is short for Start Of Authority. The `@' is a special notation meaning the origin, and since the `domain' column for this file says 0.0.127.in-addr.arpa the first line really means

```
0.0.127.in-addr.arpa.   IN      SOA ...
```

NS is the Name Server RR. There is no '@' at the start of this line; it is implicit since the previous line started with a '@'. Saves some typing that. So the NS line could also be written

```
0.0.127.in-addr.arpa.   IN      NS      ns.linux.bogus
```

It tells DNS what machine is the name server of the domain `0.0.127.in-addr.arpa`, it is `ns.linux.bogus`. 'ns' is a customary name for name-servers, but as with web servers who are customarily named `www.`*something*. The name may be anything.

And finally the PTR (Domain Name Pointer) record says that the host at address 1 in the subnet `0.0.127.in-addr.arpa`, i.e., 127.0.0.1 is named `localhost`.

The SOA record is the preamble to *all* zone files, and there should be exactly one in each zone file, at the top (but after the `$TTL` directive). It describes the zone, where it comes from (a machine called `ns.linux.bogus`), who is responsible for its contents (`hostmaster@linux.bogus`; you should insert your e-mail address here), what version of the zone file this is (serial: 1), and other things having to do with caching and secondary DNS servers. For the rest of the fields (refresh, retry, expire and minimum) use the numbers used in this HOWTO and you should be safe. Before the SOA comes a mandatory line, the `$TTL 3D` line. Put it in all your zone files.

Now restart your named (`rndc stop; named`) and use `dig` to examine your handy work. `-x` asks for the inverse query:

```
$ dig -x 127.0.0.1
```

```
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 30944
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONAL: 0

;; QUESTION SECTION:
;1.0.0.127.in-addr.arpa.                IN      PTR

;; ANSWER SECTION:
1.0.0.127.in-addr.arpa. 259200  IN      PTR     localhost.

;; AUTHORITY SECTION:
0.0.127.in-addr.arpa.   259200  IN      NS      ns.linux.bogus.

;; Query time: 3 msec
;; SERVER: 127.0.0.1#53(127.0.0.1)
;; WHEN: Sun Dec 23 03:02:39 2001
;; MSG SIZE  rcvd: 91
```

So it manages to get `localhost` from 127.0.0.1, good. Now for our main task, the `linux.bogus` domain, insert a new 'zone' section in `named.conf`:

```
zone "linux.bogus" {
        type master;
        notify no;
        file "pz/linux.bogus";
};
```

Note again the lack of ending `.' on the domain name in the `named.conf` file.

In the `linux.bogus` zone file we'll put some totally bogus data:

```
;
; Zone file for linux.bogus
;
; The full zone file
;
$TTL 3D
@       IN      SOA     ns.linux.bogus. hostmaster.linux.bogus. (
                        199802151       ; serial, todays date + todays serial #
                        8H              ; refresh, seconds
                        2H              ; retry, seconds
                        4W              ; expire, seconds
                        1D )            ; minimum, seconds
;
                NS      ns              ; Inet Address of name server
                MX      10 mail.linux.bogus     ; Primary Mail Exchanger
                MX      20 mail.friend.bogus.   ; Secondary Mail Exchanger
;
localhost       A       127.0.0.1
ns              A       192.168.196.2
mail            A       192.168.196.4
```

Two things must be noted about the SOA record. `ns.linux.bogus` *must* be a actual machine with a A record. It is not legal to have a CNAME record for the machine mentioned in the SOA record. Its name need not be `ns', it could be any legal host name. Next, `hostmaster.linux.bogus` should be read as hostmaster@linux.bogus. This should be a mail alias, or a mailbox, where the person(s) maintaining DNS should read mail frequently. Any mail regarding the domain will be sent to the address listed here. The name need not be `hostmaster', it can be your normal e-mail address, but the e-mail address `hostmaster' is often expected to work as well.

