Solutions and Examples for System Administrators

DNS & BIND Cookbook

O'REILLY®

Cricket Liu

DNS and BIND Cookbook

Cricket Liu

O'REILLY®
Beijing • Cambridge • Farnham • Köln • Paris • Sebastopol • Taipei • Tokyo

CHAPTER 2

Zone Data

2.0 Introduction

With 27 different tiles in an English Scrabble set ("A" through "Z" plus the blanks) and 7 tiles in a rack, you can draw billions of different combinations. And with over 100,000 words in the Official Word List, you can assemble a lot of words from almost any of those combinations.

In DNS, there are fewer than 300 possible types of resource records, and of those, only a handful could be called common. Still, you can do a remarkable variety of interesting things with those records.

All resource records, when written in plain text (as they'd appear in a zone data file), share the following format:

```
[owner] [TTL] [class] <type> <RDATA>
```

The fields in square brackets ("[" and "]") are optional, while the fields between angle brackets ("<" and ">") are mandatory. Recipe 2.1 explains what happens when you leave out one or more of those fields.

The RDATA field often consists of multiple subfields. The number of subfields required depends upon the type of record. For example, SOA records take seven RDATA subfields, while A and NS records need just one.

A zone data file contains the resource records attached to all of the domain names in a zone. A zone's primary master name server loads the zone data file, and the zone's slaves transfer the zone data from the primary master.

2.1 Creating a Zone Data File

Problem

You need to create a data file for a zone.

Solution

Using your favorite editor, create a file in the primary master name server's working directory. Name the file after the zone whose resource records it will contain. For example, for the *foo.example* zone, you might call the zone data file *db.foo.example*.

Begin the file with a *$TTL* control statement.* This tells other name servers (not those authoritative for the zone) how long they may cache records from this zone by specifying the zone's default *time to live*. You can specify the value as an integer number of seconds or as a scaled value: an integer followed by *s* for seconds, *m* for minutes, *h* for hours, *d* for days or *w* for weeks. For example, you can specify a time to live of one day with either:

```
$TTL 86400
```

or:

```
$TTL 1d
```

You can even concatenate two scaled values, like so:

```
$TTL 1d12h
```

Time to live values between one hour and one day are common.

Next, add an SOA record for the zone. The SOA record contains information about the whole zone, including how often the zone's slave name servers should check to see whether the zone has changed. The SOA record begins with the zone's domain name, a specification of the zone's class (which is almost always *IN*, for Internet), and the type mnemonic *SOA*. After the type, the SOA record requires seven fields:

The MNAME field
: Specify the domain name of the zone's primary master name server.

The RNAME field
: Specify an email address at which the administrator of the zone can be reached. Substitute a dot (".") for the "@" in the email address.

The zone's serial number
: If you'll only be changing the zone manually, by editing the zone data file, consider using the format *YYYYMMDDVV*, where *YYYY* is the year, *MM* is the numeric month (from 1 to 12), *DD* is the date, and *VV* is a two-digit version number that starts at 00. This will give you a handy indicator of when you last updated the zone.

The zone's refresh value
: This specifies how frequently slave name servers for the zone should check their master name server to see whether the zone's serial number has been incremented (indicating that the zone has changed). This value isn't particularly important if you use the NOTIFY mechanism, which enables your primary

* Assuming you're running a version of BIND newer than 8.2—and you should be.

master name server to *tell* slaves when the zone changes, but values between one hour and three hours are common.

The zone's retry value
: This specifies how often the zone's slave name servers should check their master name server after a check of the serial number has failed. As with refresh, this isn't that important if you use NOTIFY, but values between 15 minutes and 1 hour are common.

The zone's expire value
: This specifies how long the zone's slave name servers will continue responding if they're unable to reach their master name server to find out the current serial number. Since this determines how long your slaves will answer queries for data in the zone in the event of an outage, you should make it fairly long. Values of several weeks to a month are common.

The zone's negative caching time to live value
: This determines how long other name servers can cache negative answers given out by the zone's authoritative name servers. One such negative answer is *NXDOMAIN*, which indicates that the domain name the remote name server looked up doesn't exist in the zone. This value should be fairly low, between 15 minutes and 3 hours.

Here's a sample SOA record for the *foo.example* zone:

```
foo.example.    IN    SOA    ns1.foo.example.    (
                hostmaster.foo.example.
                2002040700
                1h
                15m
                30d
                1h )
```

Since we ran out of room for the record on the first line, we ended the line with "(", which tells the name server to treat all text between the "(" and a successive ")" as though it were on a single line. (We could also have just kept typing the whole record on a single line, but that would have been hard to read.)

Finally, add NS records listing the domain names of the authoritative name servers for the zone. You probably specified these name servers when you registered your domain name.

```
foo.example.    IN    NS    ns1.foo.example.
foo.example.    IN    NS    ns2.foo.example.
foo.example.    IN    NS    ns.isp.net.
```

Discussion

The domain names in the resource records all end in dots to keep the name server from appending the origin to them. The default origin for a zone data file is just the

domain name of the zone, which in this case is *foo.example*, so we could have written the SOA record as:

```
@    IN    SOA    ns1    (
     hostmaster
     2002040700
     1h
     15m
     30d
     1h )
```

("@" is short for "the current origin.")

