Linux networking guide: Part 3

# THE DOMAIN NAME SYSTEM

**In this, the third installment of our simple guide to configuring Linux networks from the command line, Bruce Richardson shows us how to configure DNS on both client and server**

## Introduction

The examples in the first two articles in this series used IP addresses exclusively to identify networks, subnets and hosts. But while an IP address is all a computer needs, humans work better with names.

Every so often, on a newsgroup or mailing list, some newcomer to Internet technologies will suggest that the system of IP addresses should be entirely replaced by one based on names. This is not practical: an IP address only requires four bytes to store it (IPv4 addresses, anyway), whereas a text string requires at least one byte for each character. Since each IP packet contains the address of both its source and its destination, this would add quite an overhead to TCP/IP networks.

What is needed, then, is a mechanism that allows humans to assign meaningful names to hosts on the network and enables computers to translate – to resolve – these names into IP addresses. That is the subject of this article. This article will show you how the Domain Name System is used to organise TCP/IP networks, how to configure a computer running Linux to use DNS and how to configure a DNS server on Linux.

## Table 1: DNS Record Types

| Type | Description |
|---|---|
| SOA | Start Of Authority record. If a name server has an SOA record for a domain then it is an authoratitive server for that domain. |
| A | Address record. Associates a name with an address. An address may have multiple A records, each associating it with a different name. |
| CNAME | Alias record. Gives an alternate name for a host that already has an A record. NS, MX and PTR records may not point to CNAME records and some people avoid all use of CNAMES, saying they make a mess of DNS. |
| NS | Identifies a host as a name server for a domain. |
| MX | Identifies a host as a mail server that will accept mail for the domain. |
| PTR | Pointer records are used to map addresses to names, the inverse of A records. Their use is explained further on in this section. The name in a PTR record must have an associated A record, not a CNAME record. |

## An overview of DNS

Internet domains are organised into a top-down tree-structure. At the very top is the root domain. Beneath that are the Top Level Domains, the generic TLDs like .com, .net etc and the geographical TLDs like .uk, .nz and so on. Each of those domains is further subdivided and so on. Domains further down the tree are considered subdomains of the upper domains, so the debian.org domain is within the .org domain and the uk.debian.org domain is within both the debian.org and .org domains and they are all subdomains of the root domain.

## Names

A Fully Qualified Domain Name (FQDN) is constructed by taking the name of a host or domain and adding to it the names of all the containing domains, using "." as a separator. So *ftp.uk.debian.org* is the FQDN of the host named *ftp* that resides within the *uk.debian.org* domain. *ftp* is the unqaulified name, referred to in this article as the short name.

An important point to remember is that the root domain is itself represented by ".". So the FQDN for the ftp host is actually *ftp.uk.debian.org.*. Almost all applications will add the final "." for themselves as long as the rightmost domain matches the name of a TLD. This is not the case with name servers, however. When configuring a name server it is important always to include the final "." or the daemon will attempt to fully qualify the name by appending the FQDN of the local domain.

## Name servers

For each domain there must be a name server (a minimum of two, for Internet domains) which can give authorative answers to queries about names within the domain. A name server may be authorative for an entire domain including all its subdomains or it may delegate responsibility for a subdomain to another name server.

The area within a domain that the name server does not delegate is called a zone. Name servers can be authoritative for multiple domains and so have many zones.

Name servers maintain databases of information about their domains. Each record in the database holds information of a specific type (see Table 1).

## Masters and slaves

Configuring multiple name servers for a domain provides redundancy and eases the load on each server. To ease the burden of administration, name servers can be configured as slave servers, getting their data from a master server in a regular process called a zone update.

## Root name servers

The root name servers are authoritative for the root domain (and in most cases for the generic Top Level Domains as well). Each chain of delegation starts with them and so they are the ultimate source of the answers to all DNS queries.

## Query resolution

DNS name servers accept two kinds of queries: recursive and iterative. In a recursive query, the name server searches the DNS heirarchy until it finds an answer. In an iterative query the name server simply gives the best answer it knows. This is best illustrated by example.

A host in the *example.org* domain wants to know the address of *www.linux.org.uk*. It sends a recursive query to the local name server, *ns0.example.org*. *ns0* sends an iterative query to one of the root servers, which refers it to *ns.uu.net*, a name server authoritative for the uk domain. *ns0* then sends an iterative query to *ns.uu.net*. *ns.uu.net* refers *ns0* to *ns1.nic.uk*, which is authoritative for *org.uk*. *ns1.org.uk* refers *ns0* to *tallyho.bc.nu* and since *tallyho* is one of the name servers that is authoritative for the *linux.org.uk* domain, it is able to give the address of *www.linux.org.uk*. *ns0* returns the answer to the host that made the original query.

## Caching

In the example above, *ns0* doesn't throw away the answer to the query. Instead, it keeps it in a cache for a period of time. If it is asked the same query within that period it can give the answer without having to refer onwards.