There is one new RR type in this file, the MX, or Mail eXchanger RR. It tells mail systems where to send mail that is addressed to `someone@linux.bogus`, namely to `mail.linux.bogus` or `mail.friend.bogus`. The number before each machine name is that MX RR's priority. The RR with the lowest number (10) is the one mail should be sent to if possible. If that fails the mail can be sent to one with a higher number, a secondary mail handler, i.e., `mail.friend.bogus` which has priority 20 here.

Reload named by running `rndc reload`. Examine the results with `dig`:

```
$ dig any linux.bogus
; <<>> DiG 9.1.3 <<>> any linux.bogus
;; global options:  printcmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 55239
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 4, AUTHORITY: 1, ADDITIONAL: 1

;; QUESTION SECTION:
;linux.bogus.               IN      ANY

;; ANSWER SECTION:
linux.bogus.        259200  IN      SOA     ns.linux.bogus. \
      hostmaster.linux.bogus. 199802151 28800 7200 2419200 86400
linux.bogus.        259200  IN      NS      ns.linux.bogus.
linux.bogus.        259200  IN      MX      20 mail.friend.bogus.
linux.bogus.        259200  IN      MX      10 mail.linux.bogus.linux.bogus.

;; AUTHORITY SECTION:
linux.bogus.        259200  IN      NS      ns.linux.bogus.

;; ADDITIONAL SECTION:
ns.linux.bogus.     259200  IN      A       192.168.196.2
```

```
;; Query time: 4 msec
;; SERVER: 127.0.0.1#53(127.0.0.1)
;; WHEN: Sun Dec 23 03:06:45 2001
;; MSG SIZE  rcvd: 184
```

Upon careful examination you will discover a bug. The line

```
linux.bogus.        259200  IN MX        10 mail.linux.bogus.linux.bogus.
```

is all wrong. It should be

```
linux.bogus.        259200  IN MX        10 mail.linux.bogus.
```

I deliberately made a mistake so you could learn from it :-) Looking in the zone file we find this line:

```
                    MX      10 mail.linux.bogus     ; Primary Mail Exchanger
```

It is missing a period. Or has a 'linux.bogus' too many. If a machine name does not end in a period in a zone file the origin is added to its end causing the double `linux.bogus.linux.bogus`. So either

```
                MX      10 mail.linux.bogus.    ; Primary Mail Exchanger
```

or

```
                MX      10 mail                 ; Primary Mail Exchanger
```

is correct. I prefer the latter form, it's less to type. There are some BIND experts that disagree, and some that agree with this. In a zone file the domain should either be written out and ended with a `.' or it should not be included at all, in which case it defaults to the origin.

I must stress that in the named.conf file there should *not* be `.'s after the domain names. You have no idea how many times a `.' too many or few have fouled up things and confused the h*ll out of people.

So having made my point here is the new zone file, with some extra information in it as well:

```
;
; Zone file for linux.bogus
;
; The full zone file
;
$TTL 3D
@       IN      SOA     ns.linux.bogus. hostmaster.linux.bogus. (
                        199802151       ; serial, todays date + todays serial #
                        8H              ; refresh, seconds
                        2H              ; retry, seconds
                        4W              ; expire, seconds
                        1D )            ; minimum, seconds
;
                TXT     "Linux.Bogus, your DNS consultants"
                NS      ns              ; Inet Address of name server
                NS      ns.friend.bogus.
                MX      10 mail         ; Primary Mail Exchanger
                MX      20 mail.friend.bogus. ; Secondary Mail Exchanger

localhost       A       127.0.0.1

gw              A       192.168.196.1
                TXT     "The router"

ns              A       192.168.196.2
                MX      10 mail
                MX      20 mail.friend.bogus.
www             CNAME   ns

donald          A       192.168.196.3
                MX      10 mail
                MX      20 mail.friend.bogus.
                TXT     "DEK"

mail            A       192.168.196.4
                MX      10 mail
                MX      20 mail.friend.bogus.

ftp             A       192.168.196.5
                MX      10 mail
                MX      20 mail.friend.bogus.
```