Since these first several resource records in the zone are all attached to the same domain name (*foo.example*, in our example), you can specify the domain name for just the first of them and begin the rest of the records with whitespace (spaces or tabs):

```
@    IN    SOA    ns1    (
     hostmaster
     2002040700
     1h
     15m
     30d
     1h )

     IN    NS    ns1.foo.example.
     IN    NS    ns2.foo.example.
     IN    NS    ns.isp.net.
```

The name server interprets records that begin with whitespace as belonging to the most recently specified domain name.

See Also

Recipe 1.14 for creating a *named.conf* file, Recipe 1.15 for configuring a primary master name server, and Chapter 4 of *DNS and BIND*.

2.2 Adding a Host

Problem

You need to add a host to DNS.

Solution

Add an A and a PTR record for the host to the appropriate zones (which are almost certainly two different zones: a forward-mapping and a reverse-mapping zone). For example, to add a host called *host.foo.example* with the IP address 10.0.0.1 to DNS, you could add this record to the *foo.example* zone data file:

```
host.foo.example.    IN    A    10.0.0.1
```

And you'd add this record to the zone data file for the reverse-mapping zone, which might be *10.in-addr.arpa*, *0.10.in-addr.arpa*, or *0.0.10.in-addr.arpa*, depending on how you break up administration of your reverse-mapping domain:

```
1.0.0.10.in-addr.arpa.    IN    PTR    host.foo.example.
```

Discussion

You're free to take advantage of the origin in the file to abbreviate the resource records. For example, if you're adding the A record to a line in the zone data file in which the origin is *foo.example*, you can write:

```
host    IN    A    10.0.0.1
```

If you're adding the PTR record on a line in which the origin is 0.0.10.in-addr.arpa, you can write:

```
10    IN    PTR    host.foo.example.
```

Since the default class is *IN*, for Internet, you can leave out the `IN`, too.

It's important to add PTR records for your hosts. Without PTR records, your hosts' addresses won't map to domain names, so they won't be able to access services that require reverse mapping, and your network management software may not identify them automatically.

You may also want to add other records for the host. If the host's domain name might appear on the right side of an email address, add an MX record specifying where mail addressed to the host should be delivered.

See Also

Recipe 2.4, for how to add an MX record; Recipe 2.9 to limit how long the records can be cached, Recipe 2.10 to learn how to handle multihomed hosts, and Chapter 4 of *DNS and BIND*.

2.3 Adding an Alias

Problem

You need to create an alias from one domain name to another.

Solution

Add a CNAME record to the zone that the alias belongs in. For example, to make *a.foo.example* an alias for *b.bar.example*, add this CNAME record to the *foo.example* zone data file:

```
a.foo.example.    IN    CNAME    b.bar.example.
```

Discussion

Note that a CNAME record makes the alias equivalent to the target of the alias. Queries for any types of record attached to the alias will end up as queries for the same type of record, but attached to the domain name the alias points to. Consequently, you can't add any other types of records to a domain name that is an alias.

You also shouldn't use aliases on the right side of other types of records, such as NS and MX records. The consumers of NS and MX records—name servers and mail servers, respectively—don't expect aliases on the right side and therefore don't process them correctly. The only kind of record that allows an alias on the right side is the CNAME record itself: You can point an alias to another alias, as long as the alias chain ends at a non-alias domain name. Make sure the chain isn't more than eight links long, though, and beware alias loops.

Finally, note that the CNAME record belongs in the zone that contains the domain name of the alias, not the target of the alias.

See Also

Recipe 2.6 to learn how to set up virtual web hosts; and Chapter 4 and the "Using CNAME Records" section of Chapter 16 in *DNS and BIND*.

2.4 Adding a Mail Destination

Problem

You need to add a mail destination to DNS.

Solution

Add one or more MX records to the zone that contains the domain name of the mail destination. The MX records specify the mail server or servers that accept mail addressed to that mail destination. Each MX record requires a preference value that tells mailers sending mail the order in which to contact the destination's mail servers. The *lower* the preference value, the *more preferred* the mail server.

For example, to tell mailers to send mail addressed to *foo.example* (such as an email message addressed to *hostmaster@foo.example*) to *mail.foo.example*, and *smtp.isp.net* only if *mail.foo.example* isn't up or isn't accepting connections, add these MX records to the *foo.example* zone data file:

```
foo.example.    IN    MX    0 mail.foo.example.
foo.example.    IN    MX    10 smtp.isp.net.
```

Discussion

The preference value is an unsigned, 16-bit number, so between 0 and 65535. The magnitude of the number isn't important: the preference value doesn't represent any particular units. What's important is that the preference values for a domain name's MX records, taken together, tell a sending mailer the order in which it should use the destination's mail servers.

