# NetInfo Editions 4.x User Manual

Xedoc Software Development Pty. Ltd.

May 1997

**NetInfo Editions 4.x User Manual**

Xedoc Software Development Pty Ltd.
PO Box 3038
Burnley North VIC 3121
Australia
Telephone +61 3 9428 0788
Fax +61 3 9428 0786
Internet netinfo@xedoc.com.au

**Product Development**

Brett Adam
J. Matthew Pryor
Cameron Bromley
Sean Hiscock

**Version 1.x**

Andrew C. Bernard

**Version 2.x**

Aris Colp

**Version 3.x**

Aris Colp
Sean Woodhouse
Luke Howard

**Version 4.x**

Luke Howard
Sean Woodhouse

**Manual Development**

Margaret Hassall, RTfM Technical Publishing Pty Ltd.

**Special Thanks**

David B. Wertheimer

# Contents

# Contents

# Contents

# Contents

# Preface

## Organisation

This manual is a practical guide to using *NetInfo*. It is directed towards UNIX system administrators. It assumes some knowledge of the UNIX operating system, but does not assume that all users are necessarily experienced at administering a network.

All administrators should read the *Introduction* and *Concepts* chapters at least once. Those of you who have used NetInfo on NeXT machines will find most of the concepts familiar. Administrators who are experienced with NIS will find that some concepts, such as domains, are treated differently in NetInfo.

The *NetInfo Design* chapter explains the philosophy behind the NetInfo domain hierarchy.

The *Configuration - Quick Start* chapter provides an alternative to the *Manual Configuration* chapter for those customers with existing NetInfo installations. The *Manual Configuration* chapter assumes that the reader does not already have a NetInfo based network, and must therefore start from scratch.

The chapter entitled *Using NetInfo* guides users through the NetInfo functions and utilities in detail.

The *Server Edition Features* chapter provides details of the additional features in the Server Edition that provide for automatic host addition, diskless booting and hostname management. It also discusses the optimisations and extended networking support provided in the Server Edition.

The *NIS Emulation* chapter provides further technical details on the technique that NetInfo uses when installed on a UNIX system.

The *Reference* chapter contains manual page entries for all supplied tools.

## Related Documents

Administrators should refer to their system manuals for information specific to their installation. This manual describes how to use *NetInfo* only, it does not advise on system or network configuration.

This manual describes how *NetInfo* inter-operates with other services such as NIS and BIND, however, it assumes administrators are familiar with these services. If readers are not familiar, they should consult the relevant system documentation supplied with the products.

**Terms and Conventions**

UNIX Commands

All UNIX commands are written using `courier` typeface. All commands are single line entries, though some have been written in this manual over two lines. Users should be aware that no <Return> is required at the end of the first line.

**Warnings**

A warning symbol placed next to text indicates that the information should be read before continuing.

**ChecksManual**

The check symbol is used to indicate when operations can be checked before continuing with the next step.

# Chapter 1

# Introduction to NetInfo

*NetInfo* simplifies system administration across a UNIX network. It is based on *NetInfo* from NeXT Computer, Inc. Systems running on NeXT and other machines can interact with each other.

*NetInfo* is a database containing UNIX configuration information accessible across a network. A set of tools is provided to access this database.

On standard UNIX machines, configuration information, such as user and group account details, file systems, peripheral devices, host details, and so on, is kept in flat files, usually stored in the `/etc` directory. When machines are connected in a network, much of this information is duplicated in flat files on each machine.

On a local area network, it is possible for an administrator to maintain these flat files for each machine in the network, but as a network grows, this job becomes very complex and tedious, and the files can become inconsistent.

Many UNIX vendors provide a system called NIS (Network Information Service), which provides tools to manage UNIX system configuration files across a network. NIS distributes information maps in their entirety to all NIS servers on a network as required. For small networks this is adequate, however, as a network grows, these map files also grow, and the time delays incurred during data transfer become unproductive. Currently, NIS is available on a range of UNIX based machines.

*NetInfo* allows system administrators to manage the administrative information of a heterogeneous network of UNIX based machines. It is 100% compatible with NetInfo for NeXT systems, and provides for complete integration between NeXT and non-NeXT computers.

In this manual, the name “NetInfo” is used to refer to *NetInfo Editions* as well as for *NetInfo* on NeXT machines.

## Implementation

NetInfo; is a database of network administration information for a network of computers running under the UNIX operating system. A set of command line tools is provided to access this database if necessary, however, most database access is made through system calls.

NetInfo provides administrators with the following key features:

**Hierarchical Domains**

Information stored in NetInfo databases can be organised into domain hierarchies. Just as directories on disks greatly enhance the organisation of large numbers of files, NetInfo domains enhance the organisation of network data.

A domain can hold information about a single computer, information shared amongst departmental computers or information for an entire company. If information is not found in a machine's default domain, the 'parent' domains are searched until a definitive result is obtained.

**Fast, Automatic Propagation**

NetInfo propagates only the incremental changes to the database to other machines on the network. Because NetInfo only updates data which has changed, network traffic, and hence, transfer time, is greatly reduced. Moreover, changes are propagated automatically, ensuring that every machine which needs to be updated is updated.

**Distributed Administration**

NetInfo databases may be maintained from any machine on the network, not just the NetInfo server. This offers enormous benefits if a network is spread over a wide geographic area or if the server machine does not have a console. Changes to the database can be made via simple command-line utilities or via the graphical user interface on the NeXT.

**Backwards Compatibility**

NetInfo is backward compatible with both NIS and previous file based administration. NetInfo inter-operates fully with BIND and NIS;. Thus, you can still use these tools if necessary, allowing you to convert your network over to NetInfo gradually.

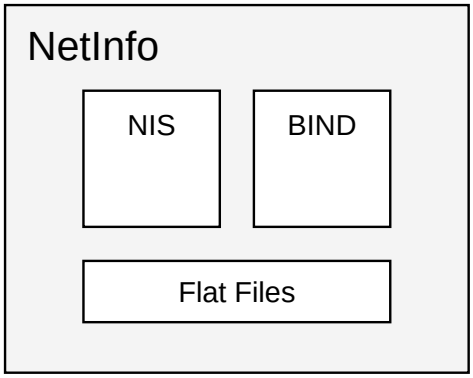



*Figure 1.1 - NetInfo Compatibility*

**Customisation & Extensibility**

NetInfo provides facilities for the storage of custom information. Clients can *write* to NetInfo. A full programmatic interface is provided for clients enabling them to read and write to the NetInfo database. This makes it possible to maintain information that is specific to the needs of a particular organisation through a single system.

**Multi-platform**

*NetInfo* inter-operates fully with NetInfo running on NeXT computers. No special configuration is required to achieve complete integration of NeXT and non-NeXT systems on the one network. NetInfo comes as standard system software with every NeXT computer. Databases created on the other UNIX hosts can be accessed using the graphical user interface on the NeXT.

# Chapter 2

# NetInfo Concepts

**UNIX Administration**

The **system administrator** is the person responsible for the upkeep of the computer system. This includes maintaining the configuration files, setting up new users, ensuring there is enough disk space and that users are able to access the datafiles and programs they require, backing up files, bringing up and shutting down the systems, and other administrative tasks as required.

The system administrator usually logs in as the **superuser** usually with the login name **root**. A superuser has special privileges to make changes to the system. Although you can set up the login id to be something different, NetInfo depends on this name being “root”, as do many other programs.

**Configuration Files**

Most configuration files are kept in the `/etc` directory. This includes the password file (`/etc/passwd`) which contains a list of all users who are able to use the system. Other files, such as the groups file (`/etc/group`) contains information about what groups exist, and which users are members. The hosts file (`/etc/hosts`) stores information about the name of a host and its Internet address.

There are many other files which are used by the system to determine its configuration. These files are all **flat**, or **ASCII** files, and must exist in order for the system to run successfully. When machines are connected in a network, much of this information is duplicated in flat files on each machine.

NetInfo manages the information normally stored in these flat files in the **NetInfo database**.

**Networks and Distributed Systems**

A **network** is an interconnected collection of computers, i.e., those capable of exchanging information. It allows resources, such as hardware, information, and services, to be shared amongst the users of an organisation.

A **distributed system** is one in which the computing functions are dispersed among several physical computing elements. Users don't need to be aware of where information or services are physically located in this type of system.

A distributed system can be considered a special case of a network. The term **network** in this document refers to networks which include distributed systems.

## Network Types and Setup

An organisation may set up a network for various reasons:

- Multiple Locations
  The organisation may have systems at more than one location, and they want the users at those locations to be able to share programs and information.
- Resource Sharing and Privacy
  Users in various departments in an organisation may share hardware such as processors, printers and tape drives, but may want to keep data separate.
- Communication
  Users in an organisation may want to contact each other using electronic mail. Users may also want to be able to connect to other organisations using a public network.

Networks confined to an office, or a floor, can be connected as **local area networks (LAN)**. Usually, users are able to share printers, disks, and other peripherals using a local network. The types of cables used define a local network and are restricted to a maximum cable length. Several local networks can be connected together using devices such as repeaters.

Network connections between, for example, interstate offices of the same company, must be connected differently than local networks. They may be connected via dedicated lines, microwave links, the telephone system with modem connection, or by some other method. This type of network is usually referred to as a **wide area network (WAN)**.

A **public network** is one that allows the connection of many individual networks. It has a well defined interface and a set of protocols by which information is transferred. This allows users and companies to transfer information, including electronic mail and news.

## NetInfo Networks

The physical aspect of the network connection is not relevant to NetInfo. It assumes that the machines, or **hosts**, are equipped to connect into an appropriate existing network, whether local, wide area, or public.

A **heterogeneous** network is one that is made up of machines of different architectures and vendors. In general, the term includes machines that are running different operating systems, but for the purposes of this document, all machines are assumed to be running **UNIX**.

NetInfo can be used on any TCP/IP network.

## The Internet

The **Internet** is a public network available across the world. It allows organisations to connect to it in order to exchange information, mail, and news.

Each machine that is part of the Internet must have a unique address. The **Internet address**, (also called the **IP address**) is a 4-byte number that is written as four numbers separated by periods. The Internet address is made up of two parts:

- the **network address**
- the **host number**

*Example:*

*Internet address*

NeXT machines are supplied with the following default three-byte network address: **192.42.172**.

Host numbers from 1 to 250 (251 - 255 reserved) can then be used to distinguish the hosts in the local network using the 4th byte.

For example,

```
|-network-| host

192.42.172.1
192.42.172.2
192.42.172.3
   ...
192.42.172.250
```

**NetInfo Addresses**

NetInfo recognises hosts in its network using a unique Internet address.

If your network is small and is likely to remain isolated, it is possible to use the default network numbers supplied with your machines.

If however, you intend to connect to the public network, you will have to request a network number from the DDN Network Information Centre. Information on contacting this organisation should be supplied with your hardware.

Depending on the size of your organisation, you will receive either a 1-byte, 2-byte or 3-byte (most common) *network address*. The *host number* will then be 3-, 2-, or 1-byte respectively to make up the four byte address.

All Internet addresses must be 4-bytes when the host number is included.

Within UNIX, “127.0.0.1” is defined as a machine's local Internet address. NetInfo uses the reserved name, **localhost**, so that a host can refer to itself.

*Example:*

*Company Structure*

The following example will be used throughout this document to illustrate networks and their management by NetInfo.

A company, ACME Software, has three major organisational divisions: Systems Development, Administration, and Sales. These divisions communicate with each other through electronic mail. The General Managers of ACME deal with each of the divisions.

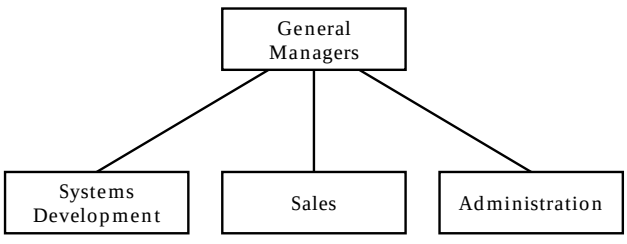



*Figure 2.1 - ACME Organisational Structure*

*Example:*

*Company Structure*

*Resources*

ACME has three UNIX computers, with the following hostnames and Internet addresses:

| | |
|---|---|
| *alpha* | 192.42.172.1 |
| *bravo* | 192.42.172.2 |
| *charlie* | 192.42.172.3 |

Two of the machines, known as *alpha* and *bravo*, are used by the System Development division. The other machine, known as *charlie*, is shared by the Admin and Sales divisions. The general managers and the superuser (login id "root") are the only users who have access to all machines.

These machines are connected in a network so that resources such as printers and disk drives can be shared, and so that the users can communicate with each other via electronic mail.

In the course of the exercises, the following machines will be added to the network.

| | |
|---|---|
| *delta* | 192.42.172.4 |
| *foxtrot* | 192.42.172.5 |
| *golf* | 192.42.172.6 |

*Users*

The following list shows the staff members of the organisation, and what division they work in. This information is normally part of the "password" configuration file.

| | |
|---|---|
| Superuser: | root |
| Managers: | genman |
| Development: | chris, bing, jo |
| Sales: | han, robin |
| Administration: | alex, pat |

## NetInfo Administration

NetInfo administration is based on the concepts of domains and databases.

### Domains

A **domain** is an abstract collection of administrative information about a group of users and the resources to which they have access. Domains can be linked together in a **hierarchy**.

Each level in the hierarchy is called a **domain level**. Information can be made available to selected levels in the hierarchy. This provides a mechanism to allow access to some resources, and to keep others private.

A domain, typically, has access to all the resources in its child domains, and in those domains lower than its children. Using set terminology, a child domain is a subset of its parent.

The domain hierarchy is structured like an inverted tree. At the top of the hierarchy is the **root domain**, represented by a slash ("/"). The **leaves** of the domain tree are the local host domains.

Each host must have a local database, called “local”. If a machine is not connected to the network, then the local domain also serves as the root domain. If there is a separate root domain, it must be called “network”.

*Example:*

*Domain Structure*

ACME has three organisational divisions:

- System Development - with two machines: *alpha* and *bravo.*
- Sales, and
- Administration - sharing one machine, called *charlie.*

**Domain Organisation**

Each host must have a local domain, which takes the name of the host (**alpha**, **bravo**, and **charlie**). These three domains are the leaves of the domain hierarchy.

Two abstract domains must be created to allocate the machines to the separate divisions.

- **admin**
  As Sales and Administration share a machine, they must also share a domain.
- **sysdev**
  Create a system development domain which has access to the two machines.

The General Managers, however, must be able to access information in all sections, and so they would be members of the root domain, **/**.

A 3-level domain structure is required as follows:

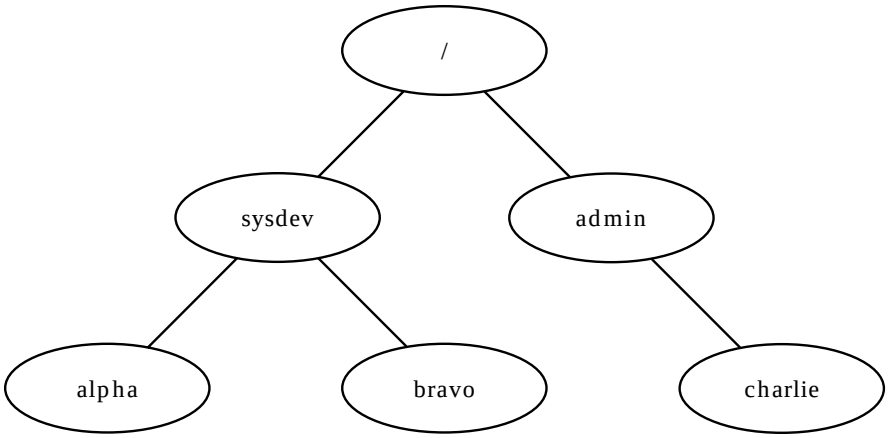



*Figure 2.2 - Domain Structure*

When a host needs to find information (for example, a login id) it first looks in its local domain. If it doesn't find the information, it then looks in its parent domain (if there is one), and continues looking up the hierarchical tree until either the information is found, or the root domain is reached.

*Example:*

*Domain Access*

User Access using the above structure:

- Admin and Sales
  Staff in this department will be registered as users in the **admin** domain. They can login to host *charlie* only.
- System Development
  Development staff will be registered as users in the **sysdev** domain. They can then login to either *alpha* or *bravo*.
- General Managers
  All users who are general managers should be registered in the root domain, /. This means they can login to any machine in the hierarchy.
- Superuser
  The superuser, whose login id must be “root”, should also be registered in the root domain, so that they, too, can login to any machine.

  The superuser must also be registered in the local domains. This is so that, should the domain tree 'break' (due to machines being down or disconnected from the network), root can still log in.

**NetInfo Database**

Domain information is stored in a **NetInfo database**. NetInfo does not use or change the flat files directly (e.g., `/etc/passwd`), but information from these files can be loaded into or from the database structures.

The same sort of information that is stored in the flat files is stored in the database, and used by the system to determine how the resources and users are to be administered.

The flat files, however, are used by the system at boot time and so cannot be removed when NetInfo is installed. The password file (`/etc/passwd`) must contain at least an entry for **root**, and the groups file must contain an entry for the group that **root** belongs to.

The database is physically located in a subdirectory of `/etc/netinfo` on the host on which it was created. The subdirectory is usually named after the domain, with an extension of “`.nidb`”. The program, **nidomain**, manages the creation of the database subdirectories. This program is explained in full in the *Reference* chapter of the manual.

A database is said to **serve** a domain, that is, it stores the information relevant to the domain. This information includes host information such as Internet address, and a list of valid users.

*Example:*

*Database Files*

As each machine must have a local domain, then a ***local*** database must be created on each host.

The **root** domain can be served from a database on any host in the hierarchy, and must be called ***network***. The **sysdev** and **admin** domains must be served from databases on any host that is part of their respective domains.

Suppose, for this example, both the **root** domain (***network*** database) and the **sysdev** domain are served from databases on the host *alpha*, and the **admin** domain is served from a database on host *charlie*. The directory structure on each machine is as follows:

| Host | Subdirectories |
|---|---|
| *alpha* | `/etc/netinfo/local.nidb`<br>`/etc/netinfo/network.nidb`<br>`/etc/netinfo/sysdev.nidb` |
| *bravo* | `/etc/netinfo/local.nidb` |
| *charlie* | `/etc/netinfo/local.nidb`<br>`/etc/netinfo/admin.nidb` |

**Binding Domains and Databases**

Once created, a database exists until it is destroyed. A domain, on the other hand, only exists when NetInfo is running and it has been bound into the hierarchy.

To **bind** into the hierarchy, a domain must have information about its parent domain, and any child domains. As the root domain does not have a parent, it is recognised by the knowledge that it is served from the database called ***network***. A domain doesn't have any knowledge of its own **domain name**: that is stored in its parent domain.

Binding is achieved by setting up properties in each database. These properties specify the database that a domain is served from, the database that a domain's parent is served from, and the databases that serve any children of the domain.

**Naming Conventions**

There are several concepts in NetInfo and UNIX which use similar, and hence confusing, naming conventions.

*UNIX*

Directory pathnames in the UNIX file system are represented using a slash (“/”) as a separator. The ancestor of all directories is the root directory, which has the name “`/`”.

In this manual, UNIX file system pathnames are written using the `courier` type face. For example, the flat file containing password information is represented as:

```
/etc/passwd
```

*Domains*

Domains are also organised in a hierarchy. The root domain is called "/". Its child domains are named similarly to the UNIX file system, with the slash character used as a separator.

In this manual, domains are written using **boldface** type. For example, the full name of the "sysdev" domain created in an earlier example is:

**/sysdev**

This domain was created with two child domains, "alpha" and "bravo":

**/sysdev/alpha** and
**/sysdev/bravo**

Domains can be referred to using relative (dot) notation. The parent is referred to as "**..**", and a domain can refer to itself using a single dot, "**.**".

For example, the parent domain of **/sysdev/alpha**, whose absolute domain name is **/sysdev**, can be referred to from the "alpha" domain as "**..**". From **/sysdev**, **/sysdev/alpha** can be referred to simply as **alpha**.

*Hostname*

Each machine, or host, must have a unique name. In this manual, host names are written using *italic* type. For example, the host names of the three machines used in examples so far are as follows:

*alpha*, *bravo* and *charlie*.

*Database - Internal Names*

A NetInfo database, which serves a domain, has an *internal* structure that is a directory hierarchy. Information is stored in the database within these directories. Directories contain zero or more properties, each of which can have zero or more values. Each directory has a unique numeric id.

Internally, the database also begins at a root level, called "**/**". The slash is used as a separator to specify pathnames in the internal database hierarchy.

In this manual, internal database pathnames are written using **`bold courier`** typeface.

For example, information about the users who have access to the resources in a domain is recorded in the following internal subdirectory:

**`/users`**

As another example, information about the machines available to the domain must be stored in a subdirectory of the **`/machines`** directory, named after the host. The information for host *alpha* would be stored in the following internal directory:

**`/machines/alpha`**

*Database - Physical File Name*

The *internal* structure should not be confused with the *physical location* of the NetInfo database.

Physically, a NetInfo database is stored in a UNIX file called `collection`, located within a UNIX directory called `/etc/netinfo/`*`database_tag`*`.nidb`. As this file grows, extra files are created, called `extension_N`, where `N` is generated by the system and usually corresponds to the database directory id stored within it.

*Database - Tags*

A database serves a domain. The database and domain names do not always have to be the same, though some consistency in naming is recommended. A database is referred to by its name, called its **tag**. A database can also be referred to by its database address, explained in the next section.

The tag needn't be the same for all databases serving a particular domain. However, it usually is the same on all servers to simplify administration. A tag can, and usually does, differ from the domain name.

One suggestion for a naming structure is as follows:

If the relative domain name is, say, **sysdev**, give its associated database a name, or tag called ***sysdev_db***. This convention distinguishes between the domain and the database, but at the same time, retains the relationship between them. The physical directory then has the name: `/etc/netinfo/sysdev_db.nidb.`

In this manual, database tags are written using ***bold italic*** type.

In the case of the compulsory database (local domain), however, the tag is fixed: all databases for local domains must have the tag ***local***. If the network database exists, then its tag is also fixed: it must be ***network***.

For example, a UNIX file, `/etc/netinfo/local.nidb` must exist on each host on which NetInfo runs. We can say the database with tag, ***local***, is used to serve the **local** domain.