CNAME (Canonical NAME) is a way to give each machine several names. So www is an alias for ns. CNAME record usage is a bit controversial. But it's safe to follow the rule that a MX, CNAME or SOA record should *never* refer to a CNAME record, they should only refer to something with an A record, so it is inadvisable to have

```
foobar          CNAME   www                     ; NO!
```

but correct to have

```
foobar          CNAME   ns                      ; Yes!
```

Load the new database by running `rndc reload`, which causes named to read its files again.

```
$ dig linux.bogus axfr

; <<>> DiG 9.1.3 <<>> linux.bogus axfr
;; global options:  printcmd
linux.bogus.            259200  IN      SOA     ns.linux.bogus. hostmaster.linux.bogus. 199802151 28800 7200 2419200 86400
linux.bogus.            259200  IN      NS      ns.linux.bogus.
linux.bogus.            259200  IN      MX      10 mail.linux.bogus.
linux.bogus.            259200  IN      MX      20 mail.friend.bogus.
donald.linux.bogus.     259200  IN      A       192.168.196.3
donald.linux.bogus.     259200  IN      MX      10 mail.linux.bogus.
donald.linux.bogus.     259200  IN      MX      20 mail.friend.bogus.
donald.linux.bogus.     259200  IN      TXT     "DEK"
ftp.linux.bogus.        259200  IN      A       192.168.196.5
ftp.linux.bogus.        259200  IN      MX      10 mail.linux.bogus.
ftp.linux.bogus.        259200  IN      MX      20 mail.friend.bogus.
gw.linux.bogus.         259200  IN      A       192.168.196.1
gw.linux.bogus.         259200  IN      TXT     "The router"
localhost.linux.bogus.  259200  IN      A       127.0.0.1
mail.linux.bogus.       259200  IN      A       192.168.196.4
mail.linux.bogus.       259200  IN      MX      10 mail.linux.bogus.
mail.linux.bogus.       259200  IN      MX      20 mail.friend.bogus.
ns.linux.bogus.         259200  IN      MX      10 mail.linux.bogus.
ns.linux.bogus.         259200  IN      MX      20 mail.friend.bogus.
ns.linux.bogus.         259200  IN      A       192.168.196.2
www.linux.bogus.        259200  IN      CNAME   ns.linux.bogus.
linux.bogus.            259200  IN      SOA     ns.linux.bogus. hostmaster.linux.bogus. 199802151 28800 7200 2419200 86400
;; Query time: 41 msec
;; SERVER: 127.0.0.1#53(127.0.0.1)
;; WHEN: Sun Dec 23 03:12:31 2001
;; XFR size: 23 records
```

That's good. As you see it looks a bit like the zone file itself. Let's check what it says for `www` alone:

```
$ dig www.linux.bogus

; <<>> DiG 9.1.3 <<>> www.linux.bogus
;; global options:  printcmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 16633
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 2, AUTHORITY: 1, ADDITIONAL: 0

;; QUESTION SECTION:
;www.linux.bogus.               IN      A

;; ANSWER SECTION:
www.linux.bogus.        259200  IN      CNAME   ns.linux.bogus.
ns.linux.bogus.         259200  IN      A       192.168.196.2

;; AUTHORITY SECTION:
linux.bogus.            259200  IN      NS      ns.linux.bogus.

;; Query time: 5 msec
;; SERVER: 127.0.0.1#53(127.0.0.1)
;; WHEN: Sun Dec 23 03:14:14 2001
;; MSG SIZE  rcvd: 80
```

In other words, the real name of `www.linux.bogus` is `ns.linux.bogus`, and it gives you some of the information it has about ns as well, enough to connect to it if you were a program.

Now we're halfway.