Most mailers will spread the load randomly among mail servers listed at the same preference value. This can come in handy with popular mail destinations: You can list a number of mail servers at the most preferred preference value and sending mailers will distribute the delivery of your mail among those mail servers.

The mail server must be specified as a single domain name, not an IP address. If you use an IP address on the right side of an MX record, mailers—expecting a domain name there—will try to look up the IP address as a domain name. This causes unnecessary queries to the root name servers, and fails to return an IP address, anyway.

It's up to you (or your fellow postmasters) to configure the mail servers to accept mail addressed to the destination. Make sure the most preferred mail exchangers understand that the mail destination is local, and make sure less preferred mail exchangers are configured to relay mail addressed to the destination.

See Also

RFC 2821 for authoritative information on SMTP and use of MX records, and Chapter 5 of *DNS and BIND*.

2.5 Making the Domain Name of Your Zone Point to Your Web Server

Problem

You want the domain name of your zone to point to your web server.

Solution

Add an A record to the domain name of your zone pointing to the IP address of your web server:

```
foo.example.    IN    A    10.0.0.1
```

Discussion

Adding such an A record lets people specify just *http://foo.example/* (leaving out the leading "www") when accessing your web site. Several popular web sites publish their URLs in this form, including CNN.

Many people try to solve this problem by adding a CNAME record to the domain name of the zone, rather than an A record:

```
foo.example.    IN    CNAME    www.foo.example.
```

This, however, is illegal because it violates the dictum that an alias have no records other than a CNAME record associated with it.

If you have multiple web servers, you can add multiple A records for the domain name of your zone:

```
foo.example.    IN    A    10.0.0.1
foo.example.    IN    A    10.0.0.2
foo.example.    IN    A    10.0.0.3
```

The records are given out in round robin order, by default, as described in Recipe 2.7.

See Also

Recipe 2.3 for more information on CNAME records, Recipe 2.6 for pointing a domain name at a particular URL, not just a particular web server, and Recipe 2.7, for a description of round robin.

2.6 Pointing a Domain Name to a Particular URL

Problem

You want people who access one of your domain names to reach a particular URL.

Solution

Add an A record to the zone to which the domain name belongs, pointing to the IP address of the web server:

```
mylink.foo.example.    IN    A    10.0.0.1
```

Then configure the web server to direct browsers requesting *http://mylink.foo.example* to the appropriate directory on your web server.

Discussion

Most of this solution is configured on the web server using a facility called "virtual hosts." The web server needs to associate your domain name, when it appears in the

HTTP/1.1 "Host" header, with a particular "document root," a directory in the web server's document tree.

If the domain name of the web server is in a zone run by someone else, or you already have a domain name in your zone pointing to the address of the web server, you can use a CNAME record instead of an A record:

```
mylink.foo.example.    IN    CNAME    www.isp.net.
```

This way, if the IP address of the web server changes, your domain name will continue to point to the right place.

Of course, if someone else runs the web server, you'll need their cooperation to set up the association between *mylink.foo.example* and the appropriate directory.

See Also

Recipe 2.5 for pointing a domain name at a web server, the Apache Software Foundation's online documents on virtual hosts at *http://httpd.apache.org/docs/vhosts/name-based.html* and *http://httpd.apache.org/docs-2.0/vhosts/*, and "HTTP/1.1 Virtual Hosts" in Chapter 3 of *Apache: The Definitive Guide*.

2.7 Setting Up Round Robin Load Distribution

Problem

You want to set up round robin for a domain name.

Solution

Just add multiple A records to the domain name. For example:

```
www.foo.example.    IN    A    10.0.0.1
www.foo.example.    IN    A    10.0.0.2
www.foo.example.    IN    A    10.0.0.3
```

In successive answers to queries for *www.foo.example*'s address, the *foo.example* name servers will rotate the order in which they return the A records, moving the first A record to the end of the list after each response.

Discussion

All modern name servers give out resource records in round robin order by default. Only very old name servers (before BIND 4.9) don't support round robin.

Remember that round robin isn't load *balancing*. The name server has no idea how busy the web servers that serve *www.foo.example*'s content are, or even whether

they're all responding. If the name server at 10.0.0.1 were to crash, the name server would still give out its address first a third of the time. For true load balancing, you need something more than just DNS.

See Also

Recipe 3.18 for details on how round robin works and how to disable it.

2.8 Adding a Domain Name in a Subdomain Without Creating a New Zone

Problem

You want to add a domain name in a subdomain of your zone, but don't want to create a new zone and delegate it from your current zone.

Solution

Just add the records associated with the new domain name, specifying the subdomain in the domain name. For example, to add the domain name *a.b.foo.example* to the *foo.example* zone, you could add this record to the *foo.example* zone data file:

```
a.b.foo.example.    IN    A    10.0.0.4
```

Doing this implicitly creates the subdomain *b.foo.example* and the domain name *a.b.foo.example*. The subdomain *b.foo.example* is part of the *foo.example* zone (as is the domain name *a.b.foo.example*), and will be included in transfers of the zone to slave name servers.