*Database Address*

A domain does not actually exist until it is bound into the domain hierarchy. Binding is specified by entering values into the database that serves the domain, and is achieved when a child domain starts and contacts is parent domain. Since the domain doesn't exist yet, the domain name cannot be used. The database itself, however, can be referred to using a **database address**.

A database address is made up of two components, the host name and the database tag, separated by a slash.

In this manual, database addresses are represented using ***bold italic*** type with the slash separator, "/". For example, the local domain on host *alpha* is served from a database with the following address:

***alpha/local***

Using the recommended convention explained above, the domain **sysdev** on host *alpha* is served from a database with the address:

***alpha/sysdev_db***

**Daemon Processes**

In order for a domain to access information in a database, a **process** must be started. The controlling NetInfo process, the **nibindd** daemon, is usually started at boot time, and must be running for NetInfo to operate.

Once **nibindd** is running, it starts up a **netinfod** daemon for each existing database in the `/etc/netinfo` directory on the host on which the database is stored. The **daemon** processes are owned by root and run in the background.

As domains are created, new instances of the **netinfod** daemon are started. These processes stop when a domain is removed, or the **nibindd** daemon is stopped.

## Internal Structure of the Database

The database is organised internally into **directories** and **subdirectories**. The NetInfo directory structure is very similar to the UNIX file system, but the two should not be confused.

In NetInfo, information associated with a directory is not stored in files, but in either of two places: in subdirectories, or in the directory itself in the form of **properties**.

A property is made up of two parts: the **property key** which can be thought of as the property's name, and the **property value**.

A database can have any number of subdirectories. Each subdirectory can have zero or more properties, and each property can have zero or more values.

The NetInfo database begins at the top level with a **root directory**, represented by a slash ("**/**").

Some subdirectories and properties are compulsory in order to administer NetInfo. These are explained in detail as required in *Chapter 5 - Using NetInfo*.

*Example:*

*Directories and Properties*

Information about a user account is stored in a database directory called **/users**. For each user who access to the resources of the domain, there must be a subdirectory with the name of that user.

The ACME company has three staff members in the Development section, (login id's *chris, bing*, and *jo*), who must be registered as users in the **sysdev** domain. The database which serves this domain, ***sysdev_db***, must have the following subdirectories:

**/users/chris**

**/users/bing**

**/users/jo**

The properties of a **/users** subdirectory correspond to the fields in the /etc/passwd file. These subdirectories have the following properties and values:

| **Directory:** | **/users/chris** | **/users/bing** | **/users/jo** |
|---|---|---|---|
| Property | | | |
| *name* | chris | bing | jo |
| *passwd* | AX#1@2Tf | DS#155Tf | PUI87#22 |
| *uid* | 101 | 102 | 103 |
| *gid* | 10 | 10 | 10 |
| *real name* | Chris Smith | Bing Lee | Jo Smith |
| *home directory* | /usr/staff/chris | /usr/staff/bing | /usr/staff/jo |
| *shell* | /bin/csh | /bin/csh | /bin/csh |

Each user on the system would have a similar database entry. Other administrative information, such as group data, is stored in a database directory in a similar way.

*Example:*

*The "serves" property*

Binding is controlled by the "serves" property in the **/machines** subdirectories in the database. A subdirectory must exist in **/machines** for every host that serves any of the following:

- its parent domain,
- its own (self) domain,
- or one of its child domains.

In the ACME domain structure, the **sysdev** domain is served from the host *alpha*. Its parent, "**/**", is also served from *alpha*, from the database ***network***. It has two child domains, **alpha** (the local domain), on host *alpha*, and **bravo** (the local domain), on host *bravo*.

As can be seen, there are two hosts that serve **sysdev**, its parent, or its child domains: therefore there must be two **/machines** subdirectories:

**/machines/alpha**

**/machines/bravo**

These subdirectories must have at least three properties each:

- a name (the value is the name of the subdirectory)
- Internet address
- a "serves" property, with one or more values showing the database address of the databases from which its parent, its self and its child domain(s) are served.

In the **/machines/alpha** subdirectory, three value entries must exist for the "serves" property to specify the database from which itself is served, its parent, and its one child on *alpha*.

**/machines/alpha:**

| Property | Value |
|---|---|
| name | alpha |
| ip_address | 192.42.172.1 |
| serves | ./sysdev_db, ../network, alpha/local |

In the **/machines/bravo** subdirectory, only one "serves" value is required, to specify the child located on *bravo*.

**/machines/bravo:**

| Property | Value |
|---|---|
| name | bravo |
| ip_address | 192.42.172.2 |
| serves | bravo/local |

The above example shows the structure of the **sysdev** domain only. Its parent, the root domain, must have a reference that **sysdev** is its child. Also, the local bravo domain, must have a reference to its parent.

## Changing Database Information

Normally, only **root** (superuser) has write access to information in the NetInfo databases. Users can be granted permission to change the values of specific properties or variables by applying the "_writers" property. In order to change their own password, users must have "_writers" access to their own password property.

### "_writers" Property

The "_writers" property has one or more values: The values are login names of those users who are able to *write* to the property or directory. There are two forms:

| | |
|---|---|
| "_writers" | This form allows users named in the list of values to write to all properties in the directory. |
| "_writers_*propkey*" | This form allows users named in the list of values to write to the specified property only. |

The value of the "_writers" property (list of usernames) can take the value "*". This is a wildcard meaning all users of the domain. The wildcard specification is generally used when defining access to printers in a domain.

## Master Servers and Clones

So far, the databases that have been discussed are **master servers**. A clone server can be set up to mirror the information of a master server. The name server in NetInfo terminology refers to the database that serves a domain.

Clones can be set up for two reasons: **reliability** and **load balancing**. It is up to the administrator to decide if there are sufficient resources available to establish a clone. Resources required include a separate host with sufficient disk space, swap space, and memory.

If an organisation requires uninterrupted NetInfo service, a clone should be established. A clone server is an exact copy of a master server. If a host is down, or a domain cannot be connected for any reason, the clone database can be used instead of the inaccessible master database.

Clone databases, however, cannot be modified if the master server is down. This ensures that there is only one source of domain information. Also, clones cannot be created on the same host as the master database they copy.

Clones can also be used to establish the best load balance of a network. A host will search for information locally before attempting to search the network. If a clone is established on a local host, it will obtain information from the clone rather than the master database elsewhere in the network.

When the **netinfod** daemon for each database is started, it first checks to see if the database is a clone, or if it has any clones. For each master database, if any clones are found, the daemon sets up tasks to ensure that any changes that are made to the master are also made to the clone.

The root directory of every database has a **"master"** property. The value of this property specifies the database address of the master database. If this property refers to itself, then the database is a master; if it refers to another database, then it must be a clone.

When a master database is created, this value is set to the database address of the new database. When a clone is created, the entire master database is copied, and so the value of the "master" property points to the master database.

# Chapter 3

# NetInfo Design

Before creating a NetInfo network, you should consider how the resources of your system have to be shared amongst the users. This means that you must consider how your domain hierarchy should be structured.

Just as hierarchical directories enhance the organisation of disk files, hierarchical domains enhance the organisation of network data. A domain can hold information about a single host, about a department, or information about an entire organisation.

Networked systems allow computer resources to be shared amongst users. Through careful design, the administrator can:

- create the correct load balancing of systems
- enhance system performance
- provide access and restrictions to resources such as printers and tape drives
- provide users with access to required information and applications.

This chapter gives the administrator an overview of available networking alternatives. It does not offer specific solutions to network configuration and equipment requirements. It is addressed to those administrators who have not already designed their network. If you have used NetInfo before, you may wish to skip this chapter.

# Dividing a Network into Domains

When designing a network, the administrator must decide what resources are to be shared and with whom, and which resources must be kept private. By organising the network into conceptual domains, resources can be shared or restricted as required.

NetInfo domains imply a hierarchical structure. This type of structure allows resources to be organised so that access permissions can be easily granted or denied to groups or individual users. The domain structure takes the form of an inverted tree. Often, the domains will be set up to mirror the hierarchical structure of an organisation.

There are two issues which can affect how a network is structured: location of the systems, and the resources that have to be shared.

### Location of Systems

Some systems are located in different cities, states, or even countries. A separate domain is often created for a system at a location. This system may include several machines, and is usually managed by the one systems administrator. This domain can be further divided into lower level domains if required.

### Sharing Resources

If a group of users need to share particular resources, such as printers and tape drives, or applications and information, they are often grouped together into a domain. Resources can also be restricted in the same way.

**Two-Level Domain Hierarchy**

A two-level structure will suit many organisations. At the top is the root domain. The next level contains the local machine domains. A local domain of a host cannot have child domains below it. Two-levels is the least number of levels that are of practical use.

*Example: Two-Level*

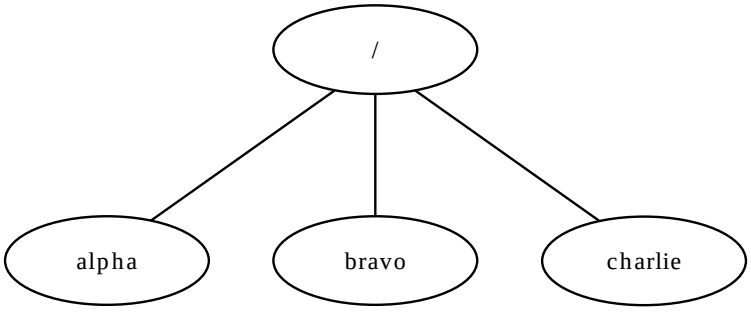



*Figure 3.1 - Two-Level Domain*

This type of structure is only useful for small companies with a few machines. Host access is commonly restricted to one host or all. It is difficult, though possible, to organise access on only some of the hosts. A user must have an account on each machine they wish to access.

**Three-Level Domain Hierarchy**

Larger organisations may want to divide their network administration along the same structure as their organisation. This structure will need at least three levels.

*Example: Three-Level*

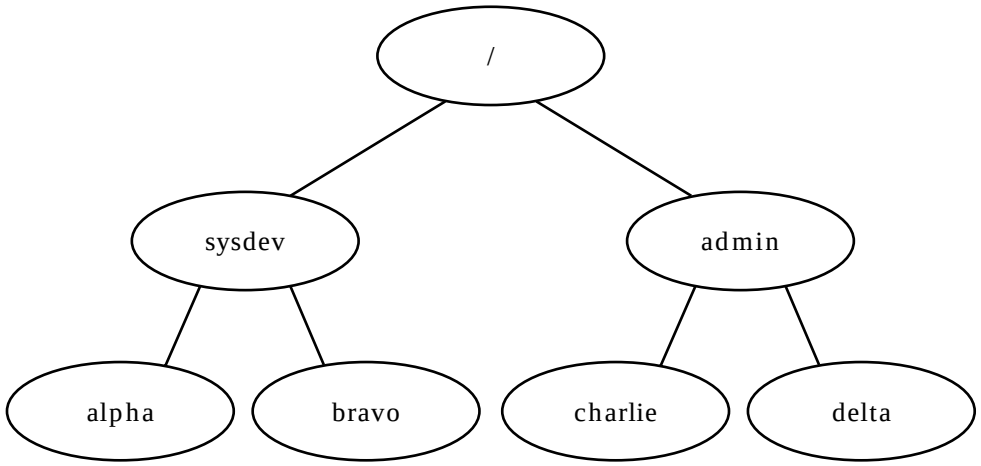



*Figure 3.2 - Three-Level Domain*

Users will have accounts in a particular departmental domain, and access to the machines in that domain only. Managers, or other users who require access to all machines, must be registered in the root domain.

**Multi-Level Domain Hierarchy**

There is no limit to the depth the hierarchy can take. The structure depends entirely on how the machines and users are organised. In practice, however, it is advisable to keep the hierarchy broad and shallow.

Example:

Multi-Level domains

Consider three departments at a large university: Computing, Engineering, and Mathematics. Each department has connected their own machines into local networks. Users within each department share information, but not much information is shared between the different departments. They all wish to be connected in order to exchange electronic mail and news as necessary, but the machines of each department must be kept separate.

Currently, each department has a similar domain hierarchy. The following example shows the Computer Science Department:

Example:

Multi-Level domains

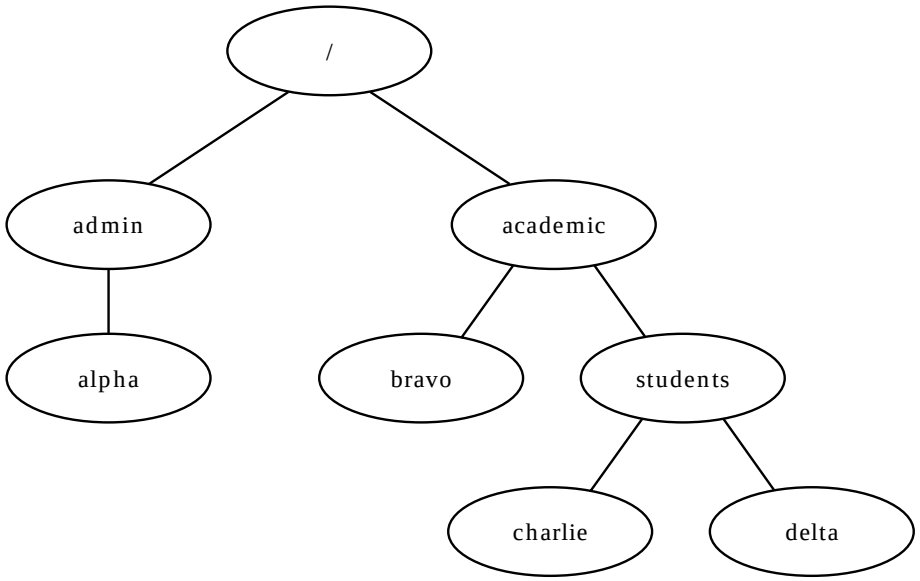



*Figure 3.3 - Multi-Level Domain*

All Computer Science students have login names registered in the **students** domain only. They have access to machines *charlie* and *delta*. Academic staff are registered in **academic**. The have exclusive access to the machine *bravo*, but also, they are able to access all machines in the child domain, **students**, that is, *charlie* and *delta*. Administrative staff have access to machine *alpha* only.

If any users need access to both the **admin** and **academic** machines, they would be registered as users in the **root** domain.

The Math and Engineering Departments are structured in a similar way, however, they use a different set of machines. Both departments have a **root** domain, and two subdomains, **admin** and **academic**, but the local domains are named after the machines in each department.

Assume the one administrator is managing all three departments. They should be organised into three distinct domains, each one level below the new **root** domain.

Each of the old **root** domains must be renamed to reflect the fact they are no longer at the top level: to something like **compute, math,** and **engine**. The rest of the domain hierarchies below these levels will remain the same.

Users within each department can only login to their own machines, but can send mail to all others.

Any user registered in the new **root** domain, such as the superuser, has access to all machines.

*Example:*
*Multi-Level domains*

The domain hierarchy should be set up as follows:

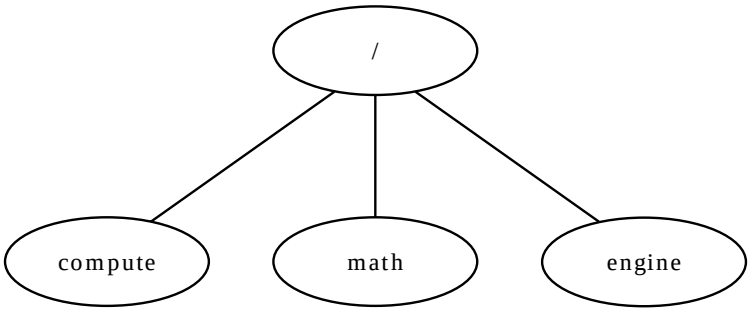



*Figure 3.4 - Multi-Level Domain*

## Selecting Hosts

The host which serves the **root** domain should be highly available, unless clones have been established. It must have a substantial amount of memory, swap space, and disk space. Administrators should select "server" style machines, i.e., those that are configured for high compute/transaction throughput, preferably a high performance disk server.

No domain can be bound into the hierarchy until it has found its parent. The host which serves a parent domain(s) must be alive before its child domains can start-up.

### Setting up Clones

A clone should be established if the organisation requires uninterrupted NetInfo service and the resources are available. The resources required include a host with enough memory and disk space to serve the cloned databases.

Any database, other than those serving the local domain, can be cloned. A clone must exist on a different host than its master.

The first database you should consider cloning is the ***network*** database which serves the root domain. This database is the parent of the hierarchy. If it is not alive, no domain can be bound into the hierarchy.

#### Reliability

If a host is down, or a domain cannot be connected for any reason, the clone database is used instead of the inaccessible master database. This ensures that the NetInfo service is not interrupted.

If the network is not connected, users can only login to hosts if they are known to the "local" database. If NetInfo is not running at all, they can connect to a host if they have a valid entry in the flat password file.

Clone databases, however, cannot be modified if the master server is down. This ensures that there is only one source of domain information.

#### Load Balancing

Clones can also be used to establish the best load balance of a network. A host will search for information locally before attempting to search the network. If a clone is established on a local host, it will obtain information from the clone rather than the master database elsewhere in the network.

## Domain Names

The database tag for a domain must be unique on a machine. When a domain is created, no matter at what level it will be bound into the hierarchy, a data-base file is created and stored in `/etc/netinfo`. All database files are stored at the same level, and so the names must be unique on a particular host.

**Unique Names**

Extending the multi-level example above, the Math and Engineering depart-ments may also wish to divide their domains further into an **admin** and **academic** domain structure. As each department is organised on a different set of machines, the same names can be chosen for the lower-level domains. At each level within a domain, and on each host, the names must be unique.

## Managing Users

To login to any machine, a user must have an account. NetInfo determines user access by looking up information in the domain hierarchy rather than using the flat files.

When a user tries to login to a particular host, NetInfo first looks in the ***local*** database for that host. If the username is found, the ordinary login process is continued. If not, NetInfo searches up through the hierarchy until it finds the user account information it wants, or it reaches the top level.

**User Access**

Because of this structure, a user need only be registered in one domain: this must be the highest level domain to which they have access. Access to a domain implies access to all its child domains.

In the three-level example above, users in the system development department should have user accounts in the **sysdev** domain. This automatically gives them access to machines *alpha* and *bravo*.

**Superuser**

The superuser should have an account in each domain, particularly the root domain and the local domains for each machine. If, for some reason, a machine is not connected to the network, NetInfo can still run, allowing the superuser to login to any machine.

**Local Users**

In a two-level hierarchy, users who have access to a machine must have accounts in the local database. If a machine is not connected to the network, these users can still login to their own machine.

In multi-level hierarchies, users may have their accounts in a higher level domain. If a machine becomes disconnected from the network, that higher level domain may be inaccessible. This means that some users cannot login to a particular machine.

The administrator can arrange to copy the password information from high-level domains into the local domains on each relevant machine to overcome this problem. Users must be aware, though, if they change their password, it will not propagate up the domain hierarchy.

## Updating the Flat Files

Flat files on each machine can be kept up-to-date using the NetInfo tools, **nidump** and **niload**, to copy information between the specified database and the file.

It is recommended that systems administrators update the flat files on a regular basis. The flat files are consulted by the system when it is booted, and also if neither NetInfo nor NIS are running. This is easily managed using the standard UNIX **cron** facility and the NetInfo tools. All tools are explained in the *Reference* chapter.

# Chapter 4

# Configuration - Quick Start

This chapter describes an alternative NetInfo installation procedure for those sites with existing NetInfo networks. This installation is somewhat simpler than the full manual configuration process described in the next chapter, *Manual Configuration*, as it makes use of an automated installation script, `quick_start`, provided as part of the NetInfo software product.

If you do not already have a NetInfo domain heirarchy, and wish to create a new root domain server, a quick_start_root script is also provided.

## Installation

You should have already installed the NetInfo software onto your system as per Chapter 1 of the *Installation guide* for your specific platform.

This section describes how to set up your server with a local NetInfo database. You should already have a NetInfo server installation with a root domain up and running before undertaking this installation. In addition, you should have noted the hostname and the IP address of the host which is to provide your **parent** NetInfo domain, as this information will be required during the Quick Start installation. You will also need to know the IP address of the machine onto which you are installing the system.

If you do not already have an NetInfo network up and running, you can use the quick_start_root script instead. However, it is recommended that you read the *Manual Configuration* chapter as well to fully understand what is involved in establishing a NetInfo network.

You will be prompted for the **root** password of the host that is serving the parent domain during installation. Please ensure that you either know this password or have someone that does know it available during the installation process.

## Backup

It is recommended that the existing system configuration files are backed up before continuing with the installation (`/etc/passwd`, etc.). NetInfo does not actually change any of the information in these files, but the utilities that load and unload existing information into NetInfo databases can affect these flat files.

# Quick Start Install

The following example shows how to connect a new server, *bravo*, to an existing NetInfo network. The server will need a ***local*** database. It is connected to the existing hierarchy as a child of another domain, referred to as its **parent domain**. This domain can be any existing domain, not necessarily the root domain. For this example, we will install the server as a child of the root domain, which is served by the host *alpha*.

The Quick Start install is automated by the script **quick_start** provided as part of the NetInfo software product.

Example

The new host, *bravo*, must have a ***local*** database. Its parent is the **root** domain on host *alpha*.

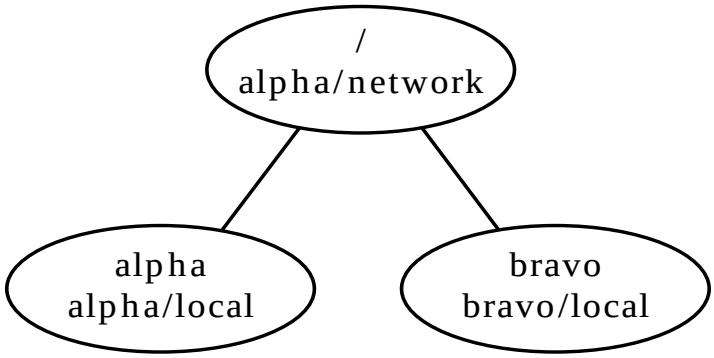



*Figure 2.1 - Install new server, bravo*

## Configuration Steps

The configuration steps for installing a server are:

1. Install the NetInfo software as described in the *Installation Guide* for your specific platform.
2. Become superuser (root).
3. Run the **quick_start** install script.
4. Test the system.
5. Reboot and test.

**1: Load the NetInfo Software**

You must first complete the software installation procedures documented in the *Installation Guide* for your specific platform.