5.3 The reverse zone

Now programs can convert the names in linux.bogus to addresses which they can connect to. But also required is a reverse zone, one making DNS able to convert from an address to a name. This name is used by a lot of servers of different kinds (FTP, IRC, WWW and others) to decide if they want to talk to you or not, and if so, maybe even how much priority you should be given. For full access to all services on the Internet a reverse zone is required.

Put this in `named.conf`:

```
zone "196.168.192.in-addr.arpa" {
        type master;
        notify no;
        file "pz/192.168.196";
};
```

This is exactly as with the `0.0.127.in-addr.arpa`, and the contents are similar:

```
$TTL 3D
@       IN      SOA     ns.linux.bogus. hostmaster.linux.bogus. (
```

```
                        199802151 ; Serial, todays date + todays serial
                        8H      ; Refresh
                        2H      ; Retry
                        4W      ; Expire
                        1D)     ; Minimum TTL
                NS      ns.linux.bogus.

1               PTR     gw.linux.bogus.
2               PTR     ns.linux.bogus.
3               PTR     donald.linux.bogus.
4               PTR     mail.linux.bogus.
5               PTR     ftp.linux.bogus.
```

Now you reload your named (`rndc reload`) and examine your work with `dig` again:

```
$ dig -x 192.168.196.4
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 58451
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONAL: 1

;; QUESTION SECTION:
;4.196.168.192.in-addr.arpa.    IN      PTR

;; ANSWER SECTION:
4.196.168.192.in-addr.arpa. 259200 IN   PTR     mail.linux.bogus.

;; AUTHORITY SECTION:
196.168.192.in-addr.arpa. 259200 IN     NS      ns.linux.bogus.

;; ADDITIONAL SECTION:
ns.linux.bogus.         259200  IN      A       192.168.196.2

;; Query time: 4 msec
;; SERVER: 127.0.0.1#53(127.0.0.1)
;; WHEN: Sun Dec 23 03:16:05 2001
;; MSG SIZE  rcvd: 107
```

---

so, it looks OK, dump the whole thing to examine that too:

```
$ dig 196.168.192.in-addr.arpa. AXFR

; <<>> DiG 9.1.3 <<>> 196.168.192.in-addr.arpa. AXFR
;; global options:  printcmd
196.168.192.in-addr.arpa. 259200 IN     SOA     ns.linux.bogus. \
        hostmaster.linux.bogus. 199802151 28800 7200 2419200 86400
196.168.192.in-addr.arpa. 259200 IN     NS      ns.linux.bogus.
1.196.168.192.in-addr.arpa. 259200 IN   PTR     gw.linux.bogus.
2.196.168.192.in-addr.arpa. 259200 IN   PTR     ns.linux.bogus.
3.196.168.192.in-addr.arpa. 259200 IN   PTR     donald.linux.bogus.
4.196.168.192.in-addr.arpa. 259200 IN   PTR     mail.linux.bogus.
5.196.168.192.in-addr.arpa. 259200 IN   PTR     ftp.linux.bogus.
196.168.192.in-addr.arpa. 259200 IN     SOA     ns.linux.bogus. \
        hostmaster.linux.bogus. 199802151 28800 7200 2419200 86400
;; Query time: 6 msec
;; SERVER: 127.0.0.1#53(127.0.0.1)
;; WHEN: Sun Dec 23 03:16:58 2001
;; XFR size: 9 records
```

Looks good! If your output didn't look like that look for error-messages in your syslog, I explained how to do that in the first section under the heading Starting named

5.4 Words of caution

There are some things I should add here. The IP numbers used in the examples above are taken from one of the blocks of 'private nets', i.e., they are not allowed to be used publicly on the Internet. So they are safe to use in an example in a HOWTO. The second thing is the `notify no;` line. It tells named not to notify its secondary (slave) servers when it has gotten a update to one of its zone files. In BIND 8 and later the named can notify the other servers listed in NS records in the zone file when a zone is updated. This is handy for ordinary use. But for private experiments with zones this feature should be off --- we don't want the experiment to pollute the Internet do we?