If the origin in the zone data file is *foo.example*, the default, you can also write the record as:

```
a.b         IN    A    10.0.0.4
```

Discussion

Sometimes the solution to a problem is just the most obvious of the possibilities. That's the case both with setting up round robin and with this problem. But many administrators—even the very experienced—aren't accustomed to adding domain names to their zones that have multiple labels to the left of their zones' domain names. They think of the domain names in their zones as always having the format *host.domain-name-of-zone*, rather than any number of labels ending in the domain name of the zone.

See Also

For more on intrazone subdomains, see "Creating a Subdomain in the Parent's Zone" in Chapter 9 of *DNS and BIND*. If you do want to delegate the subdomain and create a new zone, see Recipe 6.1.

2.9 Preventing Remote Name Servers from Caching a Resource Record

Problem

You want to prevent remote name servers from caching one or more records in your zone.

Solution

Give the record (or records) an explicit—and low—time to live (TTL). For example, to keep other name servers from caching your web server's addresses, you could add these A records to the zone data file:

```
www.foo.example.    1    IN    A    10.0.0.1
www.foo.example.    1    IN    A    10.0.0.2
www.foo.example.    1    IN    A    10.0.0.3
```

Specify the explicit TTL between the domain name owner of the record and the class field. By default, the value is an integer number of seconds. You can also use a scaled value, as you would in the *$TTL* control statement.

Discussion

Note that the TTLs for the three *www.foo.example* A records are the same; that's no accident. If you were to use different TTLs for records in the same RRset (of the same type, and attached to the same domain name), a remote name server might age only some of them out, leading to unpredictable results. Consequently, modern name servers notice this misconfiguration and "trim" mismatched TTLs within a single RRset to the smallest TTL of the group.

Why did I use a TTL of one instead of zero? After all, a zero TTL would seem to say, "Don't cache this record." Unfortunately, TTLs of zero tickle a bug in some older name servers, which age out the records before returning them to the resolver that initiated the query. D'oh!

See Also

Recipe 2.1 for the syntax of scaled values, and "Changing TTLs" in Chapter 8 of *DNS and BIND*.

2.10 Adding a Multihomed Host

Problem

You want to add a multihomed host to DNS.

Solution

Add multiple A records to the host's domain name, one per IP address. For example, for a file server with two network interfaces, you might add these records:

```
fs.foo.example.    IN    A    10.0.0.9
fs.foo.example.    IN    A    192.168.0.9
```

To handle reverse mapping for the host, add one PTR record to each of the appropriate two reverse-mapping zones:

```
9.0.0.10.in-addr.arpa.    IN    PTR    fs.foo.example.
```

and

```
9.0.168.192.in-addr.arpa.    IN    PTR    fs.foo.example.
```

Discussion

Clients looking up the address of *fs.foo.example* will see both IP addresses, and can choose which one to use (though most clients will just use the first address returned). Remember that, by default, they'll be returned in round robin order.

For troubleshooting purposes, you may want to add two more A records, each of which maps to just one of your multihomed host's addresses. For example:

```
fs-eth0.foo.example.    IN    A    10.0.0.9
fs-eth1.foo.example.    IN    A    192.168.0.9
```

This lets you test whether a particular network interface on the file server is up, by pinging *fs-eth0.foo.example*, for instance. You probably shouldn't add PTR records mapping the addresses back to these interface-specific names, though: most software can't handle multiple reverse mappings for a single IP address.

See Also

Recipe 2.7 for the behavior of round robin, and Chapter 4 of *DNS and BIND*.

2.11 Updating a Name Server's Root Hints File

Problem

You need to update a name server's root hints file.

Solution

FTP a copy of the most recent root hints file from *ftp.rs.internic.net*. It's called *named.root*, in the directory *domain*.

Discussion

The root hints file, which tells a name server the domain names and addresses of the root name servers, doesn't need to be updated often. The "current" version dates to August 1997, and the file can be slightly out-of-date without causing adverse effects. Still, you should probably check every six months or so to see if it's changed.

If you do download a new root hints file, remember to change the name of the file to whatever you have defined in your *zone* statement for the root hints, and then reload the name server.

See Also

"The Root Hints Data" in Chapter 4 and "Keeping the Root Hints Current" in Chapter 7 of *DNS and BIND*.

2.12 Using a Single Data File for Multiple Zones

Problem

You want to use a single data file for multiple zones.

Solution

Create a "template" zone data file. Make sure that all of the owner names of records in the zone are "@" (short for the origin) or relative; that is, written without a trailing dot. For example:

```
@    IN    SOA    ns1.isp.net. hostmaster.isp.net. (
     2002040900
     3600
     900
     604800
     3600 )
```

```
        IN    NS    ns1.isp.net.
        IN    NS    ns2.isp.net.

        IN    MX    smtp.isp.net.

        IN    A    192.168.0.99

    www IN    CNAME    @
```

Add *zone* statements to your name server's *named.conf* file, configuring it as primary master for the various zones, and specifying the "template" zone data file in the *file* substatement each time. For example:

```
zone "foo.example" {
    type master;
    file "db.template";
};

zone "bar.example" {
    type master;
    file "db.template";
};

zone "baz.example" {
    type master;
    file "db.template";
};
```

Since each *zone* statement sets the default origin to the domain name of the zone in the data file, the SOA record and NS records will always end up attached to the right domain name, and the rest of the records will end up "translated" into the zone.