**2: Become the super user**

Become the superuser (root) and change to the working directory.

**3: Run the quick_start installation script**

The **quick_start** script, provided as part of the NetInfo software product, automates many of the steps required to create a NetInfo ***local*** database.

In particular, it automatically creates the `/etc/netinfo` directory and installs a new ***local*** database `local.nidb` into this directory.

It starts the NetInfo daemon processes required to use the NetInfo system.

It also binds the new database into your existing NetInfo hierarchy and can **optionally** load the contents of the system configuration files (`/etc/hosts`, `/etc/passwd`, etc.) into the local database for you.

**quick_start** requires you to enter the **root** password of the host *alpha* during installation. This is required to allow changes to be made to the **parent** NetInfo database, running on host *alpha*. You may be asked for this password more than once.

**quick_start_root** does not require a parent NetInfo database, since it is establishing the new root database. Ignore any references to parentdatabase in the following if you are running `quick_start_root`.

*quick_start*

The **quick_start** script requires four parameters on the command line:

- the domain name that the new local domain is to be known by
- the **database address** of the parent domain
- the IP addresses of the server
- the IP addresses of the host that serves the parent domain

For example, to set up the new server *bravo* we would invoke quick_start as follows:

```
# quick_start bravo 192.42.172.2 alpha/network 192.42.172.1
```

where *bravo* is to be the domain name of the new database.

The parent domain, /, is served from the database *network* on host *alpha*.

The first IP address provided is the address of the new server, *bravo*.

The second IP address provided is the address of the host, *alpha*.

*loading configuration data*

Following the creation of the local database and binding it into the NetInfo hierarchy, you will be prompted for the optional loading of your local configuration files (`/etc/hosts`, `/etc/passwd` etc.).

```
Do you wish to load local configuration data into the local
NetInfo database [no]?
```

Answering '`yes`' to this question will cause **quick_start** to load the contents of the following files into the local database on your server:

```
/etc/hosts
/etc/passwd
/etc/group
/etc/rpc
/etc/services
/etc/protocols
/etc/aliases
/etc/networks
/etc/bootptab
/etc/bootparams
/etc/exports
/etc/fstab
```

If you choose not to load these files, then your server will take its configuration information from the parent NetInfo domain.

- Check that the directory and database file are created:

  ```
  # ls -al /etc/netinfo/local.nidb/Collection
  ```

- Check that the **netinfod** process is started. Use **nips** if it is installed, which should display:

  ```
  # nips
  netinfod local
  ```

- Check the default information automatically loaded into the database. Use the **niutil** tool to display a list of directories and properties:

  ```
  # niutil -list -t bravo/local /
      1    machines
      2    users
      3    groups
      4    networks
      5    protocols
      6    rpcs
      7    aliases
           ...
  ```

This command requests a list of subdirectories of the root directory, "/", in the database specified using the tag ***bravo/local***.

```
# niutil -list -t bravo/local /machines
    2    bravo
    9    alpha
```

This command requests a list of subdirectories of the directory, **/machines**, in the database ***bravo/local***.

```
# niutil -read -t bravo/local /machines/bravo
    name: bravo
    ip_address: 192.42.172.2
    serves: ./local
```

This command requests a list of properties and values of the directory, **/machines/bravo**, in the database ***bravo/local***. The result shows that the domain, self (.), is served from database ***local***.

Check that the parent binding has also been correctly installed.

```
# niutil -list -t alpha/network /machines
    ...
    2    bravo
    ...
```

This command requests a list of subdirectories of the directory, **/machines**, in the database ***alpha/network***. The result shows that the subdirectory, **/machines/bravo** does indeed exist.

```
# niutil -read -t alpha/network /machines/bravo
    name: bravo
    ip_address: 192.42.172.2
    serves: bravo/local
```

This command requests a list of properties and values of the directory, **/machines/bravo**, in the database ***alpha/network***. The result shows that the domain, **bravo**, is served from database ***local*** as desired.

*Naming the child domain*

The property value in the parent database:

```
serves "bravo/local"
```

is where the child domain gets its name. This property states that a domain with the name **bravo** (relative to the current domain) is served from a database with tag ***local***. In our example, the full pathname of this domain is **/bravo**.

**4: Test the Network**

The domain hierarchy can be tested by interrogating the database using the domain name rather than the address.

On *bravo* there is one database:

***local***, which serves the domain **/bravo**.

On *alpha*, there are two databases:

***local***, which serves **/alpha**, and
***network*** which serves the root domain, /.

List the properties which describe the machines in each of these databases:

```
# niutil -read /bravo /machines/bravo
# niutil -read /bravo /machines/alpha
# niutil -read / /machines/bravo
# niutil -read / /machines/alpha
# niutil -read /alpha /machines/alpha
```

If the binding has not worked, an error message is displayed, saying that the database served by the specified domain cannot be opened.

**5: Reboot and Test**

At this stage, reboot the system and check that all the daemons started successfully.

Check that the binding is correct by interrogating the database using the domain names instead of the database tags.

As a final confidence check, try a common command, such as:

```
# /bin/ls -lg
```

This command must look up user id and group id information, using NetInfo.

Check that the results are what you would expect.

This completes your NetInfo installation for the local server.

# Chapter 5

# Manual Configuration

Your NetInfo package includes an *Installation Guide* that describes how to unload the software from the distribution media, and what files should exist. The superuser, **root**, must have access to the programs. Many of the NetInfo functions require the user to be logged in as **root**.

The *Installation Guide* also documents the procedure you should follow to install the NetInfo software for your system. These procedures should be carried out **prior** to following the installation steps documented in this chapter.

This section describes how to set up the initial host, with a local database and a root domain. The NetInfo software must be installed on all hosts in the network, and each must have a local database. Only one root domain should exist., although advanced NetInfo administrators can create multiple seperate NetInfo heirarchies as well.

## NetInfo Programs

The NetInfo system is comprised of the NetInfo daemons and several utility programs used to access the database. Some of these programs are compulsory and must be located in specific directories. Other utilities supplied can be located in any utilities directory chosen by the administrator. This manual assumes utilities are stored in `/usr/bin`.

| Program | Notes |
|---|---|
| *Programs - Compulsory Location* | The following programs **must** be installed in `/usr/etc` |
| **nibindd** | NetInfo daemon - must be running on all hosts. |
| **niypd** | NIS emulation lookup daemon - must run in conjunction with **nibindd**. |
| **netinfod** | For each database in the `/etc/netinfo` directory, an instance of the **netinfod** daemon is started to access the specified database. |
| **nidomain** | Used to create and destroy databases. When creating a database, it also starts a **netinfod** process to access the database. |
| *Utilities - Can be stored anywhere* | Can be stored anywhere, although this manual assumes they are in `/usr/bin` |
| **niutil** | Used to list and maintain the contents of a database. |
| **niload** | Used to add information to the database from standard input. |
| **nidump** | Used to dump information from the database to standard output. |
| **niwhich** | Shows which NetInfo databases are served from which hosts. This utility also shows the Internet address of the specified host. |
| **nipasswd** | Allows users to change their NetInfo password. |

| | |
|---|---|
| **nigrep** | Search using regular expressions in a NetInfo database. |
| **nifind** | Search for directories with given key/value pairs. |
| **nireport** | Produces a tab seperated report using multiple search keys. |

## Backup

It is recommended that the existing system configuration files are backed up before continuing with the installation. (`/etc/passwd`, etc.) NetInfo does not actually change any of the information in these files, but the utilities that load and unload existing information into NetInfo databases can affect these flat files.

## Design

This chapter assumes you have already designed your network. You should know how your available machines are arranged in the network hierarchy, and which users should have access to which resources. Initially, you should have decided which host should serve the root domain. This host must be a highly available host with sufficient memory and disk space. Clone servers can be added at any time.

## Install *NetInfo* on the First Host

NetInfo is installed on a single machine in several stages. The system programs must be loaded on each machine that is part of the NetInfo network. Administrators may find it easiest to install the programs onto one host and then copy them across to all others. Databases, however, should not be copied.

Some NetInfo programs must be located in specific directories, and the directory structure for the NetInfo databases must be created.

Every host must have a ***local*** database. Only one ***network*** database should exist in the network.

*Example:*

*Install NetInfo on a single host*

In order to illustrate the NetInfo start-up procedure, the following example network is installed. Both ***local*** and ***network*** databases are installed on this host, to create a two-level hierarchy.

This network initially consists of one host, *alpha*. A subsequent example explains how to connect other hosts.

The root domain is served by the ***network*** database on host *alpha*. This host must also have a ***local*** database, serving the local domain with name **alpha**.

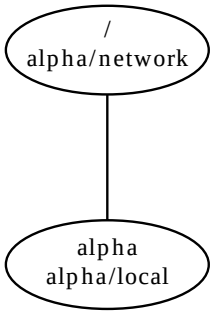



*Figure 4.1 - Install 2-Level Domain Hierarchy*

**Configuration Steps**

1. Load the software onto your system – refer *Installation Guide.*
2. Create directory structure for the NetInfo databases.
3. Start the NetInfo daemon.
4. Create ***local*** database.
5. Create ***network*** database.
6. Bind ***local*** and ***network*** into domain hierarchy.
7. Test the network.
8. Load system information.
9. Reboot and test.

### 1: Load the NetInfo Software

The NetInfo programs, as listed earlier in this chapter, must be installed on each host in the network that is to run NetInfo. Some of these programs are compulsory and must be located in specific directories. Other utilities supplied can be located in any utilities directory chosen by the administrator. The *Installation Guide* that came with your NetInfo package documents the procedure you should follow to load the software for your particular system.

The following programs must be installed in `/usr/etc` on *alpha*:

**nibindd** **niypd** **netinfod** **nidomain**

The following utilities are used in the examples throughout the manual. They can be installed anywhere at the discretion of the administrator. For this example, they will be installed in `/usr/bin`.

**niutil** **niload** **nidump** **nipasswd** **niwhich**

### 2: Create the NetInfo Database Directory Structure

NetInfo expects to find its databases in a directory called `/etc/netinfo`. This directory must exist before a domain can be created on any host.

Check that this directory exists before continuing. If not, it must be created and owned by **root**.

e.g. `mkdir /etc/netinfo`

### 3: Start the NetInfo daemon

Two daemon processes must be running on each host in order to operate using the NetInfo database: **nibindd** and **niypd**. These processes run in the background.

Initially, only **nibindd** need be started: **niypd** will not work until after the configuration information has been loaded. If you are following these instructions, **niypd** is not started until the system is rebooted.

Normally, **nibindd** must be started before **niypd**. These two processes are started at boot time. Detailed information on these two processes can be found in the *Reference* chapter.

Login as the superuser and enter the following command:

```
/usr/etc/nibindd &
```

**4: Create the local database**

The ***local*** database must exist on each host, and its name is fixed as **local**.

Login as **root** to *alpha* and use the **nidomain** tool to create the database:

```
nidomain -m local
```

- Check that the directory is created:

  ```
  /etc/netinfo/local.nidb
  ```

  This directory should contain one file:

  ```
  collection
  ```

- Check that the **netinfod** process is started . Use **nips** if it is loaded, which should display:

  ```
  netinfod local
  ```

- Check the default information automatically loaded into the database. Use the **niutil** tool to display a list of directories and properties:

  ```
  niutil -list -t alpha/local /
  ```

  ```
  1    machines
  ```

  This command requests a list of subdirectories of the root directory, "**/**", in the database specified using the tag ***alpha/local***. The result shows that there is only one subdirectory, **/machines**, which is assigned directory number 1.

  ```
  niutil -list -t alpha/local /machines
  ```

  ```
  2    alpha
  ```

  This command requests a list of subdirectories of the directory, **/machines**, in the database ***alpha/local***. The result shows that there is only one subdirectory, **/machines/alpha**.

  ```
  niutil -read -t alpha/local /machines/alpha
  ```

  ```
  name: alpha
  ip_address: 192.42.172.1
  serves: ./local
  ```

  This command requests a list of properties and values of the directory, **/machines/alpha**, in the database ***alpha/local***. The result shows that the domain, self, is served from database ***local***.

*Create "root" user*

Create a root user in the database. This user is the only one able to make changes to the database and must exist.

```
niutil -create -t alpha/local /users
niutil -create -t alpha/local /users/root
```

These commands first create a **/users** directory, then a subdirectory for the root user.

Now create properties and values corresponding to the password file. The "name" property is set to the name of the directory. The user id and group id are compulsory. The password **must** be set to null initially.

```
niutil -createprop -t alpha/local /users/root uid 0
niutil -createprop -t alpha/local /users/root gid 1
```

```
niutil -createprop -t alpha/local /users/root passwd ""
```

**5: Create the network database**

The ***network*** database serves the root domain. Its name is fixed as **network**.

Use the **nidomain** tool to create the database:

```
nidomain -m network
```

- Check that the directory and file is created:

  ```
  /etc/netinfo/network.nidb/collection
  ```

- Check that the `netinfod` process is started:

  ```
  netinfod network
  ```

- Check the default information automatically loaded into the database. This information is similar to that created for the ***local*** database.

  ```
  niutil -list -t alpha/network /
  ```

  ```
  1    machines
  ```

  ```
  niutil -read -t alpha/network /machines/alpha
  ```

  ```
  name: alpha
  ip_address: 192.42.172.1
  serves: ./network
  ```

  This shows that the domain, self, is served from database ***network***.

*Create "root" user*

Create a root user in the database. This user is the only one able to make changes to the database and must exist.

```
niutil -create -t alpha/network /users
niutil -create -t alpha/network /users/root
niutil -createprop -t alpha/network /users/root uid 0
niutil -createprop -t alpha/network /users/root gid 1
niutil -createprop -t alpha/network /users/root passwd ""
```

**6: Bind local and network**

Binding is a two-way process that is achieved by adding *serves* properties to the appropriate databases. Both the uplink (child specifying parent) and the downlink (parent specifying child) must be entered.

- The **local** domain (**local**) must specify its parent.
- The **root** domain (**network**) must specify its child.

*Specify parent of **local***

The parent of **local**, the **root** domain, is served by the ***network*** database on host *alpha*. Add a value to the *serves* property of the **`/machines/alpha`** directory of the database ***alpha/local*** specifying that the parent "**..**" is served from database ***network***.

```
niutil -addval -t alpha/local /machines/alpha
    serves "../network"
```

*Specify child of **network***

The child of **network**, is served by the **local** database on host *alpha*. Add a value to the *serves* property of the **`/machines/alpha`** directory of the database with the tag ***alpha/network*** specifying that the child is served from database **local**.

```
niutil -addval -t alpha/network /machines/alpha
    serves "alpha/local"
```

*Naming the child domain*

The property value in the parent database:

```
serves "alpha/local"
```

is where the child domain (**local** in this case) gets its name. This property is saying a domain with the name **alpha** (relative to the current domain) is served from a database with tag ***local***. The full pathname of this domain is **/alpha**.

**7: Test the Network**

Since a domain does not exist until it is bound into the hierarchy, it cannot be referred to by its domain name, only its database address. The previous interrogations of the database properties all made requests using the database address (**-t** option of **niutil**). The domain hierarchy can be tested by interrogating the database using the domain name rather than the address.

The ***local*** database serves a domain on host alpha. The full name of this domain is **/alpha**. The ***network*** database serves the root domain, called **/**.

List the properties which describe the machines in each of these databases:

```
niutil -read /alpha /machines/alpha
niutil -read / /machines/alpha
```

If the binding has not worked, an error message is displayed, saying that the database served by the specified domain cannot be opened.

**8: Load System Information**

Information that is normally stored in the configuration files must be loaded into the NetInfo network before the configuration data can be accessed. This information need not be stored in every database. A local domain inherits information from its parent; the local database is first consulted, but if the required information is not there, NetInfo searches up the hierarchy until it is found.

Some administrators may choose to load configuration information into the root domain; in this case, the information is stored only once. **niload** can be used to transfer this information according to the specified format.

**Loading information from flat files**

This example shows the configuration information loaded into the root domain from the flat files, using the domain name, "/". The commands can be entered in any order.

```
niload -v passwd /  <  /etc/passwd
niload -v hosts /  <  /etc/hosts
niload -v services /  <  /etc/services
niload -v protocols /  <  /etc/protocols
niload -v group /  <  /etc/group
niload -v rpc /  <  /etc/rpc
niload -v fstab /  <  /etc/fstab
```

**niload** adds information to the specified database, in this case, the database that serves the root domain, "**/**", called ***network***. If an entry exists in both the database and input file, the database updates its existing information with the data from the input file.

The appropriate information could also be entered into the local database using **niutil**, creating sub-directories and entering each property one line at a time, though this would be very time consuming.

### Loading information from NIS

If you are running NIS, the system information can still be loaded using `niload`, but the input should be extracted from NIS as follows:

```
ypcat passwd | niload -v passwd /
ypcat hosts | niload -v hosts /
```

and so on for each NIS map.

### Directory Names

Password information is loaded into the `/users` directory. If the appropriate directory does not exist, **niload** will create it.

Administrators should be aware that the internal directory names are not necessarily the same as the flat file names. The section on *Host Configuration Information* in *Chapter 5: Using NetInfo* explains the information loaded by **niload**.

## 9: Reboot and Test

At this stage, reboot the system and check that both the daemons started successfully.

Check that the binding is correct by interrogating the database using the domain names instead of the database tags.

Try a common command, such as:

```
/bin/ls -lg
```

Check that the results are what you would expect.

This command must look up user id and group id information, using NetInfo.

## Install *NetInfo* on Other Hosts

Most networks will have more than one host. The following example shows how to connect a second host to the network. The second host must have a ***local*** database. It is connected to the existing two-level hierarchy as a child of the root domain.

Most of the steps here are the same as in the previous example. There is no need to create another **root** domain.

Example:

Install NetInfo on a second host

The new host, *bravo,* must have a ***local*** database. Its parent is the **root** domain on host *alpha*.

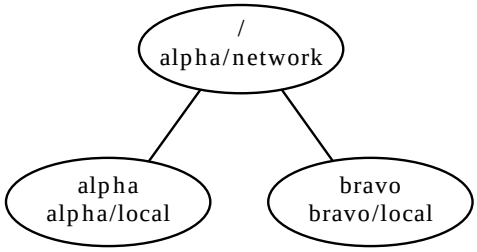



*Figure 4.3 - Install 2nd Host*

**Configuration Steps**

The configuration steps for installing a second host are:

1. Load the software onto your system – refer *Installation Guide.*
2. Create directory structure for the NetInfo databases.
3. Start the NetInfo daemon.
4. Create ***local*** database.
5. Bind ***local*** into existing domain hierarchy.
6. Test the network.
7. Load system information.
8. Reboot and test.

**1: Load the NetInfo Software**

The NetInfo programs must be installed on *every* host that is part of the NetInfo network. See Step 1 of the previous example.

**2: Create the Directory Structure**

NetInfo expects to find its databases in a directory called `/etc/netinfo`. This directory must exist before a domain can be created on any host.

Check that this directory exists before continuing. If not, it must be created and owned by **root**.

```
mkdir /etc/netinfo
```

**3: Start the NetInfo daemon**

Two daemon processes must be running on each host in order to operate using the NetInfo database: **nibindd** and **niypd**. These processes should run in the background.

Initially, only **nibindd** need be started: **niypd** will not work until after the configuration information has been loaded. If you are following these instructions, **niypd** is not started until the system is rebooted.

Normally, **nibindd** must be started before **niypd**. These two processes are started at boot time. Detailed information on these two processes can be found in the *Reference* chapter.

Login as the superuser on host *bravo* and enter the following command:

```
/usr/etc/nibindd &
```

**4: Create the local database**

The ***local*** database must exist on each host, and its name is fixed as **local**.

Login as **root** on *bravo* and use the **nidomain** tool to create the database:

```
nidomain -m local
```

- Check that the directory and database file are created:

```
/etc/netinfo/local.nidb
collection
```

- Check that the **netinfod** process is started . Use **nips** if it is loaded, which should display:

```
netinfod local
```

- Check the default information automatically loaded into the database. Use the **niutil** tool to display a list of directories and properties:

```
niutil -list -t bravo/local /
```

```
1    machines
```

This command requests a list of subdirectories of the root directory, "**/**", in the database specified using the tag ***bravo/local***. The result shows that there is only one subdirectory, **/machines**, which is assigned directory number 1.

```
niutil -list -t bravo/local /machines
```

```
2    bravo
```

This command requests a list of subdirectories of the directory, **/machines**, in the database ***bravo/local***. The result shows that there is only one subdirectory, **/machines/bravo**.

```
niutil -read -t bravo/local /machines/bravo
```

```
name: bravo
ip_address: 192.42.172.2
serves: ./local
```

This command requests a list of properties and values of the directory, **/machines/bravo**, in the database ***bravo/local***. The result shows that the domain, self, is served from database ***local***.

*Create "root" user*

Create a root user in the database. This user is the only one able to make changes to the database and must exist.

```
niutil -create -t bravo/local /users
niutil -create -t bravo/local /users/root

niutil -createprop -t bravo/local /users/root uid 0
niutil -createprop -t bravo/local /users/root gid 1
niutil -createprop -t bravo/local /users/root passwd ""
```

**5: Bind local and network**

Binding is a two-way process that is achieved by adding *serves* properties to the appropriate databases. Both the uplink (child specifying parent) and the downlink (parent specifying child) must be entered.

• **local** (the child) must specify its parent.

• The **root** domain (**network**) must specify its child.

As the parent and child domains are located on different hosts, you must ensure that you are updating the correct database.

The parent of **local**, the **root** domain, is served by the ***network*** database on host *alpha*. The ***local*** database on *bravo* does not have a **/machines/alpha** directory, and the ***network*** database on *alpha* does not have a **/machines/bravo** directory. These must both be created.

*Specify parent of* ***local***

Login as **root** on *bravo*.

Create a "machines" directory in order to specify the parent "serves" property:

```
niutil -create -t bravo/local /machines/alpha
```

Add a value to the *serves* property of this directory specifying that the parent "**..**" is served from database ***network*** on host *alpha*.

```
niutil -addval -t bravo/local /machines/alpha
     serves "../network"
```

Add a property to this directory giving the IP address of the host alpha.

```
niutil -createprop -t bravo/local /machines/alpha
     ip_address 192.42.172.1
```

This ensures that the NetInfo database bravo/local knows the IP address of the parent host aplha and is thus able to bind to it correctly on startup.