Servers answering iterative answers may also use their cache, so *ns.uu.net* will also be able to give the address of *www.linux.org.uk* for a while. Caching thus eases the burden on the DNS system in general and top level name servers in particular.

If the actual details for *www.linux.org.uk* change, those name servers which have the old details in their caches will be serving up incorrect answers. For this reason, the SOA record of each name server includes settings which indicate how long other name servers should cache its replies. Even so, the downside to caching is that changes to your DNS set-up will take a while to propagate throughout the Internet.

## Technicalities

The standard port number for DNS queries is 53. Queries are normally carried out over UDP, though TCP may be used if the data involved is too big to fit into a UDP datagram.

## Mapping addresses to names

Sometimes you want to find out what name is associated with an address. For this a special domain was created, the *in-addr.arpa* domain. Address-to-name queries are solved by looking within that domain for PTR records which list the name matching an address. PTR records are constructed by reversing the IP address and appending ip-addr.arpa, so to find the name associated with the address *195.92.249.252* you would do a DNS query for *252.249.92.195.in-addr.arpa*. The inversion of the address is done because DNS places the most significant information to the right. This allows the query to go first to the namserver authorative for in-addr.arpa, then to the nameserver for *195.in-addr.arpa* and so on.

## An example network

The rest of this article will use as the basis of its examples the internal network of an imaginary company. It is a small organisation whose public domain is managed by its ISP. All of its hosts are on a private, internal network behind a NAT-ed firewall and are not visible to the Internet, so the local domain is called "internal". This allows a simpler example (only one name server, no slaves).

## Configuring the resolver

Unix systems come with a library that is used to resolve host names, called the resolver. (Some applications, e.g. Netscape Navigator, use their own resolvers. The Netscape one is particularly brain-dead.) The Linux resolver library is called Resolv+ and

## Table 2: The Internal Domain

| Hostname | Address | Description |
|---|---|---|
| gateway | 192.168.10.1 | Gateway to the Internet, runs firewall and NAT. |
| Alpha | 192.168.10.2 | File server. |
| Oddjob | 192.168.10.3 | Used for a variety of tasks including backups and printing |
| mailbox | 192.168.10.4 | The internal IMAP mailstore. |
| Squid | N/A | This used to be a separate box acting as HTTP proxy for the workstations. That application has now been moved onto gateway. Making squid an alias for gateway allowed this to happen without reconfiguring any other applications or workstations. |
| ns | 192.168.10.254 | Nameserver. Also runs DHCP. |
| All the other computers on the network are assigned addresses by the DHCP server on ns. | | |

is an enhanced version of the library from BIND, the Berkely DNS server application. To set up a Linux box to make proper use of DNS, you edit the resolver's config files.

## Naming your computer

This isn't, in fact, directly associated with the resolver, but many of the networkworking applications on a Linux system need to associate a primary name with the computer they run on. To do this dynamically, use the hostname command:

```
hostname oddjob
```

This won't survive a reboot, so we also want to record it in a config file for the initscripts to find find and configure. With some distributions (e.g. Debian), the name is simply written to /etc/hostname. On Red Hat you need to edit the HOSTNAME line in /etc/sysconfig/networks.

## The resolver config files

Back in the early days of the Arpanet, before there was such a thing as DNS, each computer on the network kept a local copy of a file called hosts.txt, which they downloaded via ftp from the Network Information Centre at regular intervals. This system broke down as the network grew but the */etc/hosts* file is a relic from that time.

Each entry in the hosts file lists an IP address, the name associated with it and any aliases, as in this example:

```
127.0.0.1        localhost
192.168.10.1     gateway.internal gateway squid
192.168.10.2     alpha.internal alpha
192.168.10.3     oddjob.internal oddjob
192.168.10.4     mailbox.internal mailbox
192.168.10.254   ns.internal ns
```

Adding entries to */etc/hosts* allows the resolver to resolve names without consulting a DNS server. Copying the above example to all the hosts on the network would eliminate the need for a local name server. The administrator of this network, though, prefers the centralisation advantages of DNS, so alpha's hosts file is simpler:

```
127.0.0.1        localhost
192.168.10.2     alpha.internal alpha
```

The file */etc/resolv.conf* can can hold various entries that define the behaviour of the resolver, of which the most commonly used are:

- *nameserver* – Add a nameserver entry for each DNS server that you want the computer to consult. Only one server is needed but adding extra ones gives the computer options if the first one is busy.
- *domain* – Names the local domain. If given a short name (e.g "beta"), the resolver will attempt to resolve it within this domain (that is, it will combine the shortname with the domain name to make a FQDN and then try to resolve that).
- *search* – Defines a list of domains against which the computer should attempt to resolve short names, overriding the default which is just to search the local domain.

Here is alpha's resolv.conf file:

```
# /etc/resolv.conf
domain internal
nameserver 192.168.10.254
```

If this file is not present then the resolver looks for a nameserver on *127.0.0.1*, deduces the local domain from the hostname and its matching line in */etc/hosts* and has a search list consisting of the local domain only.