Discussion

This technique will only work if all of the zones are very similar—nearly identical, in fact. The zones must contain the same number and mix of records, and the records in the zones can only differ by the domain name of the zone. For example, if the domain name *www.foo.example* is an alias for *a.foo.example* in the *foo.example* zone, then *www.bar.example* will be an alias for *a.bar.example* in the *bar.example* zone.

The name server must be the primary master for all of the zones; there's no way to set up an equivalent slave name server that uses the same backup zone data file for all of its zones, since name servers write fully qualified domain names to backup zone data files.

Also, none of the zones can be dynamically updated, since dynamic updates to a zone would cause the name server to rewrite the zone data file, and the rewritten zone data file would also contain fully qualified domain names.

See Also

Recipe 2.1 for understanding the default origin of a zone data file.

2.13 Using Multiple Data Files for a Single Zone

Problem

You want to break a zone into multiple data files, possibly to organize the large number of resource records logically.

Solution

Use the *$INCLUDE* control statement in your top-level zone data file, which interpolates the contents of another file. For example, to include the contents of the file *db.foo.example.hosts* into the data file for the zone *foo.example*, you could use this *$INCLUDE* control statement:

```
$INCLUDE db.foo.example.hosts
```

Discussion

The origin in the included file is, by default, the same as the origin in the file that includes it. If you'd like to change the origin in the included file, specify the new origin as the second argument to the *$INCLUDE* control statement:

```
$INCLUDE db.subdomain.foo.example.hosts subdomain.foo.example
```

On the line after the *$INCLUDE* statement, the origin reverts to its previous setting.

See Also

Recipe 2.8, which explains how to create a subdomain within the same zone.

2.14 Resetting Your Zone's Serial Number

Problem

You need to reset your serial number to some low value, possibly because you inadvertently added a digit to it.

Solution

If you've accidentally incremented your serial number to a value larger than $2^{32} - 1$ (4,294,967,295), first find out what your current serial number is—because it probably isn't what you think it is (the serial number is only 32 bits large). The easiest way to do this is to use a query tool, such as *dig*, to look up your zone's SOA record:

```
$ dig soa foo.example

; <<>> DiG 9.2.1 <<>> soa foo.example
;; global options:  printcmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 4335
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 2, ADDITIONAL: 2

;; QUESTION SECTION:
;foo.example.                   IN      SOA

;; ANSWER SECTION:
foo.example.            86400   IN      SOA     ns1.foo.example. hostmaster.foo.
example. 2002021239 3600 900 2592000 3600
```

If the current serial number is less than 2,147,483,647, add 2,147,483,647 to the serial number. Wait for all of your zone's slave name servers to pick up the new version of the zone (if you're using NOTIFY, that shouldn't take long). Then set the serial number to your target.

If the current serial number is larger than 2,147,483,647, just set the serial number to the number you want.

Discussion

Whahuh? Why on Earth does this work?

Name servers compare serial numbers using *sequence space arithmetic*, which ain't your grandpa's 'rithmetic. In sequence space arithmetic, you have a finite set of integers, but each number has a "next" number. After 0 comes 1, then 2, all the way to 4,294,967,295 ($2^{32} - 1$). The next number after 4,294,967,295 is 0. Think of it like a clock: The hour after 1:00 is 2:00, and the hour after 12:00 is 1:00.

Half of the numbers are larger than any given number, and the other half are smaller. With a set of 2^{32} possible serial numbers, half ($2^{31} - 1$, actually) are larger than any given serial number, and half are smaller.

Consider the serial number 1,000,000,000. The next $2^{31} - 1$ serial numbers, 1,000,000,001 through 3,147,483,647, are larger. The $2^{31} - 1$ serial numbers after that, 3,147,483,648 through 4,294,967,295 ($2^{32} - 1$) and 0 to 999,999,999, are smaller. Yes, Alice, in the world of serial numbers, 3,147,483,648 is *smaller* than 1,000,000,000.

So when you add 2,147,483,647 ($2^{31} - 1$) to a serial number, you're actually adding the largest increment possible—add a larger number and the result will actually be *smaller* than the old serial number, and your zone's slaves won't transfer the zone.

Once all the slaves have the new zone, you can simply set the serial number to the serial number you want, which is now considered larger than the current serial number.

If you're not comfortable with this New Math, try out the script *reset_serial.pl*, included in the *tar* file that accompanies this book (see the Preface for where to get it). *reset_serial.pl* takes as arguments your current serial number and the serial number you want to get to, and tells you how to get there.