*Specify child of* ***network***

The parent database must specify that it has a new child, **bravo**.

Login as **root** on *alpha*.

Create a "machines" directory in order to specify the child "serves" property:

```
niutil -create -t alpha/network /machines/bravo
```

Add a value to the *serves* property of this directory specifying that the child is served from database **local** on *bravo*.

```
niutil -addval -t alpha/network /machines/bravo
     serves "bravo/local"
```

Add a property to this directory giving the IP address of the host alpha.

```
niutil -createprop -t alpha/local /machines/bravo
     ip_address 192.42.172.2
```

*Naming the child domain*

The property value in the parent database:

```
serves "bravo/local"
```

is where the child domain gets its name. This property is saying a domain with the name **bravo** (relative to the current domain) is served from a database with tag ***local***. The full pathname of this domain is /**bravo**.

**6: Test the Network**

The domain hierarchy can be tested by interrogating the database using the domain name rather than the address.

On *bravo* there is one database, ***local***, which serves the domain **/bravo**.

On *alpha*, there are two databases: ***local***, which serves **/alpha**, and ***network*** which serves the root domain, /.

List the properties which describe the machines in each of these databases:

```
niutil -read /bravo /machines/bravo
niutil -read /bravo /machines/alpha

niutil -read / /machines/bravo
niutil -read / /machines/alpha

niutil -read /alpha /machines/alpha
```

If the binding has not worked, an error message is displayed, saying that the database served by the specified domain cannot be opened.

**7: Load System Information**

Configuration information must be accessible to each host. A host will consult its local database first. If the appropriate configuration information is not stored here, the parent domain is consulted, and so on up the domain hierarchy.

In the previous example, configuration information was loaded into the root domain, the parent of this new domain, therefore, it is not necessary to load any information from the flat files.

Administrators may choose to store information at different nodes of the domain hierarchy rather than the root domain. If so, information may have to be loaded into the new local domain.

**8: Reboot and Test**

At this stage, reboot the system and check that both the daemons started successfully.

Check that the binding is correct by interrogating the database using the domain names instead of the database tags.

Try a common command, such as:

```
/bin/ls -lg
```

Check that the results are what you would expect.

This command must look up user id and group id information, using NetInfo.

# Chapter 6

# Using NetInfo

Configuration information for each host in the network is stored in the NetInfo databases. The administration of hosts is based on domains. These domains can be linked together to form a hierarchy.

A domain obtains information from the database from which it is served. The database information is used instead of the flat files or NIS when NetInfo is running. The database must exist on one of the hosts connected to the network. A domain, typically, has access to the resources specified in its child domains.

This chapter is divided into three parts:

- Database and Domain functions
  This section explains how to use the NetInfo tools to manage domains and databases, including create and delete databases, and bind domains.
- Managing Hosts
  This section explains how to add, move, and delete hosts from a network. Host management is achieved using the database and domain functions of the previous section.
- Managing Users and Groups
  Once hosts have been connected to the network, users can be given access using the database and domain functions.

## Database and Domain Functions

A domain is an abstract concept which enables the administrator to group resources together for access by specific users. Information about a domain is stored in the database from which it is served.

Domains exist dynamically in the system. When the system is started (usually at boot time), the NetInfo daemon searches the `/etc/netinfo` directory for databases and binds the domains they serve into the hierarchy. The hierarchy is determined by the "serves" properties of the machines in each database.

A domain does not know its own name: this is determined by its parent. It is possible for a domain to have a different parent. If, for example, one host is unavailable, another may be used to link the domain hierarchy. The database on this new host may act as parent to those on another, but it may access them using different names. Normally, this is transparent to the users.

Two tools are used to create and manage database information:

| | |
|---|---|
| **nidomain** | creates and deletes databases. |
| **niutil** | manages databases directories and properties. |

**niwhich** can be used to display domain and database details. Two other tools, **niload** and **nidump**, are used to move information between NetInfo and other systems.

### Compulsory Database Information

Some information is compulsory in all databases; specifically, some directories must exist, and some subdirectories must contain certain properties.

Each host in the network must have a ***local*** database. This database must be called "local". The name of the local domain is the same as the host name of the machine on which it is located.

For example, the local domain on host *alpha*, whose parent is the root domain has the domain name **/alpha**. It is served from the database with the tag, ***local***, and has the database address ***alpha/local***.

The local database is the only compulsory one. A host need not be connected to other machines in order to run NetInfo. If, however, a hierarchy of domains is generated, the top level domain, the root domain, must be served from a database called ***network***.

### Directories /users Directory

There are several directories that must exist in every database:

Each database must have a **`/users`** directory. Users who require access to the domain served by the database must have an entry in this directory. Access is achieved by creating a sub-directory with the same name as the user's login name.

For example, to create a superuser account, create a subdirectory called **`/users/root`**, and to create an account for a user called "chris", create a subdirectory called **`/users/chris`**.

#### Inherited Information

Not all users who have access to a domain need an account (entry in **`/users`**) in all domains. Users who have access to a higher level domain automatically have access to all its child domains.

*Example:*

*Inherited Information*

Suppose the following hierarchy exists where the domains are served from the specified databases:

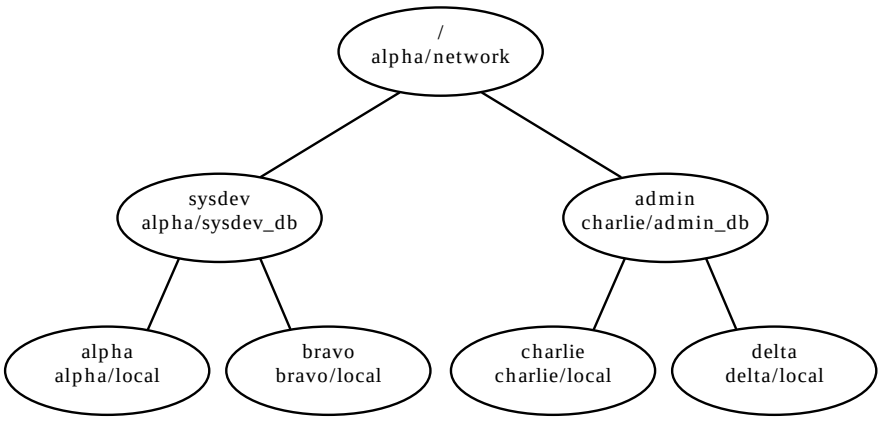



*Figure 5.1 - Inheriting Domain Information*

All users with accounts in the **admin** domain have access to hosts *charlie* and *delta*: they don't need to have **/users** entries in the local databases as well.

It is recommended, however, that the superuser has an entry in every domain, not just the root domain. In case there are any problems, such as machines going down, the superuser is then able to connect to any host and make changes as necessary.

**Properties of /users**

Each subdirectory of **/users** must have the following properties:

| | |
|---|---|
| name | Login name of the user. The value of this name is the name of the subdirectory. |
| passwd | User's password. The value of the password is encrypted. |
| uid | User ID. Each user must have a unique identification number. |
| gid | Group ID. Each user must be a member of a group. |
| real_name | The user's real name can be entered into this property. |
| home | The value of the "home" property determines the users home directory. The value of this property must be a valid UNIX pathname. |
| shell | The "shell" property determines the user's login shell. The value must be the pathname of a valid UNIX shell program. |

**Superuser**

The following subdirectory must exist in each domain to create a superuser account:

```
/users/root
```

The root user must have user id (“uid” property) set to 0, and group id (“gid” property) set to 1.

## /machines Directory

The **/machines** directory contains information about what hosts the domain has access to. Each domain must know about its parent domain, and its child domain(s).

There must be a subdirectory within **/machines** for each host that stores a database for the current domain, for its parent, or for any of its child domains.

When a database is created, one entry is created in the **/machines** directory: it corresponds to the name of the host on which the database is created.

For example, to create a domain, **sysdev**, served from a database called ***sysdev_db*** on host *alpha*, the following command is issued whilst logged into *alpha*:

```
nidomain -m sysdev_db
```

This command creates a database called ***sysdev_db***. This database is created with the following directory entries:

```
/machines
     /alpha
          name = “alpha” (determined from the name of the
          host)
          ip_address - “192.42.172.1” (determined from the
          host file)
          serves = “./sysdev_db” (self is served from
          database sysdev_db)
```

### Properties of /machines

A subdirectory of the **/machines** directory has three properties created automatically.

| | |
|---|---|
| name | The name of the machine must be the host name. This becomes the name of the subdirectory. |
| ip_address | Each machine must have a unique Internet address. |
| serves | The serves property has one or more values. The values have the form:<br>*domain / tag*<br>where “domain” is either the self “.”, the parent “..”, or the domain name of a child. This is the point at which the parent determines the name of a child domain. |

This directory can have many other properties used by the system. These are explained in full in the *Reference* chapter.

*Example:*

*Properties of **/machines** database directory*

Suppose a network is comprised of two machines: *alpha* and *bravo*, which are linked together in a two-level hierarchy. Each system has a local domain, the parent of which is the root domain, located on host *alpha*.

Three domains must be created:

On *alpha*, create a local database called ***local***, whose parent, the root domain, **/**, is served from a database called ***network***, also located on *alpha*.

Host *bravo* also has a local domain, and it is the child of the root domain. The domain hierarchy is as follows:

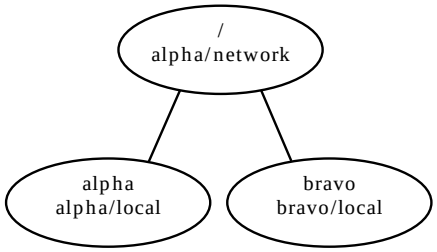



*Figure 5.2 - Two-Level Domain Hierarchy*

The ***local*** database on *alpha* (domain name **/alpha**, database ***alpha/local***) has the following **/machines** entries:

```
/machines/alpha
        serves =“./local”
                “../network”
```

This entry is saying:

For domain **/alpha**, served by database ***local*** on host *alpha*, its self (“.”) is served from database with tag **local** on machine *alpha* (**/machines/alpha**).

Its parent (“..”) is served from database with tag **network** on machine *alpha* (**/machines/alpha**).

The ***local*** database on *bravo* (domain name **/bravo**, database ***bravo/local***) has the following **/machines** entries:

```
/machines/alpha
        serves = “../network”
/machines/bravo
        serves = “./local”
```

This entry is saying:

For domain **/bravo**, served by database ***bravo/local***, its self (“.”) is served from database with tag ***local*** on machine *bravo* (**/machines/bravo**).

Its parent (“..”) is served from database with tag ***network*** on machine *alpha* (**/machines/alpha**).

*Example:*

*Properties of* ***/machines*** *database directory*

The **root** domain, "**/**", served from database ***alpha/network***, has the following **/machines** entries:

/machines/alpha

serves = "./network"

"alpha/local"

/machines/bravo

serves = "bravo/local"

This entry is saying:

For the root domain **/**, served by database ***alpha/network***, its self (".") is served from database with tag ***network*** on machine *alpha* (**/machines/alpha**).

This domain does not have a parent. It does, however, have two child domains.

On machine *alpha*, (**/machines/alpha**) there is one child domain, called ***alpha*** which is served from database with tag **local**.

On machine *bravo*, (**/machines/bravo**) there is one child domain, called ***bravo*** which is served from database with tag **local**.

## Managing Databases

A domain obtains its information from a database that has been created on a specific host. The domain only exists when it is bound into the domain hierarchy.

Databases are created (and deleted) using the **nidomain** tool, and the directories and properties within that database are manipulated using **niutil**. Some standard database information can also be loaded using **niload**, and subsequently dumped out of the database using **nidump**. Password information can be changed using **nipasswd**. This is explained in the "*Managing Users*" section.

### Naming Databases

In general, a database can have any name: it does not have to be the same as the domain name. In fact, it is probably less confusing to give the databases a different name to the domains they serve.
There are two compulsory names, "local" and "network". Local domains must be served from a database called ***local*** and the root domain must be served from a database called ***network***.

*Example:*

*Create a local database*

Each host must have a local database in order to connect into the hierarchy. This example creates this database on host *alpha*.

Login as **root** on host *alpha* and use **nidomain** to create the database:

```
nidomain -m local
```

Here, "local" is the name of the database, not the domain. The "-m" option specifies that the database is a master database as opposed to a clone. In most cases, the database will be a master. See the section on *Creating a Clone* for more information about clones.

**nidomain** performs several functions. First, it creates a directory for the new database, and creates a database file. It then starts up a **netinfod** daemon to access this database. Some initial information is also loaded into the new database.

### Directory and Files

All database directories are created in `/etc/netinfo`, which must exist. This directory should be created at installation time. The subdirectory is named after the database, with an extension of ".nidb". The database file is called "collection". As the database grows, extra files will be created in this sub-directory with the name "extension_N", where "N" is an integer generated by the system that corresponds to the internal directory number stored in that file.

The above example creates the following directory and file structure:

```
/etc/netinfo/local.nidb
     collection
```

**Process - "netinfod"**

Each database is accessed by a **netinfod** process. This process is started when the database is created. If the system is rebooted, the **nibindd** daemon will restart a **netinfod** process for each database it finds in the directory `/etc/netinfo`.

The example above started up the following process (display using **ps**):

```
root 71   S   0:12 /usr/etc/netinfo local
```

**Database Information**

Some information is automatically loaded in the database when it is created. A **`/machines`** internal directory is created, with one subdirectory that corresponds to the name of the host on which the database was created. This subdirectory has three properties: the name, taken from the host name; the Internet address, extracted from the `/etc/hosts` file; and the "serves" property, which has one value specifying that the self domain is served from database with tag ***local***.

The above example creates the following information:

```
/machines/alpha
     name:       alpha
     ip_address: 192.42.172.1
     serves:     ./local
```

This database has tag ***local*** on host *alpha*. Its database address is ***alpha/local***. The name of the domain that it will serve is not known until it is bound into the hierarchy. Then, the name is determined by the parent domain. This is explained in a later section in this chapter, *Managing Domains*.

*Example:*

*Create a network database*

The ***network*** database, which serves the root domain, is created using the same tool, **nidomain**.

Login as **root** on *alpha* and use **nidomain** to create the ***network*** database:

```
nidomain -m network
```

This example creates the following directory and file structure:

```
/etc/netinfo/network.nidb
     collection
```

This file, contains the following default information:

```
/machines/alpha
     name:       alpha
     ip_address: 192.42.172.1
     serves:     ./network
```

This database has tag ***network*** on host *alpha*. Its database address is ***alpha/network***. Once it is bound into the domain, it will serve the root domain, and will have the domain name "/".

### Deleting Databases

The **nidomain** tool is used to delete databases as well as create them. Use the destroy option to delete a database. Login as **root** to the host on which the database is stored.

```
nidomain -d some_db
```

This example removes the directory and database files:

```
/etc/netinfo/some_db.nidb
```

**nidomain** will also stop the netinfod process.

When a database is deleted, the domain it serves must be removed from the domain hierarchy. This means that the serves property of its parent domain, and any child domains, must be adjusted. This is explained more fully in the section *Managing Domains*.

## Managing Database Directories

The **niutil** program is used to manipulate information in the database. Databases can be altered by specifying them using the database tag, (with the **-t** option) or by using the domain name, if the domain exists. A domain does not exist until it is bound into the hierarchy, so much of the database information can only be manipulated using the tag.

**Directory Management Options**

Three **niutil** options are relevant to database directories:

**niutil** [ -t ] [ -p ] **-list** *domain path*
**niutil** [ -t ] [ -p ] **-create** *domain path*
**niutil** [ -t ] [ -p ] **-destroy** *domain path*

If using the **-t** option, "domain" must be a database tag, otherwise, "domain" must exist.

"**-create**" is used to create directories, and "**-destroy**" will remove them.
"**-list**" is used to display what subdirectories exist in the specified "path".

The following example is used to illustrate the "display", "create" and "delete" options of the **niutil** tool.

*Example:*

*Managing Database Directories*

A new host, *bravo*, will be added to the network to replace the host called *foxtrot*, which is a child of the root domain. The network database is currently located on host *alpha*.

The current domain structure is as follows:

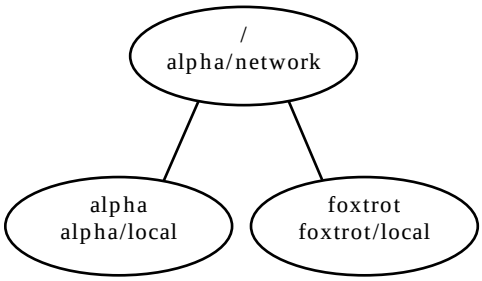



*Figure 5.3 - Managing Database Directories*

The **network** database has two subdirectories in **/machines**

**/machines/alpha**
**/machines/foxtrot**

Changes

A new subdirectory, **/machines/bravo**, must be added to the **network** database which serves the root domain.

The existing directory, **/machines/foxtrot**, must be removed from the database as it is no longer required.

**Display Directory Information**

The **-list** option of **niutil** displays the internal directory structure of the specified directory, of the specified database.

The database can be named using the domain name if it exists, or the database tag.

*Using the domain name*

Check the current directory list of the **root domain**. This domain is served by the **network** database, but, as the root domain exists in the hierarchy, the domain name, /, can be used.

```
niutil -list / /
     1    machines
     5    users
```

This example asks for a list of directories of the root domain, from the root directory. The two slashes are used to represent two different components: the first requests the root domain, the second, the root directory of the specified domain.

The result shows there are two subdirectories: **/machines** is directory number 1 and **/users** is directory number 5. These numbers are generated by the system when the directories are created.

```
niutil -list / /machines
     2    alpha
     3    foxtrot
```

This example asks for a list of directories of the root domain, from the **/machines** directory. The result shows there are two subdirectories, **/alpha**, and **/foxtrot**, which are directories 2 and 3 respectively.

*Using the database tag*

The above examples accessed the root domain using its domain name "**/**". In this example, its tag is used. It is served from the database called **network**, on host *alpha*, therefore its database address is: ***alpha/network***.

Check the root directory list of the network database:

```
niutil -list -t alpha/network /
     1    machines
     5    users
niutil -list -t alpha/network /machines
     2    alpha
     3    foxtrot
```

As can be seen, the results are the same, whether using the domain names or the database tags.

**Create Directory Information**

The internal directory structure of a database is created using the **niutil** tool. The **-create** option creates the specified directory, of the specified database.

The database can be named using the domain name if it exists, or the database tag.

*Using the domain name*

The new directory, **/bravo**, must be added to the root domain, "**/**", as a subdirectory of **/machines**. Use the create option as follows:

```
     niutil -create / /machines/bravo

and check:
     niutil -list / /machines
          2    alpha
          3    foxtrot
          6    bravo
```

*Using the database tag*

Add the new directory, **/bravo**:

```
     niutil -create -t alpha/network /machines/bravo

and check:
     niutil -list -t alpha/network /machines
          2    alpha
          3    foxtrot
          6    bravo
```

**Delete Directory Information**

The internal directory structure of a database is removed using the **niutil** tool. The **-destroy** option deletes the specified directory, of the specified database.

The database can be named using the domain name if it exists, or the database tag.

*Using the domain name*

The old directory, **/foxtrot**, must be removed from the root domain, /, as a subdirectory of **/machines**. Use the destroy option as follows:

```
niutil -destroy / /machines/foxtrot
```

and check:

```
niutil -list / /machines
     2    alpha
     6    bravo
```

*Using the database tag*

Remove the old directory:

```
niutil -destroy -t alpha/network /machines/foxtrot
```

and check:

```
niutil -list -t alpha/network /machines
     2    alpha
     6    bravo
```

## Managing Properties

Properties are attached to directories in a database. A property can have zero or more values.

The following options of the **niutil** tool are used to manage the values of properties:

**niutil** [ -t ] [ -p ] **-createprop** *domain path propkey [ propval ... ]*
**niutil** [ -t ] [ -p ] **-destroyprop** *domain path propkey*
**niutil** [ -t ] [ -p ] **-mergeprop** *domain path propkey propval1 ...*
**niutil** [ -t ] [ -p ] **-appendprop** *domain path propkey propval1 ...*
**niutil** [ -t ] [ -p ] **-destroyval** *domain path propkey propval1 ...*
**niutil** [ -t ] [ -p ] **-read** *domain path*

If using the **-t** option, "domain" must be a database tag, otherwise, "domain" must be bound into the hierarchy.

"**-createprop**" is used to create properties. Values can be assigned at the same time.
"**-destroyprop**" removes the specified property and all its values.
"**-read**" is used to display the properties and values of the specified directory path.
"**-mergeprop**", "**-appendprop**" and "**-destroyval**" are used to manage values. The specified property need not already exist.

*Example:*

*Managing Database Properties*

This example showing how to manage properties and values follows from the previous example that manipulates directories.

The new directory for host *bravo* has been added in the previous example. However, this new directory has does not have any properties yet (apart from the name).

The domain structure required is as follows:

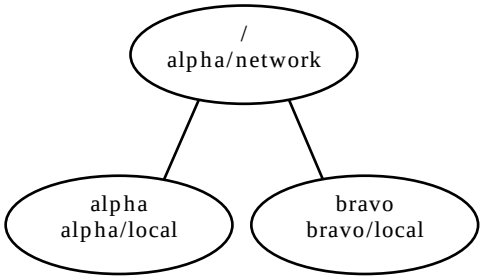



*Figure 5.4 - Managing Database Properties*

The **network** database has two subdirectories in **/machines**

**/machines/alpha**
**/machines/bravo**

Changes to "network"

The "ip_address" property must be created, with value "192.42.172.2".

The serves property must be created and assigned a value that specifies ***bravo/local*** as its child domain.

**Display Database Properties**

Directory properties are displayed using the **-read** option of the **niutil** tool.

*Using the domain name*

Check the properties of the **/machines/alpha** directory in the root domain. The domain name can be used as it has been bound into the hierarchy.

```
niutil -read / /machines/alpha
     name: alpha
     ip_address: 192.42.172.1
     serves: ./network  alpha/local
```

The “serves” property specifies that self (root domain) is served from a database called **network** and it has a child domain, which it calls **alpha** served from a database called **local**. As both these entries are in the **/machines/alpha** directory, then the database files must be located on the host *alpha*.

*Using the database tag*

Check the properties of the **/machines/alpha** directory:

```
niutil -read -t alpha/network /machines/alpha
     name: alpha
     ip_address: 192.42.172.1
     serves: ./network
```

**Create Properties and Values**

Database properties can be created using the **-createprop** option of **niutil**. This option also adds values to the new property as required.
If the property created using **createprop** already exists, all existing values are overwritten. Normally, values are changed using **destroyval** and **addval**; **createprop** can be used to change all values of a property by overwriting the existing values.