The file */etc/host.conf* can take options which define the general behaviour of the resolver, as opposed to the more specific options in resolv.conf. If it is absent, sensible defaults are used. Here is a typical configuration:

```
# /etc/host.conf
order hosts,bind
multi on
```

The first entry tells the resolver to consult */etc/hosts*

### Main BIND config file

```
# /etc/named.conf
options {
        directory "/var/cache/bind";
};

zone "." {
        type hint "/etc/bind/db.root";
        file "/etc/bind/db.root";
};

zone "internal" {
        type master;
        file "db.internal";
};

zone "0.0.127.in-addr.arpa" {
        type master
        file "db.root"
};

zone "10.168.192.in-addr.arpa" {
        type master;
        file "db.10.168.192";
};
```

before trying any nameservers. The second tells the resolver that if it finds multiple addresses for a given name it should return them all, rather than just the first.

## So far, so good

If you have followed all this, you now know how to configure a typical Linux box to resolve names properly. Obviously, if you are setting up DNS for the first time then you should configure the DNS server before referencing it from any other machines.

## The Berkely Internet Name Daemon

BIND is the most commonly used DNS server in the world and so the one I have chosen for this example. Specifically, I use BIND 8. BIND 9 is a recent major rewrite which is still turning up significant bugs and has not yet supplanted 8.x as the most popular version.

You can get the source code from the Internet Software Consortium's Web site or FTP site (see the Info boxout). I recommend installing the BIND package that comes with your distribution, though.

## The main config file

BIND expects to find its main configuration file in */etc/named.conf*, though you can put it somewhere else and pass an appropriate command line option. The format for named.conf is extremely simple, as can be seen in the config file for ns.internal, listed in the Main BIND config file boxout. The basic pattern is of a series of blocks, bounded by braces.

The first block contains the global options. In this example there is just one option, which sets the default directory to be */var/cache/named*. Any file that doesn't have an explicitly set location will be looked for there.

The second block tells BIND that the root hints file is in */etc/bind/db.root*.
This file contains a list of all the root name servers and their addresses and should be kept up to date for BIND to function properly. A simple way to do this is to query a reliable name server, like this:

```
dig @reliable.name.server . ns > root.hints
```

Then copy that to wherever you keep your hints file and restart the daemon.

Each block after that simply names a zone for which this name server is authoritative, states that this is a master (rather than slave) server for that zone and names the file containing the zone's details. Since no path is given for the files, they should be placed in */var/cache/named*.

At this point, if you looked carefully at the zones listed, you might ask "Why a reverse-mapping zone for the loopback interface?". The simple answer is that your name server will ocassionally be asked to perform a reverse look-up on the loopback address, so this covers it.

## BIND db file for internal domain

```
internal. IN SOA ns.internal. postmaster.example.org.uk. (
         1       ; Serial
         10800   ; Refresh after 3 hours (10800 seconds)
         3600    ; Retry after 1 hour
         604800  ; Expire after 1 week
         86400)  ; Minimum TTL is 1 day

internal.        IN NS ns.internal.

; Addresses
localhost.internal.      IN A 127.0.0.1
gateway.internal.        IN A 192.168.10.1
alpha.internal.          IN A 192.168.10.2
oddjob.internal.         IN A 192.168.10.3
mailbox.internal.        IN A 192.168.10.4
ns.internal.              IN A 192.168.10.254

; Aliases
squid.internal.          IN CNAME gateway.internal.
```

## The data files

Next we must create the data files for each zone. The file for the main internal domain is shown in the BIND db file for internal domain boxout. Please note that all FQDNS end with "." – do not forget this.

First we have the SOA record (the IN SOA identifies it as an Internet Start Of Authority record). It begins with the name of the domain, "internal.". Then comes the name of the primary name server, followed by the email address of the main email contact (with the "@" replaced by "."). Finally there is a block of settings. These mostly relate to slave servers, which we shall skip. The TTL setting has a broader import, though, as it is returned with each query response. It tells the querying host how long it can reasonably cache the response before checking back. A TTL of one day is very common.

Next comes an NS record identifying ns as a nameserver for the domain, followed by A records for each named host on the network. Finally there is a CNAME record making squid.internal an alias for gateway.internal.

## Starting and maintenance

Now all you need to do is start the daemon. The daemon itself is called named. If you have moved the config file you will need to pass it an option to tell it where:

```
/usr/sbin/named -b /etc/bind/named.conf
```

And that's it: not the intimidating process you may have heard it was. Just be sure to keep your root hints file up to date. Each time you update the data files, restart the daemon or send it a SIGHUP signal.

## Info

ISC Web site
*http://www.isc.org/*
BIND FTP download
*ftp://ftp.isc.org/isc/bind/src/cur/bind-8/*
djbdns Web site
*http://cr.yp.to/djbdns.html/*