There's also a brute force method for resetting your serial number: set the serial number to your target in the zone data file. Then delete your zone's backup data files on all of your slaves and restart *named*. Your slave name servers won't have any choice but to transfer the zone, regardless of its serial number.

This won't work if you don't have administrative control of all of your slaves, of course, and it has all the elegance of using a flat-head screwdriver as a chisel.

See Also

"Starting Over with a New Serial Number" in Chapter 7 of *DNS and BIND*, and RFC 1982 for an explanation of serial number arithmetic.

2.15 Making Manual Changes to a Dynamically Updated Zone

Problem

You want to edit a zone data file by hand, but the zone is dynamically updated.

Solution

On a BIND 8 name server, stop the name server with *ndc stop*, delete the zone's dynamic update log file (whose name is the name of the zone data file with *.log* appended, by default) and the IXFR log file, if any (whose name is the zone data file's plus *.ixfr*). Then edit the zone data file and start the name server.

On a BIND 9 name server, stop the name server with *rndc stop*, delete the zone's journal file (whose name is the zone data file's with *.jnl* on the end), edit the zone data file and start the name server again.

On a BIND 9.3.0 or newer name server, you can freeze the zone with *rndc freeze*, edit the zone data file, and unfreeze the zone with *rndc unfreeze*.

Discussion

With dynamic zones, it's better to make all changes to the zone using dynamic updates. However, sometimes that's just not practical.

The problem is that, with most BIND name servers, if you edit a zone data file while the name server is running, you can lose your changes. When you restart the name server (reloading dynamic zones doesn't work), the name server will rewrite the zone data file if it has received any dynamic updates to the zone that haven't yet been written to the zone data file. What happens to your changes? Poof! They disappear without a trace, like so many dot-coms. You need to stop the name server before editing the zone data file. And that means your name server may miss dynamic updates while you're manually editing the zone data file, so be quick about it!

Also, when you edit the zone data file manually, the changes you make don't get entered into the dynamic update log—the *.log* file, for BIND 8, and the *.jnl* file for BIND 9. When the name server loads the zone data file and then checks the content of the log file, it discovers a gap: It's missing the record of the last change, the one you made manually. So you have to delete the log file before loading.

The price of deleting the log file is that your zone's slaves won't be able to get an incremental zone transfer on their next try, since the record of the last change—necessary to get them up-to-date—is missing. They'll request an incremental zone transfer but receive a full zone transfer instead.

The BIND 9.3.0 name server has two new *rndc* commands, *freeze* and *unfreeze*, which allow you to suspend and resume the processing of dynamic updates to a zone. *freeze* also deletes the log file. So you can *rndc freeze* the zone, edit the zone data file, then *rndc unfreeze*.

See Also

Recipe 5.19, to learn how to use the *nsupdate* program to modify a zone.

2.16 Moving a Host

Problem

You want to move a host from one address to another.

Solution

At least one TTL before the move, reduce the TTL on the host's A record and PTR record to a low number, like 60 seconds. For example, say you're planning on moving the host *z.foo.example*. If its current A record looks like this:

```
z.foo.example.    86400    IN    A    192.168.0.254
```

Reduce the TTL at least a day (86,400 seconds) ahead of time, like this:

```
z.foo.example.     60     IN     A     192.168.0.254
```

At the same time, reduce the TTL on the host's PTR record:

```
254.0.168.192.in-addr.arpa.     60     IN     PTR     z.foo.example.
```

Then, after you've moved the host, change the A record to reflect the host's new address and restore the TTL:

```
z.foo.example.     86400     IN     A     10.0.0.254
```

Delete the old PTR record and add one (to the appropriate zone data file!) for the new address:

```
254.0.0.10.in-addr.arpa.     86400     IN     PTR     z.foo.example.
```

Discussion

You need to reduce the TTL on the old records ahead of time to keep name servers from caching them just before the move. If you left the TTL alone, a remote name server could cache the old address just before you made the change, and it would take some time for that record to time out. If you don't use NOTIFY, you should also add in the refresh time of the zones the records are in, since it could take that long for the lower TTL records to make it out to all of your slaves.

This technique applies to more than just A and PTR records, of course. You could just as easily use it to change MX records or any other record type. If it's a name server you're moving, however, or you need to change your zone's NS records, see Recipes 6.6 and 6.7.

Notice that the new PTR record may well belong in a different zone data file than the old one.

See Also

Recipe 2.9, for an explanation how to reduce the TTL on a single record; Recipes 6.6 and 6.7, for moving a name server and changing all of a zone's name servers; and "Changing TTLs" in Chapter 8 of *DNS and BIND*.

2.17 Mapping Any Domain Name in a Zone to a Single IP Address

Problem

You want to map every domain name in a zone to a single IP address.