*Using the domain name*

Add the “ip_address” and “serves” properties to the /machines/bravo directory in the **network** database.

```
niutil -createprop / /machines/bravo ip_address “192.42.192.2”
niutil -createprop / /machines/bravo serves “bravo/local”
```

*Using the database tag*

Add the “ip_address” and “serves” properties to the /machines/bravo directory in the **network** database.

```
niutil -createprop -t alpha/network /machines/bravo
     ip_address “192.42.192.2”
niutil -createprop -t alpha/network /machines/bravo
     serves “bravo/local”
```

**Add Property Values**

In the current example, the root domain is served from the **network** database on host *alpha*. This domain has a child, the local database, also on alpha.

The properties of the **/machines/alpha** directory are:

```
niutil -read / /machines/alpha
     name:          alpha
     ip_address:    192.42.172.1
     serves:        ./network
```

A value must be added to the serves property to specify the child domain.

```
niutil -mergeprop / /machines/alpha serves “alpha/local”
```

and the directory values should read:

```
niutil -read / /machines/alpha
     name:     alpha
     ip_address:    192.42.172.1
     serves:        ./network  alpha/local
```

The same result could be achieved using the **-createprop** option only. This option, however, overwrites any existing values, and so all values need to be entered:

```
niutil -createprop / /machines/alpha
     serves “./network” “alpha/local”
```

This option can be run using the domain name, if it exists, otherwise the database tag should be used.

### Remove Values and Properties

A property and all its values can be removed from a directory using the **-destroyprop** option. If a value but not the entire property needs to be removed, use the **-destroyval** option instead.

Once again, these options can be used with the domain name, if it exists, or else the database tag.

Adding and removing property values:

```
niutil -mergeprop / /machines/alpha serves “delta/local”
niutil -destroyval / /machines/alpha serves “delta/local”
```

Adding and removing properties:

Create an arbitrary property with no values:

```
niutil -createprop / /machines/alpha new_prop
```

Remove the new property:

```
niutil -destroyprop / /machines/alpha new_prop
```

If any values existed for this property, they, of course, would be removed.

## Managing Domains

A domain only exists when it is bound into the domain hierarchy. There are two stages to creating a domain:

- create the database
- bind the domain into the hierarchy.

Database creation is explained above. Binding is achieved by setting the "serves" properties of the **/machines** sub-directories in each database to point to the following:

- the database that serves the current domain (self),
- the database serving the parent,
- and the database serving any child domains, if they exist.

The leaves of the domain hierarchy are the local domains serving each host, which are usually the first domains to be created on the machine. Local domains cannot have child domains below them.

### Names and Binding

Binding is a two-way process that is achieved by setting the "serves" properties of each database. The "serves" property must point to up to three places:

- Database serving current domain (self - ".").
- Database serving parent ("..."). The root domain does not have a parent.
- If a domain has child domains, then the parent must specify the databases which serve each of its children.

For each of the above, both an uplink and a downlink must be created. An uplink exists in a child database and points to the database serving its parent domain. The downlink exists in the parent database, and points to the database serving the child domain. The downlink also gives the child domain its name.

*Example:*

*Domain Pathnames*

The name of a domain is specified by the path it is in.

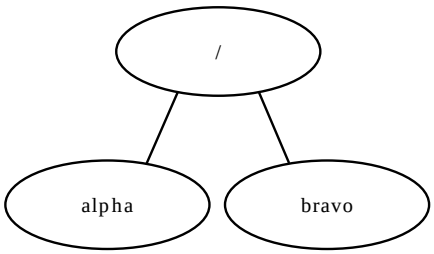



*Figure 5.5 - Domain Pathnames*

The root domain is called "/".

The domain served by the local database on *alpha* is a child of root. Its full domain name is **/alpha**.

The domain served by the local database on *bravo* is a child of root. Its full domain name is **/bravo**.

If another domain is inserted between the local and root domains, say, **sysdev**, then the leaf domain on *alpha* would have the full domain path name: **/sysdev/alpha**.

A domain does not know its own name. It must know which database it is served by, but the name is generated by the downlink in its parent database.

*Example:*

*Domain Name*

Suppose a database is created on host *alpha* using the following command:

```
nidomain -m sysdev_db
```

This command creates a database, ***sysdev_db*** on host *alpha*, which therefore has the database address ***alpha/sysdev_db***. The domain name will exist when it is bound into the hierarchy, and will be specified by a "serves" property value in its parent database.

Suppose this database is to serve a domain called **sysdev** which is the child of the root domain, also located on *alpha*.

In the root database, ***network***, the **/machines/alpha** directory must be updated with the following value of the serves property:

```
"sysdev/sysdev_db"
```

The first part of the above line gives the domain its name, and the second part specifies the database it is served from. We know it is located on host *alpha*, as this value is assigned to the **/machines/alpha** subdirectory.

## Insert a Domain

A domain only exists when it is bound into the domain hierarchy.

The first step is to create the database. Then create the uplink and downlinks by updating the “serves” properties in the databases serving the self, parent and child domains.

Example:

Create **sysdev**

This example creates a non-local domain called **sysdev**, which is a child of the root domain, and the parent of the local **alpha** domain. This new domain should be served from the database file, **sysdev_db**.

The following structures show the initial and the final domain hierarchies:

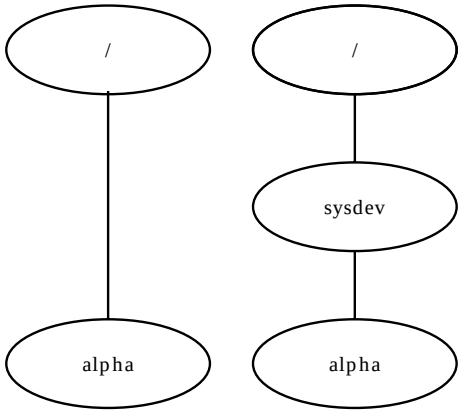



*Figure 5.6 - Inserting Domains into the Hierarchy*

This example inserts the domain, **sysdev**, between the existing structure, which means that the current binding must be changed.

All domains in the example are served from databases on host *alpha*. Therefore, only one **/machines** subdirectory is required in any of the databases.

*Steps*

The following actions should be performed in the specified order to create and bind in the new domain. You must be logged into *alpha* as the superuser to make these changes.

1. Create the database, **sysdev_db**.
2. Remove the existing binding between **/** and **/alpha**.
3. Bind the domain to the parent - parent is the root domain (served from the ***network*** database).
4. Bind the domain to its child domain - local domain on *alpha* which was previously called **/alpha**.

*Create a Database*

The name of the database which serves the **sysdev** domain is called ***sysdev_db***.

Use **nidomain** to create the database:

```
nidomain -m sysdev_db
```

A database with the address, ***alpha/sysdev_db*** now exists. The domain itself, to be known as **sysdev**, does not yet exist. See the section earlier for more details on creating a database.

*Remove Previous Binding*

The binding that connects **/** to **/alpha** must be removed. This can be done specifically using the **-destroyval** option of **niutil**, and removing the appropriate values of the serves properties. Another way of removing values is to overwrite them using the **-createprop** option.

1. Destroy the uplink (in the ***local*** database).

```
niutil -destroyval -t alpha/local /machines/alpha
     serves "../network"
```

2. Destroy the downlink (in the database serving the root domain).

```
niutil -destroyval  /  /machines/alpha
     serves "alpha/local"
```

*Bind to Parent*

Two steps are involved in binding a new domain to its parent.

1. Current domain must specify parent (uplink). Update the ***sysdev_db*** database by adding a new value to the serves property which specifies that the parent domain ("..") is served from the database ***network*** on host *alpha*. This database can only be updated using the database address, ***alpha/sysdev_db*** as the domain does not yet exist.

```
niutil -mergeprop -t alpha/sysdev_db  /machines/alpha
     serves "../network"
```

Add the uplink: **sysdev** to **root** domain

```
niutil -mergeprop -t alpha/sysdev_db  /machines/alpha
     serves "../network"
```

2. The parent domain must specify its new child **sysdev**. This parent domain exists and has the name "**/**".

Add the downlink: **root** domain to **sysdev**

```
niutil -mergeprop / /machines/alpha serves
"sysdev/sysdev_db"
```

This specifies that the parent knows its child domain by the name **sysdev**, and it is served from database ***sysdev_db*** on host *alpha*. A domain never knows its own name, only its parent knows it.

*Bind to Child*

The new domain must now be bound to its child domain. Two steps are involved here.

1. Current domain must specify its child (downlink). Update the ***sysdev_db*** database by adding a new value to the serves property which specifies that the child domain, **alpha**, is served from the database ***local*** on host *alpha*. This database can only be updated using the database address, ***alpha/sysdev_db*** as the domain does not yet exist.

Add the downlink: **sysdev** to **alpha**

```
niutil -mergeprop -t alpha/sysdev_db /machines/alpha
     serves "alpha/local"
```

2. Child domain must specify the new domain, **sysdev**, as its parent. The child domain has database address, ***alpha/local***.

Add the uplink: **alpha** to **sysdev**

```
niutil -mergeprop -t alpha/local /machines/alpha
     serves "../sysdev_db"
```

**Note - Overwriting and Adding Information**

The **createprop** option of **niutil** creates a property with the specified values. It overwrites any other values that may already exist.

The **addval** option adds a property value to an existing property key without overwriting other values. The property list should be checked before changes are made to ensure the connections remain correct.

The **destroyval** option removes a value without destroying the property key, while the **destroyprop** option removes the property and hence, all its values.

For example, the serves property values of the parent domain started as:

```
name: alpha
ip_address: 192.42.172.1
serves: "./network" "alpha/local"
```

After the above changes are made they should read:

```
name: alpha
ip_address: 192.42.172.1
serves: "./network" "sysdev/sysdev_db"
```

Part two of steps 2 (destroyval) and 3 (addval) could have been achieved using **createprop**:

```
niutil -createprop / /machines/alpha
     serves "./network" "sysdev/sysdev_db"
```

Although the value specifying that its self domain is served from ***network*** should not change in this example, it must be entered if the **createprop** option is used.

## Moving Domains

You may want to move a domain to a different position in the hierarchy as the access needs of a company change. All that is required is to change the binding properties of the old and new parent, and any child domains that are affected.

*Example:*

*Moving Domains*

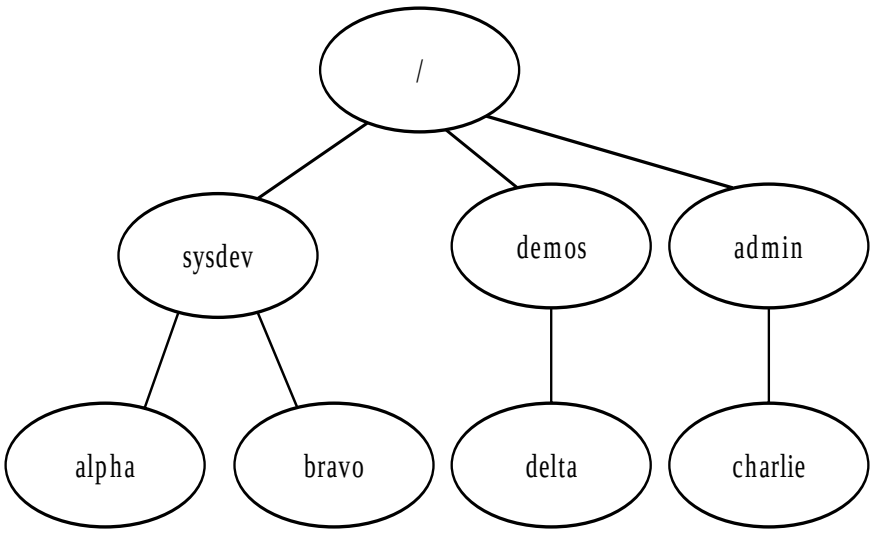



*Figure 5.7 - Moving Domains*

This exercise moves the **demos** branch from the **sysdev** domain, to become a child of the root domain.

**Changes**

The link between **sysdev** and **demos** must be removed.

A new link between **demos** and the **root** domain must be created.

None of the child domains of **demos** (**delta**) are affected as the whole branch is being moved.

The next diagram, *Figure 5.8*, shows the resulting domain structure.

Steps

1. Remove the link to old parent (**demos** and **sysdev**).
2. Add link to new parent (**demos** and **root**).

*Check current bindings*

Use **niutil** to check current "machines" and properties of the parent, **sysdev** domain, served from ***alpha/sysdev_db*** (must be logged in as superuser on *alpha*).

```
niutil -list -t alpha/sysdev_db /machines
     1 alpha
     5 bravo
     8 delta
```

*alpha*

```
niutil -read -t alpha/sysdev_db /machines/alpha
     serves: alpha/local, ./sysdev_db, ../network
```

*bravo*

```
niutil -read -t alpha/sysdev_db /machines/bravo
     serves: bravo/local
```

*delta*

```
niutil -read -t alpha/sysdev_db /machines/delta
     serves: demos/demos_db
```

Use **niutil** to check current "machines" and properties of the **demos** domain, served from ***delta/demos_db*** (must be logged in as superuser on *delta*).

```
niutil -list -t delta/demos_db /machines
     2 alpha
     1 delta
```

*alpha*

```
niutil -read -t delta/demos_db /machines/alpha
     serves: ../sysdev_db
```

*delta*

```
niutil -read -t delta/demos_db /machines/delta
     serves: ./demos_db, delta/local
```

Use **niutil** to check current "machines" and properties for the **root** domain (served from ***alpha/network***)

```
niutil -list -t alpha/network /machines
     1 alpha
     2 charlie
```

*alpha*

```
niutil -read -t alpha/network /machines/alpha
     serves: ./network, sysdev/sysdev_db
```

*charlie*

```
niutil -read -t alpha/network /machines/charlie
     serves: admin/admin_db
```

*Remove the link to the old parent*

Remove the old downlink: **sysdev** to **demos**

The downlink is specified in the **/machines/delta** subdirectory. As no other information exists here, the subdirectory can be removed. If other domains existed on this host, only the specific property value should be removed.

```
niutil -destroyval /sysdev /machines/delta
     serves "demos/demos_db"
```

or

```
niutil -destroy /sysdev /machines/delta
```

Remove the old uplink: **demos** to **sysdev**

The parent is no longer **sysdev**, so this value should be removed.

```
niutil -destroyval -t delta/demos_db /machines/alpha
     serves "../sysdev_db"
```

*Create the link to the new parent*

Create the new uplink: **demos** to **root**

The new uplink must point to the **root** domain, located on *alpha*.

```
niutil -mergeprop -t delta/demos_db /machines/alpha
     serves "../network"
```

Create the new downlink: **root** to **demos**

The downlink from the root domain must be added. The name of the domain is **demos**, served from database ***demos_db*** on host *delta*.

The **demos** domain is served from the ***demos_db*** database on host *delta*. A new **/machines** subdirectory must be created, and the "ip_address" and "serves" properties generated. Login as the superuser on host *alpha*.

```
niutil -create / /machines/delta
niutil -createprop / /machines/delta
     ip_address "192.42.172.4"
```

Now create the downlink by setting the "serves" property.

```
niutil -createprop / /machines/delta
     serves "demos/demos_db"
```

## Deleting Domains

Administrators may wish to delete domains from the system completely. If a host is no longer available, its local domain should be removed from the hierarchy. The hierarchy may also be restructured for some reason, resulting in the need to remove domains.

If a host is removed, all domains served from that host are affected: they have to be either moved, or deleted. If a host serves only its local domain, then the removal is simple: only the binding between the local and the parent domains is affected.

If a domain other than a local domain is removed, then bindings to both the parent and child domains are affected. There are three major steps involved in deleting a domain from the middle of the hierarchy:

- reset the binding from the parent
- reset the binding from child domains
- remove the database and **netinfod** process.

*Example:*

*Deleting a Domain*

This exercise removes the 2nd-level domain, **demos**. It does not remove the host *delta*, from the network, so the local database has to be relinked to the network. In this example, it is connected to the **admin** domain.

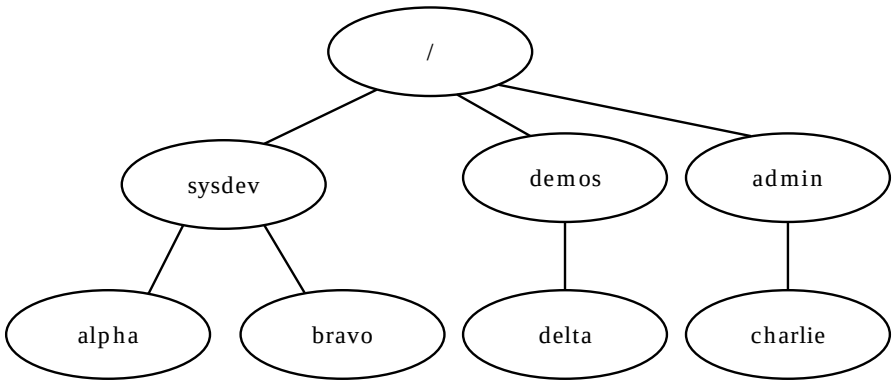



*Figure 5.8 - Deleting Domains*

Changes

The link between **root** and **demos** must be removed.

A new link between **delta** and the **admin** domain must be created.

The database and process serving **demos**, ***demos_db***, must be removed.

*Check current bindings*

Use **niutil** to check current "machines" and properties for the **root** domain (served from ***alpha/network***)

```
niutil -list -t alpha/network /machines
     1 alpha
     2 charlie
     9 delta
```

*alpha*

```
niutil -read -t alpha/network /machines/alpha
     serves: ./network, sysdev/sysdev_db
```

*charlie*

```
niutil -read -t alpha/network /machines/charlie
     serves: admin/admin_db
```

*delta*

```
niutil -read -t alpha/network /machines/delta
     serves: demos/demos_db
```

Use **niutil** to check current "machines" and properties of the **demos** domain, served from ***delta/demos_db*** (must be logged in as superuser on *delta*).

```
niutil -list -t delta/demos_db /machines
     2 alpha
     1 delta
```

*alpha*

```
niutil -read -t delta/demos_db /machines/alpha
     serves: ../sysdev_db
```

*delta*

```
niutil -read -t delta/demos_db /machines/delta
     serves: ./demos_db, delta/local
```

Use **niutil** to check current "machines" and properties of the **admin** domain, served from ***charlie/admin_db***.

```
niutil -list -t charlie/admin_db /machines
     2 alpha
     1 charlie
```

*alpha*

```
niutil -read -t charlie/admin_db /machines/alpha
     serves: ../sysdev_db
```

*charlie*

```
niutil -read -t charlie/admin_db /machines/charlie
     serves: ./admin_db, charlie/local
```

*Remove link from old parent*

As the domain is being removed, only the downlink from the parent (**root**) has to be altered.

Remove the downlink

```
niutil -destroyval / /machines/delta
     serves “demos/demos_db”
```

*Reset binding for “local” database*

The child domain, ***delta/local*** must change its old uplink to **demos** to an uplink to the **admin** domain. As the new parent is on a different machine, a new **/machines** subdirectory must first be created.

Add the uplink

```
niutil -create -t delta/local /machines/charlie
niutil -createprop -t delta/local /machines/charlie
     ip_address “192.42.172.3”
niutil -createprop -t delta/local /machines/charlie
     serves “../admin_db”
```

The downlink from the **admin** domain must be added. This domain in on host *charlie.* As this parent had no previous knowledge of the host *delta*, a new **/machines** subdirectory must be created.

Add the downlink

```
niutil -create -t charlie/admin_db /machines/delta
niutil -createprop -t charlie/admin_db /machines/delta
     ip_address “192.42.172.4”
niutil -createprop -t charlie/admin_db /machines/delta
     serves “delta/local”
```

## Joining Two Networks

The database serving the root domain of a network must be called **network**. Although unusual, there may be an occasion where two networks must be joined together.

*Example:*

*Join Networks*

Suppose two faculties of a University, Computer Science and Physics, have always managed separate networks using NetInfo. In order to share information and resources, including a systems administrator, they decide to connect these two together with a new root domain over the top.

The existing structure is as follows:

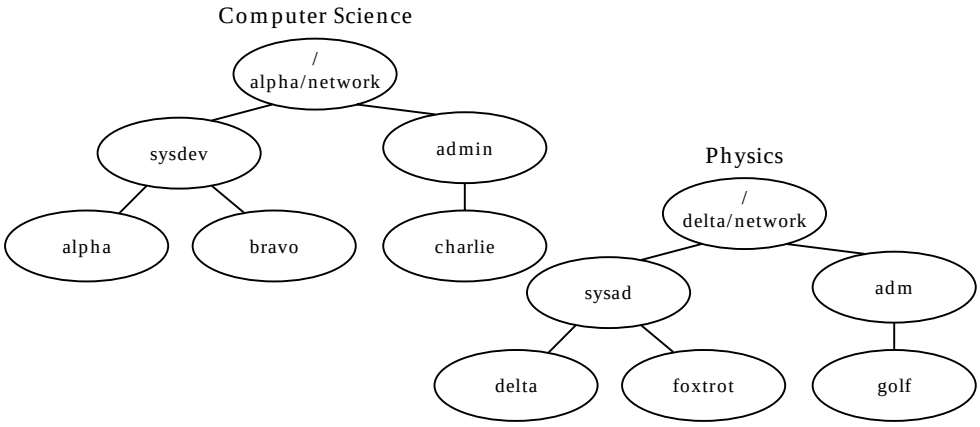



*Figure 5.9 - Separate Networks*

In this situation, there are two database files called "network", on hosts *alpha* and *delta*. If a new root domain was created on a new host, there could be a third file called network created, though this would be very confusing.

If, on the other hand, the new root domain was created on one of the existing hosts, (which is more likely), at least one of the network files will have to change its name.

It is recommended that only one **network** database ever exists in a network. Therefore, in order to connect two networks, some database files have to be renamed, and hence, some "serves" properties have to be altered.