And, of course, this domain is highly bogus, and so are all the addresses in it. For a real example of a real-life domain see the next main-section.

5.5 Why reverse lookups don't work.

There are a couple of ``gotchas'' that normally are avoided with name lookups that are often seen when setting up reverse zones. Before you go on you need reverse lookups of your machines working on your own nameserver. If it isn't go back and fix it before continuing.

I will discuss two failures of reverse lookups as seen from outside your network:

The reverse zone isn't delegated.

When you ask a service provider for a network-address range and a domain name the domain name is normally delegated as a matter of course. A delegation is the glue NS record that helps you get from one nameserver to another as explained in the dry theory section above. You read that, right? If

your reverse zone doesn't work go back and read it. Now.

The reverse zone also needs to be delegated. If you got the `192.168.196` net with the `linux.bogus` domain from your provider they need to put `NS` records in for your reverse zone as well as for your forward zone. If you follow the chain from `in-addr.arpa` and up to your net you will probably find a break in the chain, most probably at your service provider. Having found the break in the chain contact your service-provider and ask them to correct the error.

You've got a classless subnet

This is a somewhat advanced topic, but classless subnets are very common these days and you probably have one if you're a small company.

A classless subnet is what keeps the Internet going these days. Some years ago there was much ado about the shortage of IP numbers. The smart people in IETF (the Internet Engineering Task Force, they keep the Internet working) stuck their heads together and solved the problem. At a price. The price is in part that you'll get less than a ``C'' subnet and some things may break. Please see Ask Mr. DNS for an good explanation of this and how to handle it.

Did you read it? I'm not going to explain it so please read it.

The first part of the problem is that your ISP must understand the technique described by Mr. DNS. Not all small ISPs have a working understanding of this. If so you might have to explain to them and be persistent. But be sure you understand it first ;-). They will then set up a nice reverse zone at their server which you can examine for correctness with dig.

The second and last part of the problem is that you must understand the technique. If you're unsure go back and read about it again. Then you can set up your own classless reverse zone as described by Mr. DNS.

There is another trap lurking here. (Very) Old resolvers will *not* be able to follow the `CNAME` trick in the resolving chain and will fail to reverse-resolve your machine. This can result in the service assigning it an incorrect access class, deny access or something along those lines. If you stumble into such a service the only solution (that I know of) is for your ISP to insert your PTR record directly into their trick classless zone file instead of the trick CNAME record.

Some ISPs will offer other ways to handle this, like Web based forms for you to input your reverse-mappings in or other automagical systems.

5.6 Slave servers

Once you have set up your zones correctly on the master servers you need to set up at least one slave server. Slave servers are needed for robustness. If your master goes down the people out there on the net will still be able to get information about your domain from the slave. A slave should be as long away from you as possible. Your master and slave should share as few as possible of these: Power supply, LAN, ISP, city and country. If all of these things are different for your master and slave you've found a really good slave.

A slave is simply a nameserver that copies zone files from a master. You set it up like this:

```
zone "linux.bogus" {
        type slave;
        file "sz/linux.bogus";
        masters { 192.168.196.2; };
};
```

A mechanism called zone-transfer is used to copy the data. The zone transfer is controlled by your SOA record:

```
@       IN      SOA     ns.linux.bogus. hostmaster.linux.bogus. (
                        199802151       ; serial, todays date + todays serial #
                        8H              ; refresh, seconds
                        2H              ; retry, seconds
                        4W              ; expire, seconds
                        1D )            ; minimum, seconds
```

A zone is only transferred if the serial number on the master is larger than on the slave. Every refresh interval the slave will check if the master has been updated. If the check fails (because the master is unavailable) it will retry the check every retry interval. If it continues to fail as long as the expire interval the slave will remove the zone from it's filesystem and no longer be a server for it.

Next Previous Contents