Solution

Add an A record to the zone attached to the wildcard domain name. For example:

```
*.foo.example.    IN    A    10.0.0.1
```

Discussion

Technically, this record doesn't map *every* domain name in the zone to 10.0.0.1. In fact, the wildcard domain name doesn't apply to domain names in the zone data file. Say you also had the domain name *ns1.foo.example* in the *foo.example* zone:

```
ns1.foo.example.    IN    A    192.168.0.1
```

The wildcard domain name *wouldn't* match queries for the address of *ns1.foo.example*, which is probably a good thing, since *ns1.foo.example* has a different address. The wildcard domain name wouldn't apply to domain names that own other types of records, either. For example, you might have this record in the zone:

```
text.foo.example.    IN    TXT    "Text comment"
```

Queries for the address of *text.foo.example* would return an empty answer, because *text.foo.example* has no addresses.

So what *does* the wildcard domain name apply to? Queries for domain names in the zone that don't appear in the zone data file, which means any domain name you can think of that ends in *foo.example*, doesn't appear in the *foo.example* zone data file, and isn't part of a delegated subdomain of *foo.example*.

Wildcard domain names can own other types of records, too. Take, for example, this CNAME record:

```
*.foo.example.    IN    CNAME    foo.example.
```

This creates aliases from any domain name in the zone without explicit records attached to the domain name *foo.example*. So iif you leave out explicit records for *www.foo.example*, someone looking up *www.foo.example* would find that domain name is an alias for *foo.example*. Someone looking up *zaphod.beeblebrox.foo.example* would find that it, too, is an alias for *foo.example*—assuming you didn't have any records attached to the domain name *zaphod.beeblebrox.foo.example*, that is. So you might think of a wildcard as a "default" domain name for a zone: any explicit domain name in the zone has only the records you give it, but the wildcard applies to every other domain name in the zone.

As the *zaphod.beeblebrox.foo.example* example suggests, wildcards can match more than one label. In fact, a wildcard matches zero or more labels. The wildcard domain name in the CNAME record wouldn't match just *foo.example*, though, since even at zero labels, **.foo.example* has one more dot than *foo.example*.

See Also

"Wildcards" in Chapter 16 of *DNS and BIND*.

2.18 Adding Similar Records

Problem

You want to add a number of records that differ only slightly.

Solution

Use the *$GENERATE* control statement to specify a template that the name server will use to generate a group of similar records. For example, to add a series of PTR records that differ only by a single digit, you could use this *$GENERATE* control statement:

```
$GENERATE 11-20 $.0.168.192.in-addr.arpa. PTR dhcp-$.foo.example.
```

Your BIND name server will read the range (11–20) and it will also read the template (*$.0.168.192.in-addr.arpa. PTR dhcp-$.foo.example.*) from the *$GENERATE* control statement. Then it will iterate through the range, replacing any dollar signs ("$") in the template with the current value, creating 10 PTR records:

```
11.0.168.192.in-addr.arpa. PTR dhcp-11.foo.example.
12.0.168.192.in-addr.arpa. PTR dhcp-12.foo.example.
13.0.168.192.in-addr.arpa. PTR dhcp-13.foo.example.
...
20.0.168.192.in-addr.arpa. PTR dhcp-20.foo.example.
```

Discussion

$GENERATE supports a limited set of record types: A, AAAA, CNAME, DNAME, NS and PTR. Also, the template can't contain a TTL or a class field, just a type.

If you want to get fancy, you can also step through the range using the range format *start-stop/range*. So *0–100/2* would count from 0 to 100 by twos.

BIND 8.2 introduced *$GENERATE* to the world. BIND 9.1.0 introduced *$GENERATE* to the BIND 9 releases.

Note that, unlike the *$INCLUDE* and *$ORIGIN* control statements, *$GENERATE* is only supported by BIND name servers; you can't use it in a zone data file on a Microsoft DNS Server, for example.

See Also

"Subnetting on a Non-Octet Boundary" in Chapter 9 of *DNS and BIND*, and Section 6.3.6 of the BIND 9 Administrator Reference Manual.

2.19 Making Your Services Easy to Find

Problem

You want to make it easy for users to find the services you offer.

Solution

Give your servers "functional" domain names. For example, most users will expect to find an organization's FTP server at the domain name *ftp.domain-name-of-zone*. In most cases, the domain name can be an alias for the canonical name of the host running the service; that's not possible with name servers or mail servers, though.

Other common functional domain names include:

domain-name-of-zone
: The zone's domain name, by convention, owns one or more A records that point to the organization's web server, and one or more MX records that tell mailers where to deliver mail addressed to the organization's users.

imap.domain-name-of-zone
: An IMAP mail server.

mail.domain-name-of-zone
: An SMTP mail server. Note that this domain name can't be an alias; it must own an A record. Moreover, the mail server must recognize itself as this domain name in order to prevent mail loops.

ns[N].domain-name-of-zone
: The authoritative name servers for your zone. Since there are often more than one, use an integer to distinguish between them: *ns1*, *ns2*, etc. Or, for the unapologetically geeky, *ns0*, *ns1*, etc. Note that these domain names *can't* be aliases; they *must* own A records.

ntp.domain-name-of-zone
: An NTP (Network Time Protocol) server. If you have more than one, disambiguate them by using *ntp1*, *ntp2*, etc.

pop.domain-name-of-zone
: A POP mail server.

smtp.domain-name-of-zone
: An alternative to *mail.domain-name-of-zone*. As with *mail.domain-name-of-zone*, this must own an A record.

www.domain-name-of-zone
: This convention is so common it's almost not worth discussing, but most users expect to find an organization's web site here.