The new structure, with the root domain on *alpha*, is as follows:

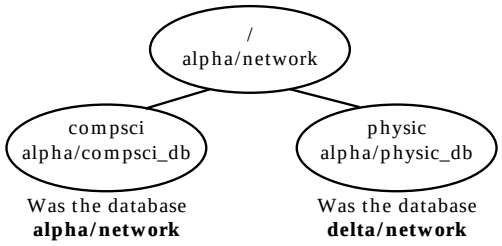



*Figure 5.10 - Joining Existing Networks*

**Changes required for joining two networks**

- Rename the old ***network*** database files on each host.
- Create a new ***network*** database on *alpha*.
- Update the binding to the CompSci sub-network. All second level domains (now third level) whose parent was served from the Compsci ***network*** will have to be changed.
- Update the binding to the Physics sub-network. All second level domains (now third level) whose parent was served from Physics ***network*** will have to be changed.

*Rename old "network" database files*

The existing ***network*** files are stored on *alpha* and *delta*. These must be renamed.

Once the names are changed, the domain hierarchy is "broken". Database tags rather than domain names must be used to make the changes.

On *alpha*:

```
mv /etc/netinfo/network.nidb /etc/netinfo/compsci_db.nidb
```

On *delta*:

```
mv /etc/netinfo/network.nidb /etc/netinfo/physic_db.nidb
```

*Create new "network" database*

Login to alpha as the superuser and use **nidomain** to create a new ***network*** database:

```
nidomain -m network
```

This new database needs two entries in the **/machines** subdirectories: **/machines/alpha** for the CompSci network and itself, and **/machines/delta**, for the Physics network.

The **/machines/alpha** directory should be created automatically, as the new network is created on this host.

*Create downlinks from new "network"*

The two downlinks to the old networks must be specified. Use **niutil** to alter the "serves" properties. The new ***network*** database should contain the following information:

```
/machines/alpha
     serves: ./network, compsci/compsci_db
/machines/delta
     serves: physic/physic_db
```

*Alter self references*

The new databases, renamed from ***network***, did not have parent directories, but it specified that it served itself from database ***network***. This reference must be removed.

**compsci** domain (***alpha/compsci_db***).

The reference to self served from network (***./network***) must be changed to the new database name (***./compsci_db***).

```
niutil -destroyval -t alpha/compsci_db /machines/alpha
     serves "./network"
niutil -mergeprop -t alpha/compsci_db /machines/alpha
     serves "./compsci_db"
```

**physic** domain (***delta/physic_db***).

The reference to self served from network (***./network***) must be changed to the new database name (***./physic_db***).

```
niutil -destroyval -t delta/physic_db /machines/alpha
     serves "./network"
niutil -mergeprop -t delta/physic_db /machines/delta
     serves "./physic_db"
```

*Bind to new parent*

The parent of the new domains is the new **root** domain. This information has to be added (may have to add the appropriate **/machines** directories).

Create uplinks: renamed networks to new network

**compsci** domain (***alpha/compsci_db***).

```
niutil -mergeprop -t alpha/compsci_db /machines/alpha
     serves "../network"
```

**physic** domain (***delta/physic_db***).

```
niutil -mergeprop -t delta/physic_db /machines/alpha
     serves "../network"
```

*Reset binding to child domains*

In the CompSci network, two domains, **sysdev** and **admin**, specify their parent as being served from database ***network*** on *alpha*. This in no longer true: the parent is now served from file ***compsci_db*** on *alpha*.

Similarly for the Physics network, child domains **sysad** and **adm** should specify their parents as ***physic_db*** on *delta*.

Each of these domains already has a parent, which should be destroyed, and the new value added. Depending on whether there are other values, the **-createprop** option may be used instead of **-destroyval** and **-mergeprop**:

On *alpha* (for **sysdev** domain)

```
niutil -destroyval -t alpha/sysdev /machines/alpha
     serves "../network"
niutil -mergeprop -t alpha/sysdev /machines/alpha
     serves "../compsci_db"
```

On *charlie* (for **admin**)

```
niutil -destroyval -t charlie/admin /machines/alpha
     serves "../network"
niutil -mergeprop -t charlie/admin /machines/alpha
     serves "../compsci_db"
```

On *delta* (for **sysad**)

```
niutil -destroyval -t delta/sysad /machines/alpha
     serves "../network"
niutil -mergeprop -t delta/sysad /machines/alpha
     serves "../physic_db"
```

On *golf* (for **adm**)

```
niutil -destroyval -t golf/adm /machines/alpha
     serves "../network"
niutil -mergeprop -t golf/adm /machines/alpha
     serves "../physic_db"
```

## Moving Information between NetInfo and Flat Files

If NetInfo is not running, a UNIX system can operate using the flat configuration files. Some administrators may wish to keep the flat files up to date in case NetInfo is not available. Two utilities are available which enable information to be transferred to and from NetInfo.

### Loading Information

**niload** loads information from a UNIX-format file into the database. It reads from standard input, interpreting the information according to the specified format. See the *Reference* section for a full explanation of this command.
**niload** updates information if entries exist in both the database and the input file. If input file entries don't exist in the database, they will be added. It will not delete (by default) any entries that do exist in the specified database but do not exist in the input file.

**niload** understands the format of the following system files:

| | | | |
|---|---|---|---|
| aliases | bootparams | bootptab | fstab |
| group | hosts | networks | passwd |
| printcap | protocols | rpc | services. |

The format of the command is:

```
niload [ options ] format domain
```

*Example:*

*Loading Information*

Load information from the flat password file into the **sysdev** domain as follows:

```
niload passwd /sysdev < /etc/passwd
```

For each entry in the password file, a subdirectory in **`/users`** is created (or updated), and the properties assigned the values of the password entry.

### Dumping Information

**nidump** extracts information from a UNIX-format file into the database. It writes to standard output, interpreting the information according to the specified format. See the *Reference* section for a full explanation of this command.

**nidump** understands the same file formats as **niload**. The format of the command is:

```
nidump [-t] format domain
```

*Example:*

*Dumping Information*

Extract the password information from the **sysdev** domain and store it in a temporary password file as follows:

```
nidump passwd /sysdev > /tmp/passwd
```

For each entry in the **/users** subdirectory, a colon delimited line is written to standard output.

## Copying Information

The load and dump utilities can also be used to copy information from one NetInfo database to another.

Of course, there should be no need to copy password information from one database to another. In general, if two domains require similar password information, then it should be loaded in the parent database.

*Example:*

*Copying Information*

The dump and load commands can be used to copy information, such as “services” data, from one local domain to another as follows:

```
nidump services /alpha | niload services /bravo
```

# Managing Hosts

Hosts are the major resources in the network. Each host must have a local database before it can be bound into the network hierarchy. The information in the databases serving the domains on each host determines who has access to the host.

## Host Configuration Information

When NetInfo is running, the system first consults the NetInfo databases for configuration information. Before users can access a host, certain information must be made known.

The following information, if it exists, must be installed into a database before it can be used in the NetInfo network. The information need not exist in all local databases; it can be loaded into a mid-level parent database, or even into the root database for access by all other domains.

This information can be extracted from the UNIX flat files using **niload** (recommended), or entered a line at a time using other NetInfo utilities.

The information can be loaded into the specified internal directories using **niload** and the appropriate format.

Administrators should note that the flat file names, the format names, and the database directory names do not correspond exactly. **niload** can be used to transfer information using the specified file format.

| *File Format* | *Description* |
|---|---|
| **aliases** | Name aliases file recognised by sendmail, for the local host<br>Information from `/etc/aliases` is loaded into the **`/aliases`** directory by **niload**. This directory contains a subdirectory for each known alias with the following properties and values: |
| name | alias name |
| members | one value for each user who is part of the alias. |
| **bootparams** | This bootparams entry contains a list of diskless clients and their specific boot information.<br>Information from `/etc/bootparams` is loaded into the **/bootparams** directory by **niload**. |
| **bootptab** | Information from `/etc/bootptab` is loaded into the **/bootptab** directory by **niload**. |

| *File Format* | *Description* |
|---|---|
| **fstab** | Contains static information about file systems. |
| | Information from `/etc/fstab` is loaded into the **`/mounts`** directory by **niload**. This directory contains a subdirectory for each known file system with the following properties and values: |
| name | name of the file system |
| dir | pathname of directory on which to mount the file system |
| type | filesystem type |
| opts | mounting options |
| freq | interval (in days) between dumps |
| passno | the **fsck(8)** pass in which to check the partition |
| **group** | Contains information about user groups. |
| | Information from `/etc/group` is loaded into the **`/groups`** directory by **niload**. This directory contains a subdirectory for each known group with the following properties and values: |
| name | alias name |
| passwd | group password, can be set to null |
| gid | unique group id |
| users | the user name of each user who is a member of the group |
| **hosts** | Contains information about known hosts. |
| | Information from `/etc/hosts` is loaded into the **`/machines`** directory by **niload**. This directory contains a subdirectory for each known host, or machine, with the following properties and values: |
| name | main host name and aliases. Can contain more than one value if the machine has alias names. |
| ip_address | unique Internet address. |
| **networks** | Contains the network name database. |
| | Information from `/etc/networks` is loaded into the **`/networks`** directory by **niload**. This directory contains a subdirectory for each known network with the following properties and values: |
| name | main network name and any known aliases |
| address | network number |

| *File Format* | *Description* |
|---|---|
| **passwd** | Contains information about user accounts. |
| | Information from `/etc/passwd` is loaded into the **`/passwd`** directory by **niload**. This directory contains a subdirectory for each known user with the following properties and values: |
| name | user name |
| passwd | user's password, can be set to null |
| uid | unique user identification number |
| gid | group id of default group |
| realname | text field containing user information, can be null |
| home | pathname of UNIX home directory |
| shell | pathname of UNIX shell program |
| **printcap** | Contains printer capabilities entries. |
| | Information from `/etc/printcap` is loaded into the **`/printers`** directory by **niload**. This directory contains a subdirectory for each known printer with the following properties and values: |
| name | name of printer, and aliases |
| *options* | there must be a property for each printcap option required |
| **protocols** | Contains the protocol name database. |
| | Information from `/etc/protocols` is loaded into the **`/protocols`** directory by **niload**. This directory contains a subdirectory for each known protocol with the following properties and values: |
| name | main name as well as any aliases |
| number | protocol number |
| **rpc** | Contains the readable names that can be used in place of the rpc program numbers. |
| | Information from `/etc/rpc` is loaded into the **`/rpcs`** directory by **niload**. This directory contains a subdirectory for each known rpc program with the following properties and values: |
| name | name given to program |
| number | program number |
| **services** | Contains the service name database. |
| | Information from `/etc/services` is loaded into the **`/services`** directory by **niload**. This directory contains a subdirectory for each known service with the following properties and values: |
| name | main service name as well as any aliases |
| port | port number |
| protocol | protocol name |

## Clone Servers

If an organisation requires uninterrupted NetInfo service, a clone should be established. A clone server is an exact copy of a master servermaster server, that is, a database serving a particular domain. Clones cannot be created on the same host as the master database they copy.

It is up to the administrator to decide if there are sufficient resources available to establish a clone. Resources required include sufficient disk space, swap space, and memory.

### Reliability

If a host is down, or a domain cannot be connected for any reason, the clone database can be used instead of the inaccessible master database. Clone databases, however, cannot be modified if the master server is down. This ensures that there is only one source of domain information.

### Load Balancing

Clones can also be used to establish the best load balance of a network. A host will search for information locally before attempting to search the network. If a clone is established on a local host, it will obtain information from the clone rather than the master database elsewhere in the network.

### Propagating Information

When the **netinfod** daemon for each database is started, it first checks to see if the database is a clone, or if it has any clones. For each master database, if any clones are found, the daemon sets up tasks to ensure that any changes that are made to the master are also made to the clone as the changes occur.

### "master" property

The root directory of every database has a "master" property. The value of this property specifies the database address of the master database. If this property refers to itself, then the database is a master; if it refers to another database, then it must be a clone.

For example, check the "master" property of the root domain on alpha:

```
niutil -read -t alpha/network /
    master:alpha/network
```

Or using the domain name:

```
niutil -read / /
    master:alpha/network
```

This indicates that the database serving the root domain, ***alpha/network***, is a master database.

A clone of this database, with tag ***network***, served from host *foxtrot*, has the following "master" property:

```
niutil -read -t foxtrot/network /
    master:alpha/network
```

## Creating a Clone

When a clone is created, the entire master database is copied to the new host. When a clone is bound into the hierarchy, changes made to the master are automatically copied to the clone database.

A master database must know what clones exist. This is done using the "serves" property.

Each database has a self reference in the "serves" property of the **/machines** directory which specifies where the self domain is served from.

For example, the network database serving the root domain on *alpha* has the following value in the "serves" property in **/machines/alpha**:

```
ip_address:192.42.172.1
serves:   ./network
```

Clones are also specified using the dot "." notation. The system determines whether an entry is to a clone or not by looking at the value of the Internet address.

For example, if a database called ***network*** on host *foxtrot* is set up to clone the ***network*** database on host *alpha*, then ***network*** must have the following entries:

In **/machines/alpha**:

```
ip_address:192.42.172.1
serves:    ./network
```

In **/machines/foxtrot**:

```
ip_address:192.42.172.6
serves:    ./network
```

Both these entries refer to the self domain, ".", but the second is a clone on another host.

When a clone is created using the **nidomain** tool, information from the specified master database is copied to the new clone database. The master must be updated with the details of the new clone *before* the copying is done, to ensure that the clone is created correctly.

Note that it is possible to create a clone with a tag that is different from the tag of the master database. However, such clones will *not* be consulted during lookups as lookups always use the tag of the database as a key. Such clones therefore do not offer any advantages for load-balancing and so forth, although they do provide a form of automated backup for the database that they are cloning.

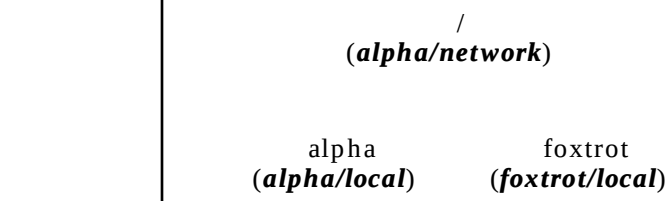
The following examples all show the creation of clones with the same tag as the master database, as this is the most common usage of the clone support provided by NetInfo.

*Example:*

*Setting up a Clone*

This example creates a clone of the ***network*** database. The database is stored on host *alpha* and the clone is created on host *foxtrot*.

The host on which the clone is to be created must have been connected to the network before the clone can be created. Hosts that serve clones must still have a local database, and be bound into the hierarchy. In this example, the local database on *foxtrot* is a child of the root domain. The structure is as follows:




On *alpha*, the following "serves" properties exist for ***network*** database:

network

**/machines/alpha:**

```
name          alpha
ip_address1   92.42.172.1
serves        ./network, alpha/local
```

**/machines/foxtrot:**

```
name          foxtrot
ip_address1   92.42.172.6
serves        foxtrot/local
```

Steps

1. Select the host on which the clone is to be created and choose a tag.
2. Update the master database specifying a "serves" property for the clone.
3. Create the clone database.
4. Reboot the system to start-up the clone.

*Select the clone details*

Before a clone is created, you must update the master with information about the clone. This includes the "serves" property that determines the clone server, which is copied when the clone is created.

For this example, the clone will be located on host *foxtrot* and the database will have the tag ***network***. Therefore the database address of the clone database is ***foxtrot/network***.

*Update the master database*

Clones are bound into the hierarchy by setting a "serves" property in the master database. The value of the serves property must read "./*clone_database_tag*" (i.e., self served from *clone_database*).

The database address of the clone is ***foxtrot/network***. Therefore, the **`/machines/-foxtrot`** directory must be updated to include the serves property "./network".

Update the ***network*** database with the following information:

```
niutil -mergeprop / /machines/foxtrot
        serves "./network"

/machines/foxtrot:

        name        foxtrot
        ip_address192.42.172.6
        serves      foxtrot/local, ./network
```

*Create the clone*

Login to *foxtrot* as root and create the clone database:

```
nidomain -c network alpha/network
```

This creates a database called ***network*** which is a copy of ***alpha/network***.

Check the following details:

- Check that the "network.nidb" database was created in `/etc/netinfo`.
- Check that the **netinfod** process to serve **network** was started.

*Reboot the system*

A new clone is not available to the network until the system has been rebooted.

When NetInfo is first started, the **netinfod** daemon for each master database checks for the existence of clones. If any are found, it sets up tasks to propagate any changes made to the master to update the clone. These tasks are only set-up when the daemon is first fired up, and so NetInfo must be restarted in order to use a new clone.

You can also kill and restart the nibindd processes on both machines to achieve the same result.

## Add a New Host to the Network

Each host that uses NetInfo must have a local domain. This domain is usually named after the machine, and is served from a database which must be called **local**.

A host does not have to be bound into the network in order to run NetInfo: it can be used isolated from the network as long as it has a **local** database. If this database did not exist, NetInfo cannot run, and another system, such as NIS or the flat configuration files would have to be used to configure the machine.

This section explains the steps involved in setting up the local database on a new host. If you have read directly from the *Getting Started* chapter, you will notice that the same information is covered.

*Example:*

*Add a New Host*

This example adds a new host to the network and binds it into the hierarchy as a child of the root domain.

The following diagram shows where the connection is made. This example assumes that the root domain is served from the machine *alpha*, and that this machine has been bound into the hierarchy. The local domain for *delta* is served from a database called ***local*** on *delta*.

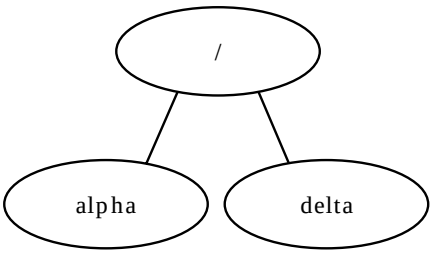



*Figure 5.11 - Adding a Host*

Steps

The following actions must be performed in the specified order to create and bind in the local domain for the new host.

1. Login to *delta* as the superuser, and check system.
2. Create the local database.
3. Create the “root” user for the database.
4. Load the configuration information.
5. Bind into the domain hierarchy.
6. Test the binding.

*Check new system*

Before adding a host to the hierarchy, the NetInfo programs must be loaded and running. See the *Getting Started* section for an explanation of initial requirements.

Check that the following exist:

Processes: **nibinddl ookupd**

Directories: /etc/netinfo

*Create the local database*

The **local** database; must exist on each host, and its name is fixed as **local**.

Login to *delta* and use the **nidomain** tool to create the database:

```
nidomain -m local
```

- Check that the `/etc/netinfo/local.nidb` directory is created; the **netinfod** process started and the default information loaded into the database.

*Create the "root" user*

Create a root user in the local database on *delta*. This user is the only one able to make changes to the database and must exist.

```
niutil -create -t delta/local /users
niutil -create -t delta/local /users/root

niutil -createprop -t delta/local /users/root uid 0
niutil -createprop -t delta/local /users/root gid 1
niutil -createprop -t delta/local /users/root passwd ""
```

*Load configuration information*

Configuration information can be loaded from the flat files, or extracted from NIS maps if the host is running NIS. See the earlier section, *Managing Hosts*, to see what information has to be loaded.

*Bind local domain to its parent*

Binding is achieved by adding *serves* properties.

- The **local** domain must specify its parent.
- The parent domain must specify the new domain as its child.

*Specify parent of **local***

The parent of **local**, is served by the **network** database on host *alpha*. Add a value to the *serves* property of the **`/machines/alpha`** directory of the database ***alpha/-local*** specifying the parent "**..**" is served from database **network**.

```
niutil -mergeprop -t alpha/local /machines/alpha
     serves "../network"
```

*Specify child of **root***

The child of **root** (***alpha/network***) is served by the **local** database on host *alpha*. Add a value to the *serves* property of the **/machines/alpha** directory of the database ***alpha/network*** specifying the child is served from database **local**.

```
niutil -mergeprop -t alpha/network /machines/alpha
     serves "alpha/local"
```

*Test the Network*

Since a domain does not exist until it is bound into the hierarchy, it cannot be referred to by its domain name, only its database tag. The previous interrogations of the database properties all made requests using the database tag (**-t** option of **niutil**). The domain hierarchy can be tested by interrogating the database using the domain name rather than the tag.

The ***local*** database serves a domain on host alpha. The full name of this domain is **/alpha**. The ***network*** database serves the root domain, called **/**.

List the properties which describe the machines in each of these databases:

```
niutil -read /alpha /machines/alpha
niutil -read / /machines/alpha
```

If the binding has not worked, an error message is displayed, saying that the database served by the specified cannot be opened.

## Move a Host to a Different Domain

This section explains how to move a host from one domain to another.

The ***local*** database for the host already exists, so all that is required is to change the bindings to the parent domain. A host must be a leaf node of the hierarchical tree, therefore there are no child domains to rebind.

Example:

Move host

This example moves the host *bravo* from the **sysdev** domain to the **admin** domain.

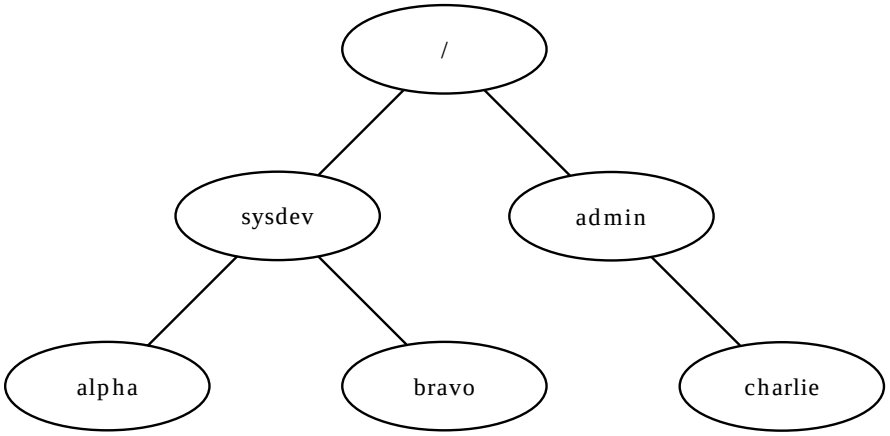



*Figure 5.12 - Moving a Host*

After the host *bravo* is moved from the **sysdev** domain to the **admin** domain, the structure is as follows:

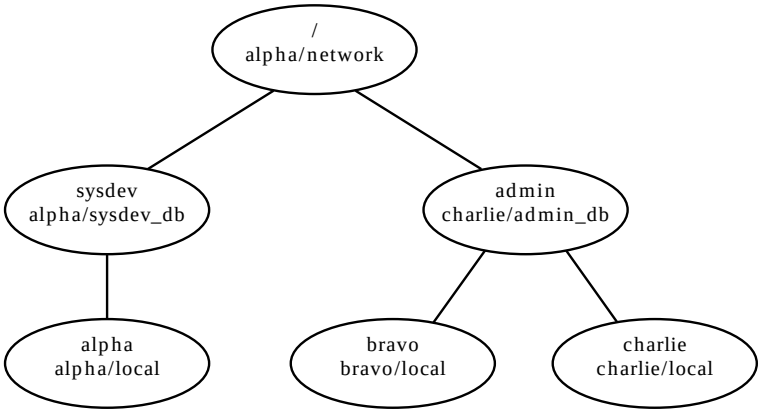



*Figure 5.13 - Moving a Host*

Only two steps are required:

1: Break the old link binding *bravo* to **sysdev**.
2: Create a new link binding *bravo* to **admin**.

This example involves three hosts *alpha*, for domain **sysdev**, *charlie*, for domain **admin**, and the host being moved, *bravo*.

*Remove old binding*

This is a two way process - break the uplink in the local database, and then break the downlink from sysdev.

On *bravo*, remove the parent reference in the local database:

```
niutil -destroyval -t bravo/local /machines/alpha
     serves "../sysdev_db"
```

On *alpha*, remove the child reference in the **sysdev** domain:

```
niutil -destroyval -t alpha/sysdev_db /machines/bravo
     serves "bravo/local"
```

*Bind to new parent*

Because the host is being moved to a domain in a different branch of the hierarchical tree, there is unlikely to be an entry in the **/machines** subdirectories.

Create the uplink (***local*** to ***admin)***

```
     niutil -create -t bravo/local /machines/charlie
     niutil -createprop -t bravo/local /machines/charlie
          ip_address "192.42.172.3"
     niutil -createprop -t bravo/local /machines/charlie
          serves "../admin_db"
```

Create the downlink (***admin*** to ***local)***

```
     niutil -create -t charlie/admin_db /machines/bravo
     niutil -createprop -t charlie/admin_db /machines/bravo
          ip_address "192.42.172.2"
     niutil -createprop -t charlie/admin_db /machines/bravo
          serves "bravo/local"
```

## Delete a Host

This section explains how to delete a host from the system.

If a host is removed from the system, all domains whose databases are served from that host are affected.

In the simplest case, a host only serves a local domain. Therefore, only two steps are required: remove the downlink from the parent domain, then remove the database.

It is possible, though, that a host serves several domains. In this case, each database that is located on the host, must be removed and all binding links have to be changed. This can be a very complex process, and so all care should be taken before removing a host from the system. If none of the domains served from the host are required, then they can all be deleted, but all downlinks from parent domains must be reset.

*Example:*

*Delete a host*

This example deletes the host *charlie* from the following network.

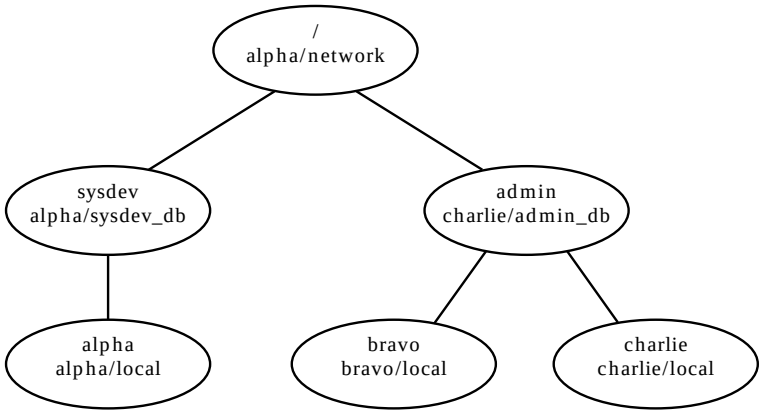



*Figure 5.14 - Deleting a Host*

As can be seen, two domains, **admin** and the local domain, are served from *charlie*. The other child domain, **bravo**, is also affected, as is the parent, the root domain. The binding links between these two will have to be changed.

In this example, we want to retain the **admin** and **bravo** domains. The host is being removed as it no longer functions satisfactorily. In reality, a new host may replace the old one, but in this example, the host *bravo* will take over as the server of the **admin** domain. This means that the existing ***admin_db*** database has to be copied to the new host, and all binding has to be changed.

The new structure is as follows:

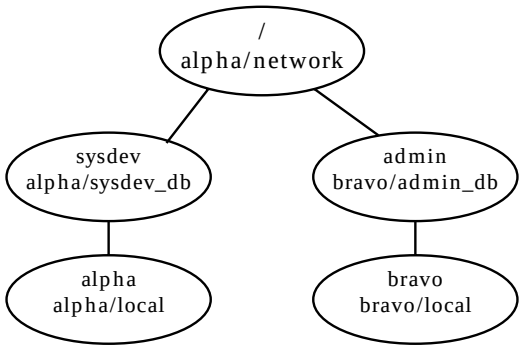



*Figure 5.15 - Deleting a Host*

Steps

1: Copy the admin_db database from charlie to bravo

2: Change the binding from root to the admin domain on bravo

3: Change the binding from admin to the local domain.

4: Remove the domains on charlie and disconnect it from the network.

This example involves three hosts *alpha*, *charlie* and *bravo*.

*Copy existing database*

The database on *charlie*, stored in the subdirectory `/etc/netinfo/admin_db.nidb`, must be copied across to the host *bravo*, retaining the same name. Make sure that the files in this directory are copied as well. Use any available remote copy utility to do this.

*Bind to parent*

The root domain (***network*** database) must be altered to reflect the change in hosts. As **admin** was served from *charlie*, the network database would have had a **/machines/charlie** entry. This is no longer required. Its child, **admin**, is now served from host *bravo*, so a new machines entry has to be created.

Create the downlink (**root** to **admin**)

```
niutil -destroy / /machines/charlie
niutil -create / /machines/bravo
niutil -createprop / /machines/bravo
     ip_address “192.42.172.3”
niutil -createprop / /machines/bravo
     serves “admin/admin_db”
```

Create the uplink (**admin** to **root**)

As the binding has changed, you will have to use the tag instead of the domain name. The **/machines/alpha** subdirectory will have to be created.

```
niutil -destroy -t bravo/admin_db /machines/charlie
niutil -create -t bravo/admin_db /machines/alpha
niutil -createprop -t bravo/admin_db /machines/alpha
     ip_address "192.42.172.1"
niutil -mergeprop -t bravo/admin_db /machines/alpha
     serves "../network"
```

*Bind to child*

The **admin** domain had a child domain on *bravo*. This connection should still be correct. Check the serves property entries in the **admin** database.

```
niutil -list -t bravo/admin_db /machines
niutil -read -t bravo/admin_db /machines/alpha
niutil -read -t bravo/admin_db /machines/bravo
alpha
     name: alpha
     ip_address: 192.42.172.1
     serves: ../network
bravo
     name: bravo
     ip_address: 192.42.172.2
     serves: bravo/local
```

The child domain, local on bravo, previously pointed to host charlie as its parent. This must be changed to bravo.

Change the local **bravo** domain.

```
niutil -destroy -t bravo/local /machines/charlie
niutil -mergeprop -t bravo/local /machines/bravo
     serves "../admin_db"
```

Check the serves properties of this database:

```
niutil -list -t bravo/local /machines
niutil -read -t bravo/admin_db /machines/bravo
bravo
     name: bravo
     ip_address: 192.42.172.2
     serves: ./local  ../admin_db
```

*Remove databases from old host*

Login as the superuser on host *charlie* to remove the old databases.

```
nidomain -d admin_db
```

```
nidomain -d local
```

These commands should remove the database files and stop the **netinfod** processes. The system can now be disconnected from the network.

# Managing Users and Groups

User and group accounts are created, and the users given access to domains by entering information into the database.

**Users**

User account information must be entered in subdirectory called **/users**. For each user who has access to the resources of the domain, a subdirectory must exist with the same name as the user's login name.

The subdirectory can have the following properties:

| | |
|---|---|
| name | The compulsory directory name must correspond to the users login name. |
| passwd | The value of the password can be left null. |
| uid | Each user must have a unique user identification number. |
| gid | Each user must belong to a group. The group identification number must exist in the **/group** subdirectory. |
| real_name | The user's real name can be entered. |
| home | The value of this property is a UNIX file system directory specifying the user's home directory. |
| shell | The value of this property should be the full pathname of the user's login shell program. |

The **/users** subdirectory can be automatically loaded with information from the flat file /etc/passed using the niload utility.

### Domain Access

If a user has an entry in a domain, then that user can access all its child domains.

*Example:*

*Creating user accounts*

Two users, ”chris” and ”jo”, must be able to login to the system development machines, *alpha* and *bravo*. These two machines are both members of the **sysdev** domain, therefore, the users should be registered as valid users in this domain.

Login to the host which serves the **sysdev** domain (*alpha*). First create the **/users** directory if it doesn't already exist. Use **niutil -list** to check this.

```
niutil -create -t sysdev/sysdev_db /users
```

Create the subdirectory for a user.

```
niutil -create -t sysdev/sysdev_db /users/chris
```

Create the properties and values for this user (the name property is created when the subdirectory is created).

```
niutil -createprop -t  sysdev/sysdev_db /users/chris passwd “”
niutil -createprop -t  sysdev/sysdev_db /users/chris uid 100
niutil -createprop -t  sysdev/sysdev_db /users/chris gid 50
```

Repeat the process for each user:

```
niutil -create -t sysdev/sysdev_db /users/jo
```

```
niutil -createprop -t  sysdev/sysdev_db /users/jo passwd “”
niutil -createprop -t  sysdev/sysdev_db /users/jo uid 101
niutil -createprop -t  sysdev/sysdev_db /users/jo gid 50
```

These two users should now be able to login to all hosts connected to the **sysdev** domain.

*Example:*

*Creating accounts in the root domain*

Another user, the general manager, whose login name is “genman”, must have access to all machines. Therefore, genman should be a registered user of the root domain.

Login to the host which serves the **root** domain (*alpha*). First create the **/users** directory if it doesn't already exist. Use **niutil -list** to check this.

```
niutil -create / /users
```

Create the subdirectory for the user “genman”.

```
niutil -create / /users/genman
```

```
niutil -createprop / /users/genman passwd “”
niutil -createprop / /users/genman uid 110
niutil -createprop / /users/genman gid 30
```

## Groups

Information about user groups must be entered in subdirectory called **/group**. For each group in the system, a subdirectory must exist with the same name as the group name.

The subdirectory can have the following properties:

| | |
|---|---|
| name | The compulsory directory name must correspond to the group name. |
| passwd | The value of the password is usually left as null for group access. |
| gid | Each group must have a unique group identification number. |
| users | This property contains a list of all users who are members of this group. |

# Chapter 7

# Maintenance

## Network Administration

All binding of domains into the hierarchy occurs when the **nibindd** daemon is started. This daemon must be running on each host in the network that wishes to use NetInfo. When **nibindd** starts, it searches the `/etc/netinfo` directory for databases. For each one found, it binds it into the hierarchy and launches a **netinfod** daemon to access the database.

Each domain knows who its parent is, but not its own name. If its parent exists on another host, then that host must be alive for the new host and domains to be bound correctly. In a nutshell, all ancestors of a new domain must be alive before the new domain can be bound into the hierarchy.

Usually, the **nibindd** daemon is started at boot time.

**NetInfo Start-up**

Two processes must be running to operate NetInfo: **nibindd** and **niypd**. These processes are normally started at boot time: see the *Software Installation* chapter of the Installation Guide for your particular server platform for an example rc.local script.

These two processes must be started on each host in the network. Use the **nips** utility, if it has been loaded, or any available **ps** program, to check what processes are running.

One function of the **nibindd** daemon is to read the database directory and bind all domains into the hierarchy. A child domain does not know its own name, only its parent does, therefore the parent must be alive before the child domain is bound.

If all domains are stored on the one machine, this will happen automatically. However, in a network of machines, the order of start-up is crucial.

The administrator should ensure that the root domain is located on a highly available machine. If network operation is crucial to the organisation, then a clone domain should also exist.

*Example:*

*Start-up Order*

Suppose the following domain structure exists:

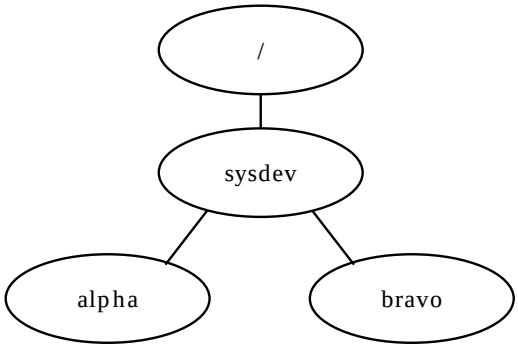



*Figure 6.1 - Start-up Order*

In this example, there are two machines, *alpha* and *bravo*: the root domain, "**/**", and the domain **sysdev** are both located on host *alpha*. *alpha* and *bravo* both have local domains.

NetInfo on *alpha* must be started before NetInfo on *bravo*. On *alpha*, three domains will be bound into the hierarchy. When *bravo* is then started, the local database will find its parent, **/sysdev**, on *alpha*.

**NetInfo Shut-down**

NetInfo can be shutdown gracefully using **nistop**. This is a supplied script that administrators may use or adjust to suit their environment. It shuts down each **netinfod** daemon, serving a database on the host, as well as the **nibindd** and **niypd** daemons.

## User Maintenance

Users must be able to change their own password without requiring super-user access. A supplied program, called **nipasswd**, that allows users to do this.

### Using nipasswd

The password value is encrypted on the screen when displayed or written using the **niutil** program. It is possible to change the value of this property using **niutil**, but the command line tools echo the value as they are typed in, and it will not be encrypted.

**nipasswd** allows users to change their own password. It works like the UNIX **passwd** tool: it prompts for the old and new passwords, without echoing them on the screen, encrypts and validates them, and updates the password information only if the responses are valid. Users must have "_writers" access to their own password property in order to change their password.

#### "_writers" Property

Normally, only **root** has write access to information in the NetInfo databases. Users can be granted permission to change the values of specific properties or variables by applying the "_writers" property. In order to change their own password, users must have "_writers" access to their password property.

The "_writers" property has one or more values: The values are login names of those users who are able to *write* to the property or directory. There are two forms:

| | |
|---|---|
| "_writers" | This form allows users named in the list of values to write to all properties in the directory. |
| "_writers_*propkey*" | This form allows users named in the list of values to write to the specified property only. |

The value of the "_writers" property (list of usernames) can take the value "*". This is a wildcard meaning all users of the domain. The wildcard specification is generally used when defining access to printers in a domain.

*Example:*

*Changing a Password*

The properties of a **/users** subdirectory correspond to the fields in the /etc/passwd file.

*Directory*: **/users/chris**

| Property | Value |
|---|---|
| **name** | chris |
| **passwd** | AX#1@2Tf |
| **uid** | 101 |
| **gid** | 10 |
| **real name** | Chris Smith |
| **home directory** | /usr/staff/chris |
| **shell** | /bin/csh |
| **_writers_passwd** | chris |

The "_writers_passwd" entry allows the user specified, **chris**, to write, and therefore, change, the password property, **passwd**, in the directory, **/users/chris.**

## Backup

The system administrator should ensure that a backup of the programs is kept, and the NetInfo databases are backed up on a regular basis. It is up to the administrator to choose their favourite backup strategy and medium. The following files comprise the NetInfo system:

**Programs**

Some NetInfo programs must be located in `/usr/etc`. Other utility programs are stored in locations at the discretion of the administrator, though usually in `/usr/bin`. A backup of all these programs should be made initially.

| | Program | Directory | Notes |
|---|---|---|---|
| *Compulsory Locations* | **nibindd** | `/usr/etc` | NetInfo daemon - must be running on all hosts. |
| | **niypd** | `/usr/etc` | NIS Emulation lookup daemon - must run in conjunction with **netinfod**. |
| | **netinfod** | `/usr/etc` | For each database in the `/etc/netinfo` directory, an instance of the netinfod daemon is started. This daemon is used to access the specified database. |
| | **nidomain** | `/usr/etc` | **nidomain** is used to create and destroy databases. When creating a database, it also starts a **netinfod** process to access the database. |
| *Utilities - Can be stored anywhere* | **niutil** | `/usr/bin` | **niutil** lists and maintains the contents of a database. |
| | **niload** | `/usr/bin` | **niload** adds data to the database from standard input. |
| | **nidump** | `/usr/bin` | **nidump** dumps data from the database to standard output. |
| | **nipasswd** | `/usr/bin` | **nipasswd** allows users to change their own password. |
| | **niwhich** | `/usr/bin` | **niwhich** shows which NetInfo hosts are served from which databases. This utility also shows the Internet address of the specified host. |

**Database Files**

All NetInfo administrative information is stored in directories within `/etc/netinfo`; *on each host* in the network. Each subdirectory here must be backed up on a regular basis. It is up to the administrator to decide on the backup strategy. Generally, these files need only be backed up as often as the NIS maps and the flat files are.

Standard configuration information can always be dumped from NetInfo into the flat files using the **nidump** utility. It is recommended that this be done for consistency, as these files are consulted during boot time.

## Security

Changes to the database can only be made by a user with root privileges (uid = 0). Therefore, all programs and the NetInfo database files must be owned by root.

It is recommended that the root user have read, write and execute privileges to the programs and database files, and that all other users have read and execute privilege only.

# Chapter 8

# Enterprise Edition Features

This chapter documents the additional features provided in the Enterprise Edition of NetInfo. Some of these features, such as hostname acquisition and automatic host addition, are also available in the Workgroup Edition.

In order to better support large networks of NEXTSTEP workstations, the NetInfo Enterprise Edition includes additional tools and has been modified to improve server performance.

The additional features provided are:

- Readall proxies
- RFC 1048 support
- NetInfo domain aliasing
- Host name acquisition
- Automatic host addition
- Support for diskless NEXTSTEP workstations
- Support for Multi-homed servers
- Performance enhancements

## Readall Proxies

The master can fork a child process to respond to a clone's readall request. This feature is only included in the Enterprise edition of NetInfo.

These children are called “readall proxies.” This is configured by setting the readall_proxies property in the domain's root directory. The property has up to two values: first, the maximum number of readall proxies; second, whether only a readall proxy may respond to a readall request (if the maxmium number of proxies are running), which is called “strict” proxies. If the number of proxies allowed is -1, an unlimited number of proxies will be used, up to the system's resource limitations. By default, no proxies are used; by default, if proxies are used, strict proxies will be used.

Clone now logs when a readall results in no database being sent (due to the clone being current), and when it results in the database being sent.

One important side-effect of using readall proxies is a change in the way modifications are handled when a master is replying to a readall request. Without proxies, the master won't handle the request, and won't reply to the request, until it's done replying to the readall. This would lead to read requests timing out; in conjunction with preferential binding and the way the client library handles reconnecting, a master's machine could appear catatonic until the readall completed because the lookupd or niypd on that machine is trying, continually, to contact the master. In the case of write requests, the client would, likely, time out waiting for the master to respond; the change was made, though, because it's in the master's request queue.

Rather than just hanging on the request, when readall proxies are used the master will be able to handle the request. If the request is a write of some sort, the master will reply with NI_MASTERBUSY.

## RFC1048 Support

nibootpd is now RFC1048-compliant: it can return netmask, router, log host (if the machine `loghost' exists) and DNS configuration information to clients that support the RFC1048 extensions (such as Windows NT).

To enable this, configure the NetInfo directories /locations/resolver, /locations/router, /locations/ntp, and /machines, as per the following example:

```
# First do the resolver entries - this format is the same as
# required for DNS in NS3.3 (replaces the /etc/resolv.conf file)
#
niutil -create     / /locations/resolver
niutil -createprop / /locations/resolver domain pcp.ca
niutil -createprop / /locations/resolver nameserver
xxx.xxx.xxx.xxx xxx.xxx.xxx.xxx
#
# Now for the router entry.  This is specific to the RFC1048 bootpd
#
niutil -create     / /locations/router
niutil -createprop / /locations/router address xxx.xxx.xxx.xxx
xxx.xxx.xxx.xxx
#
# Put the netmask info in the /machines entry (also RFC1048 bootpd
specific)
#
niutil -createprop / /machines netmask 255.255.255.0
#
# Create the entry for the time server
```

```
#
niutil -create     / /locations/ntp
niutil -createprop / /locations/ntp host ntpserver
```

Note how the domainname is appended to the hostname (foobar) in the HN field.

You can pass vendor-specific information using the /locations/bootpd directory. Create a property of the format Txxx where xxx is the vendor tag number. For example, to pass the string `hpnp/laser.cfg' for vendor tag 144, you would create the following properties. (Note that the double quotes must be embedded in the property value; otherwise, the value will be interpreted as a hexadecimal value.)

```
niutil -create     / /locations/bootpd
niutil -createprop / /locations/bootpd T144 '"hpnp/laser.cfg"'
```

## NetInfo Domain Aliasing

nibindd now supports netinfod aliasing, allowing multiple servers on the same machine potentially to respond to parent requests for the same tag. This is particularly useful for multi-homed machines (which NEXTSTEP does not currently support). The modification allows the binder daemon to substitute some other NetInfo server (an “aliased” server) for the server that was requested by a client . For instance, if a client (NetInfo server for tag local) requests a given tag (network), then the binder daemon passes the request off to some other server (netinfod network-33 perhaps). The criteria for deciding which requests (they're all for network) get passed to which server is based on some additional NetInfo properties.

The extra information resides as properties in the domain's root directory, using the alias_name and alias_addrs properties. The former specifies the alias to which this domain will respond; the latter is a series of values containing the address and netmask of clients which should be referred to this domain when the alias name is requested. The address and netmask are one value, separated by an ampersand (&). For example:

```
% niutil -read -t mustang/network-41 /
master: mustang/network-test
alias_name: network
alias_addrs: 192.42.172.0&255.255.255.0
192.42.173.0&255.255.255.192
```

This says that any requests for tag network from machines with IP addresses that match either 192.42.172 in the high-order 24 bits or that match 192.42.173.0 in the high-order 26 bits should get handled as if they were asking for tag network-test.

You may want to set the restrict_multi_homed property in the root directory such that broadcasting on multihomed servers will be done properly; see the multi-homing section of this manual for more information on this property.