Discussion

One big benefit of using functional domain names is that you can move a service from one host to another by changing only the A or CNAME record for the functional domain name, and without changing the configuration of every client of that service. For example, if you moved your NTP server from *a.foo.example* to *b.foo.example*, you could just change the *ntp.foo.example* CNAME record to:

```
ntp.foo.example.    IN    CNAME    b.foo.example.
```

Assuming you'd configured your NTP clients to refer to your NTP server by the domain name *ntp.foo.example*, you wouldn't have to make any changes to your clients' configuration.

The domain names of mail servers and name servers are special because of the way they're used. The domain name of a name server will usually appear in an NS record, delegating a zone to that name server. A name server sending that NS record in a referral will only add A records for the name server's domain name to the response. If the domain name owns a CNAME record, the name server won't find it.

Likewise, mail servers sending mail to your email addresses expect to find A records for the mail servers you list in your MX records. If you use CNAME records, they won't find the address they're after.

Also, if one of your backup mail servers receives the email, it will "trim" the list of MX records by removing itself and any less-preferred mail servers. If it doesn't recognize itself in the list because you've used an alias in an MX record, it may try to send mail to itself, or to a less-preferred mail server.

2.20 Storing the Location of a Host in DNS

Problem

You want to store the location of a host in your zone data.

Solution

Depending on what you mean by "location," add either a TXT or LOC record to the host's domain name.

Many administrators want to store a descriptive location for the host in DNS. For example, you might want to specify that the host *a.foo.example* is in on your Building 20's level C, near post C3K. To do that, you might add this TXT record to your zone:

```
a.foo.example.    IN    TXT    "Building 20, level C, post C3K"
```

If, on the other hand, you'd like to specify the host's geographical location (i.e., its latitude, longitude, and altitude), you can add a LOC record to your zone. For example, if *a.foo.example* is also at 40 degrees, 2 minutes, 0.373 seconds north latitude; 105 degrees, 17 minutes, 23.528 seconds west longitude; and 1,638 meters altitude, you could add this LOC record to your zone:

```
a.foo.example.    IN    LOC    40 2 0.373 N 105 17 23.528 W 1638m
```

Discussion

The TXT record is enormously versatile, since you can put just about *anything* into the RDATA. Just remember that only people who know to look up the TXT records for a domain name will find the data you store there. Also, if you add multiple TXT records to a domain name, there's no guarantee of the order in which the name server will return them.

The LOC record, on the other hand, is absolutely specialized: it only stores geographical location data. The format is exactly as I've shown it above, with separate RDATA fields for degrees, minutes, and seconds, followed by N for north, S for south, E for east, and W for west. And you can use negative elevation values if you happen to use a colocation facility in Death Valley.

If you're not sure what your hosts' latitude, longitude, and altitude are and you can't persuade your boss that you need a new GPS receiver to find out, you can use Etak's Eagle Geocoder (*www.geocode.com/eagle.html-ssi*) or AirNav's Airport Information, (*www.airnav.com/airports/*) to find the values for your address or a nearby airport, respectively.

See Also

For more information on LOC records, see the "Location" section of Chapter 16 of *DNS and BIND*, RFC 1876, or Christopher Davis's excellent web site at *http://www.ckdhr.com/dns-loc/*.

2.21 Filtering a Host Table into Zone Data Files

Problem

You want to filter an existing host table, such as an */etc/hosts* file, into zone data files.

Solution

Use a tool such as *h2n* to filter your host table into the corresponding zone data files. With *h2n*, you specify the domain name of a forward-mapping zone to create as the argument to the *–d* option and the networks associated with that zone as the argument to one or more *–n* options. For example, the following command would build data files for the *foo.example* and *168.192.in-addr.arpa* zones:

```
% h2n -d foo.example -n 192.168
```

These zone data files would each contain a SOA record and an NS record pointing to the local host, as well as A records or PTR records for hosts in */etc/hosts* on the 192.168/16 network. Additional options allow you to create other records, including NS records pointing to other name servers.

Discussion

You can get a copy of *h2n* from the tar ball that accompanies *DNS and BIND*, located at *ftp.oreilly.com/published/oreilly/nutshell/dnsbind/dns.tar.Z*. Also, Andris Kalnozols of Hewlett-Packard has enhanced *h2n* significantly; he makes his souped-up version available at *ftp.hpl.hp.com/pub/h2n/h2n.tar.gz*.

There are other tools available for filtering host table-format data into zone data files; *h2n* is only one option. Take a look at the contents of *bind-contrib.tar.gz*, available in the same directory as the latest BIND 8 release, for some of your options.

See Also

Recipe 1.11 for how to get a copy of BIND (or *bind-contrib.tar.gz*), and "Tools" in Chapter 4 of *DNS and BIND*.