## Hostname Acquisition

The Enterprise Edition of NetInfo provides support for NEXTSTEP workstations that need to acquire their hostnames and IP address information at boot time.

This is a standard feature of the NEXTSTEP workstation that allows for greater flexibility in moving equipment from place to place and changing network addresses and hostnames. It allows a system administrator to change the hostname and IP address of any NEXTSTEP workstation centrally, using NetInfo, rather than by editing configuration files stored on each and every workstation in the network.

This facility is implemented through the BOOTP protocol and the associated server processes **nibootpd**(8) and **nibootparamd**(8).

By following the installation procedures documented in this Guide, the necessary software will have been installed for your server to provide this service to NEXTSTEP workstations.

*How Hostname Acquisition Works*

When a NEXTSTEP workstation boots, it checks the file `/etc/hostconfig`, to see whether it has a fixed hostname and IP address specified.

If the `/etc/hostconfig` file specifies that the hostname and IP address of the workstation are **automatic**, then the NEXTSTEP system will begin looking for a BOOTP server on the network, providing its hardware Ethernet address as a lookup key. Hardware Ethernet addresses are, by definition, unique.

The supplied **nibootpd** program running on a server will receive the lookup request and consult NetInfo to determine the hostname and IP address of the workstation in question. This information is then returned to the NEXTSTEP workstation, which configures itself appropriately.

If the information cannot be found in NetInfo by **nibootpd**, then the NEXTSTEP workstation will initiate Automatic Host Addition. The following section describes this feature in detail.

*Information stored in NetInfo*

**nibootpd** obtains the hostname and IP address for the NEXTSTEP workstation from the **`/machines`** directories in the NetInfo database.

For hosts which are configured in this fashion, an extra property must exist: the value of which is the hardware Ethernet address of the machine.

*Example*

For example, the host *bravo* has a hardware address of 00:00:0f:01:0d:dc

This value is stored in the **`/machines/bravo`** directory as the property **`en_address`**:

```
/machines/bravo
                name            bravo
                ip_address      192.42.172.2
                en_address      00:00:0f:01:0d:dc
                serves          ./local
                bootfile        mach
```

**nibootpd**
*search strategy*

When a NEXTSTEP workstation starts up, **nibootpd** searches all the **/machines** subdirectories in the NetInfo system for a **/machines/host/en_address** property value that matches the workstation Ethernet address.

When it finds the matching directory, it responds with the **name** and the **ip_address** properties of the directory as the host name and IP address for the NEXTSTEP workstation to use.

**Loading bootp information into NetInfo**

You can create the **en_address** properties by hand using **niutil**. However, there are two other, easier alternatives.

- Create a `bootptab` format file and use **niload** to read its contents into NetInfo.

or

- Use the Automatic Host Addition features described in the next section, and have each NEXTSTEP workstation configure itself the first time it boots.

**Using niload**

The **niload** command is capable of reading a `bootptab` format file.

`bootptab` is the traditional UNIX file used for managing BOOTP information and contains lines of the following format:

| # hosthtype | haddr | iaddr | boot file |
|---|---|---|---|
| bravo1 | 00:00:0f:01:0d:dc | 192.42.172.2 | mach |

An example `bootptab` file was installed in your `/etc` directory by the **install_netinfo** script.

**niload** can be used to read a `bootptab` file as follows:

```
# niload bootptab / < /etc/bootptab
```

This command will cause **niload** to read all the entries in the file `/etc/bootptab` and create **en_address** and **bootfile** properties for each and every hostname encountered. The entries will be made in the **root** domain, "/".

You can also add these properties using **niutil**:

```
# niutil -createprop / /machines/bravo en_address 0:0:f:1:d:dc
```

where the entry is also to be made in the root domain, "/".

The **bootfile** property is used during diskless booting to identify the file which contains the image of the NeXT kernel, **mach**. See the section on support for diskless workstations for more information.

## Automatic Host Addition

Another feature provided by the Enterprise Edition of NetInfo is support for NeXT's technique for adding new NEXTSTEP workstations to a network.

As described above under Host Name Acquisition, a NEXTSTEP workstation can be configured to acquire its host name and IP address from a NetInfo based server on the network.

In the case where the NEXTSTEP workstation is already known and has information stored in NetInfo, this results in the workstation using the NetInfo supplied data.

However, if the hardware address of the NEXTSTEP workstation is unknown to NetInfo, the Automatic Host Addition feature comes into play.

*How Automatic Host Addition works*

When a NEXTSTEP workstation boots, it checks the file `/etc/hostconfig`, to see whether it has a fixed hostname and IP address specified.

If the `/etc/hostconfig` file specifies that the hostname and IP address of the workstation are **automatic**, then the NEXTSTEP system will begin looking for a BOOTP server on the network, providing its hardware Ethernet address as a lookup key.

The **nibootpd** program running on a server will receive the lookup request and will consult NetInfo to determine the hostname and IP address of the workstation in question.

If this workstation is unknown to NetInfo, the user of the NEXTSTEP workstation will be told that the "Network does not recognise this computer" and will be asked to enter a hostname for the new system.

This hostname will then be relayed to the **nibootpd** server which will allocate an unused IP address and assign it to the NEXTSTEP workstation in question. This information will then be updated in NetInfo automatically, without requiring an administrator to maintain any configuration files.

**nibootpd** makes an entry in NetInfo for the new host. This new entry will include the Ethernet hardware address of the new workstation as well as the hostname and the allocated IP address, removing the need to update it by hand later on.

*Information stored in NetInfo*

**nibootpd** allocates the new workstation an IP address. It determines which IP address to use based on two new properties of the **/machines** directory in the server's local domain.

```
/machines
  name machines
  assignable_ipaddr192.42.172.10
       192.42.172.250
  configuration_ipaddr192.42.172.253
```

These properties are properties of the **/machines** directory itself, not of a subdirectory of the **/machines** directory. You can read them by using **niutil** as follows:

```
# niutil -read . /machines
```

The first property, **assignable_ipaddr**, has two values and describes a range of IP addresses available for allocation by **nibootpd**. In our example, the **nibootpd** server could allocate addresses from 192.42.172.10 through 192.42.172.250 inclusive.

The second property, **configuration_ipaddr**, is required and specifies the address that must **not** be allocated by **nibootpd**. This address is in fact the address that NeXT uses to identify a new workstation temporarily during the boot process. It should **always** be set to 192.42.172.253 explicitly.

Note that **nibootpd** will search the range of IP addresses specified when allocating a new address to ensure that the address chosen is not already in use.

These values can be set with command lines as follows:

```
# niutil -createprop . /machines assignable_ipaddr 192.42.172.10  192.42.172.250

# niutil -createprop . /machines configuration_ipaddr 192.42.172.253
```

where the entry is made in the server's **local** domain, "."

## Support for Diskless Workstations

In addition to the hostname and autoconfiguration features, support is provided for diskless NEXTSTEP workstations, and for workstations that may have a local disk, but use it only for swap storage purposes.

To boot a NEXTSTEP workstation diskless using a NetInfo server as the disk server, you must ensure that the usual procedures to allow a workstation to boot diskless from the server have been observed. This includes setting up the necessary **tftp**(1C) directories on the server, stocking them with the appropriate kernel images (**mach** for NEXTSTEP) and ensuring that the required NFS mount facilities have been put in place.

Please see your server documentation, supplied by your server vendor, for details of the required configuration.

Diskless booting support is provided through the server processes, **bootparamd** and **bootpd**. **bootparamd** is a standard tool supplied with most UNIX systems, however, the standard **bootparamd** does not use NetInfo based information.

Provided with this Enterprise Edition of NetInfo are new versions of **bootpd** and **bootparamd**, called **nibootpd** and **nibootparamd** respectively. These are enhanced to read information from NetInfo. If you followed the install procedures documented in the Installation chapter of this Guide, these programs have been installed in your `/usr/etc` directory.

*How Diskless Workstations work*

When a NEXTSTEP workstation tries to boot as a diskless workstation, it first downloads an image of the Mach kernel via the trivial file transfer protocol, **tftp**(1C).

In order to determine the host to load this kernel from and the file that contains the kernel, it makes a broadcast request looking for a BOOTP server.

**nibootpd** responds by looking up the Ethernet hardware address of the requesting workstation and returning the IP address, hotsname and kernel file name found in NetInfo.

Hence, you must ensure that you have updated the `bootptab` information in NetInfo as well as following the instructions herein. See the earlier section on Hostname acquisition for a description of the `bootptab` file and NetInfo support.

After determining its hostname and IP address, the workstation then uses the bootparams protocol to determine the NFS server it should mount its **root** and **private** filesystems from.

This information is provided by the **nibootparamd** server and is also stored in NetInfo for ease of administration.

*Information stored in NetInfo*

**nibootpd** and **nibootparamd** obtains this information from the **`/machines`** directories in the NetInfo database by looking up the directory with the same **`name`** as the NEXTSTEP workstation that wishes to boot diskless.

**nibootpd** reads the **`en_address`** and **`bootfile`** properties, as discussed earlier in the section on Hostname Acquisition.

**nibootparamd** reads the **bootparams** property, another new property for the Enterprise Edition.

This property is stored in the **/machines/bravo** directory as the multi-valued property **bootparams**.

For example, to allow the machine *bravo* to boot diskless from the server *alpha*:

```
/machines/bravo
   name           bravo
   ip_address     192.42.172.2
   en_address     00:00:0f:01:0d:dc
   serves         ./local
   bootfile       mach
   bootparams     root=alpha:/
                  private=alpha:/next_area/private
```

This entry tells **nibootpd** that the machine with Ethernet address 00:00:0f:01:0d:dc is named *bravo*, has IP address 192.42.172.2 and should use the kernel image stored in the file *mach*.

This entry also tells **nibootparamd** that the machine *bravo* should mount its root file system, /, by NFS mounting the root file system of *alpha*, and that it should mount its /private file system by NFS mounting the directory /next_area/private from the host *alpha*.

**Loading bootparams information into NetInfo**

You can create the **bootparams** properties by hand using **niutil** if you wish:

```
# niutil -createprop / /machines/bravo bootparams root=alpha:/
private=alpha:/next_area/private
```

where the entry is made in the root domain, "/".

You can also use **niload** to read a bootparams(5) format file.

**Using niload**

The **niload** command is capable of reading a bootparams(5) format file.

bootparams is the traditional UNIX file used for managing **bootparamd** information and contains lines of the following format:

```
# host key=valuekey=value

  bravoroot=alpha:/private=alpha:/next_area/private
```

An example bootparams file was installed in your /etc directory by the **install_netinfo** script.

**niload** can be used to read a bootparams file as follows:

```
# niload bootparams / < /etc/bootparams
```

This command causes **niload** to read all the entries in the file /etc/bootparams and create **bootparams** properties for each and every host name encountered, in the **root** domain, "/".

## Support for Multi-homed Servers

NetInfo Enterprise Edition provides explicit support for servers that have more than one active Ethernet interface.

This support is, for the most part, invisible to the user, in that it has been implemented as internal enhancements in the NetInfo server software.

However, one of the enhancements made has a special use, which can ease configuration for some customers.

Broadcasthost and 255.255.255.255

Support has been added for the special broadcast token, 255.255.255.255.

If a host is specified with this address, NetInfo will broadcast on all available Ethernet interfaces when attempting to find the given host on the network.

This is useful in the particular case of specifying the host which serves the **parent** domain of a given domain.

For example, if the following entry exists in the *local* database:

```
/machines/broadcasthost
   name            broadcasthost
   ip_address      255.255.255.255
   serves          ../network
```

NetInfo will broadcast on all Ethernet interfaces looking for a host which serves a database with the tag *network*.

This is a particularly useful feature as it allows you to re-configure your network without necessarily altering your NetInfo bindings.

This particular feature is also present in the Workstation Edition of NetInfo Version 1.04 or later, and is also a standard feature of NetInfo as provided by NeXT Computer, Inc.

## Performance Enhancements

NetInfo Enterprise Edition integrates all of the enhancements to the core NetInfo product provided by NeXT Computer, Inc. as part of the NeXTSTEP 3.0 release.

These changes, whilst relatively minor in nature, result in some significant performance improvements in large networks with many domains and when the amount of data stored in NetInfo is great.

The enhancements are:

- Improved clone/server propagation.
- File format optimisations.
- Smarter binding of parent servers to accommodate slower network links.

**Clone propagation**

The protocol between NetInfo master and clone servers has been improved to eliminate excessive data transfers, and the distribution of NetInfo changes has been streamlined.

Both these changes result in improved clone performance and reduce the traffic required to propagate changes to clone servers.

*Details*

When a number of NetInfo changes are made in succession, as often occurs when data is loaded with **niload**, **netinfod** will coalesce the individual changes into a single composite update transaction. Since only one transaction with each clone server is needed to distribute the composite update, the overhead associated with updating clones is reduced.

In addition, the distribution of updates is now multi-threaded. Modifications made to the master database are immediately distributed to clones; an update need not wait for prior updates to finish before it is handled. Update threads operate independently, except that updates are guaranteed to arrive at each clone in the correct order.

**File format**

The disk format for NetInfo databases has changed to increase the default record size. In most cases, this reduces the number of files in a database by about 90% and decreases the cost of large searches by 25%.

These changes have been made to **netinfod**, and support is provided for both the new and the old database formats.

Also, all versions of NetInfo can be used together on a network, regardless of the revision level or database format of the individual servers. All configurations, including master and clone servers which use different versions of software and different database formats, are supported.

Note that a Enterprise Edition database cannot be directly copied to a Workstation Edition server due to this difference in disk format. Use the **nidomain** command to create a clone database over the network instead.

**Smarter binding**

In order to deter NetInfo servers from binding to inappropriate parent servers over a slow network link, NetInfo servers now check for the existence of a local clone of the parent domain before broadcasting to find a suitable parent.

Also, by explicitly configuring the IP address of the machine that serves the parent domain, broadcasts can be eliminated altogether.

See also the discussion of support for Multi-homed Servers earlier in this section for other relevant information and the NEXTSTEP NetInfo 3.3 documentation for further details.

# Chapter 9

# NIS Emulation

This chapter documents the technique used for integration of NetInfo data with the UNIX operating system.

With SPARC SunOS 4.x and Netinfo Edtion 1.x, NetInfo data was integrated into the UNIX lookup mechasnism through modifications to the standard library, libc.so, in conjunction with the lookup daemon, **lookupd**. This is the same technique used by NeXT in the NEXTSTEP operating system.

With NetInfo Editions 2.x and above, this technique is no longer used.

Instead, this version of NetInfo provides a special server process to look up data, niypd. niypd appears to the UNIX system to be the NIS lookup process, ypserv.

This chapter discusses this technique in greater detail and provides important background information to enable you to better manage NetInfo.

## How NIS Emulation works

Programs running under the UNIX operating system access the information in the NetInfo database, if it is running, by accessing a new NIS server process, **niypd**, that emulates the NIS server, **ypserv**. NIS must be configured so that all NIS lookup queries generated by software on the local machine go first to the new server process.

This technique works in two steps as follows:

- run the NetInfo server process, **niypd**, on the local machine,
- configure the local machine so that NIS lookup queries generated by software go first to the new server process.

Installation is a matter of replacing the existing NIS server process, **ypserv**, with the provided process **niypd**. The supplied startup script does this at boot time by creating a symbolic link from /usr/sbin/ypserv to /usr/etc/niypd.

## Ensure NIS domains are correct

The key to this technique working is that you must also set up your NIS domain names correctly. **niypd** emulates NIS server routines to enable the UNIX environment to make NIS lookup calls as if they were NIS lookups. For this to work correctly, you must ensure that the local machine is a member of a NIS domain of one - that is, itself only.

This is necessary to provide for the conceptual mapping of NetInfo domains to NIS domains. NIS is inherently a non-hierarchical system, where many machines all draw their data from a single NIS database.

NetInfo provides a fully hierarchical database system, where each machine has a local database for its own data but also accesses data in 'parent' databases up the hierarchy.

To preserve this concept of local data, you must create a new NIS domain for each of your machines that is private to that machine only.

*Example*

For example, say you have an existing NIS domain `abc.com.au`. All of your machines are currently located within this domain and therefore draw their NIS data from it. The NIS database is served by the **ypserv** process running on your central server machine, *alpha*.

To move to NetInfo, you would need to use the **domainname** command to change the domain name on each and every machine that is to become a NetInfo client. The simplest way to do this is to use the hostname as the first part of the individualised domain name.

For example:

> The machine *alpha* in `abc.com.au` has its NIS domain name changed to `alpha.abc.com.au`

Having done this, you can then run the **niypd** process on *alpha* instead of **ypserv**, and **niypd** will start up as though it were the NIS server for the domain `alpha.abc.com.au`. Each machine that you run **niypd** on will be similarly configured.

## Using NIS as well as NetInfo

Since **niypd** emulates NIS calls for a given (localised) domain, the question arises of how to get your local machine to still use NIS as a backup. That is, if a lookup query cannot find matching data in NetInfo, you would like it to somehow continue the query in the old NIS system.

This is possible as **niypd** has been enhanced to provide a 'fall-through' mechanism to an existing NIS setup. To use this feature,you can pass **niypd** the domain name of the 'parent' NIS domain on the command line.

*Example*

To start up niypd with the domain abc.com.au as the parent NIS domain:

```
# niypd abc.com.au
```

If the `domain_name` parameter is not provided on the command line, the parent domain is derived by removing the first component of the local domain name.

*Example*

The local domain is `alpha.abc.com.au`

If you use:

```
# niypd
```

then the parent domain will be `abc.com`.au. If you want to force a different parent domain, you must specify the parent domain explicitly as in the first example above.

Either usage will cause **niypd** to emulate all NIS queries for the local NIS domain (`alpha.abc.com.au`) and then to 'fall-through' and make queries on the parent NIS domain `abc.com.au` if it should not find the result data in NetInfo.

## Using NetInfo *without* using NIS

If you only want to use NetInfo for your system, then you can tell the **niypd not** to fall through to NIS at all. To do this, start up **niypd** with the special `local` domain name.

```
# niypd local
```

This tells **niypd** that it is only to perform localised NetInfo queries and should not use NIS for fall backs.

Remember, the standard installation scripts provided link **ypserv** to **niypd** at boot time. Edit these scripts if you need to pass a parameter to **niypd** as described here.

## Mapping NetInfo data to NIS maps

NIS stores all of its data in databases called 'maps'. A single NIS map contains the password file, another contains the hosts file. These files are simple dbm databases.

NetInfo is very different in internal structure, with each database in a NetInfo domain hierarchy having a tree-like structure internally. A single NetInfo database can contain data for passwords, hosts, and printers and many other (even user-defined) data types.

In order to take a NIS lookup query and make it a valid query in the NetInfo context, it is necessary for the **niypd** server to emulate a series of NIS 'maps'. Each map is a logical name for a collection of NetInfo data, and the **niypd** process takes care of the conversion of the NIS style map calls to NetInfo queries.

The following maps are implemented by **niypd**, being almost all of the maps supported by NIS as standard.

```
passwd.byname
passwd.byuid
group.byname
group.bygid
hosts.byname
hosts.byaddr
services.byname
services.byport
services.bynameproto
mail.aliases
networks.byaddr
networks.byname
protocols.bynumber
protocols.byname
rpc.byname
rpc.bynumber
```

The following maps are NOT implemented at this time:

```
netid
netmask
ethers
bootparams
```

Note that even though the `bootparams` map is not implemented by **niypd**, **bootparamd** still uses NetInfo directly, as a modified **bootparamd**, called **nibootparamd**, is supplied with the system.

**Using NIS tools**

Since **niypd** emulates these maps, you can use the standard NIS management tools **ypcat** and **ypmatch** to dump data from NetInfo.

For example:

```
# ypcat passwd
.. passwd data from all sources ...
```

This command will use **niypd** to dump out all of the password data - first the local data and then the data all the way up the NetInfo hierarchy, followed then by any password data from the 'parent' NIS domain. Remember, **niypd** falls through to NIS if it can't find data to resolve a query, and in the case of **ypcat**, the query is asking for all data - until there is no more to tell.

Similarly **ypmatch** allows you to find a keyed record in the combined NetInfo / NIS databases.

NetInfo is searched first in all cases:

For example:

```
# ypmatch john passwd
.. passwd record for user john ...
```

If you're ever unsure about which record will be found first, **ypmatch** will give you the definitive answer. This is particularly useful when you have a duplicated record in higher levels of the NetInfo database, but have a local definition to 'override' the common definition. **ypmatch** allows you to confirm which record is being found first.

## Caveats

### Installation

Because NetInfo queries work through the NIS lookup mechanism, it becomes necessary for you to perform the same installation as you would for NIS **before** installing NetInfo. Even if you don't want to use NIS following your NetInfo installation you must still install your operating system options to provide for NIS support.

Also, you should be careful to ensure that you do the things necessary to cause your operating system to make NIS lookup calls in the first place.

For example, you must ensure that you add the infamous NIS '+' lines to the end of the password and groups files. This is the magic token that NIS relies on in order for it to be invoked when query for password or groups.

Also, on certain systems (DEC UNIX and Solaris 2.x for example), you must also configure the `/etc/svc.conf` file to specify that you wish to use NIS data and the order in which data is to bea searched.

As with all NetInfo installations, it is recommended that you only keep the bare minimum number of entries in any of your `/etc` files, and that you put all other data in NetInfo.

Functions which are not required, or make no sense in the NetInfo environment, such as **ypxfr**, are not implemented by **niypd**. You can safely disable these processes.

### yppasswd

You should use the new tool **nipasswd** to change users passwords in NetInfo. The standard **yppasswd** tool that works for NIS installations uses special NIS protocols that are not supported by this release of the NetInfo emulation system.

### Search order

The search order using NIS Emulation is different from the search order documented in previous NetInfo Edition technotes, and differs from the search order currently used in the SPARC Edition of NetInfo.

In these earlier, lookupd based Editions, the search order was NetInfo first, followed by the flat files, followed by NIS.

In the NIS Emulation based versions of NetInfo, where data would normally come from NIS, it comes from NetInfo followed immediately by NIS. Thus, if you configure `/etc/svc.conf` .to search local files then NIS, you will get local files, NetInfo, NIS as the search order.

# Chapter 10
# NetInfo Reference

The *Reference* Section provides manual page entries for the NetInfo programs. These programs are also stored on-line and can be accessed using the **man** utility.

The following programs are described:

- bootparams(5)
- netinfo(5)
- netinfod(8)
- nibindd(8)
- nibootparamd(8)
- nibootpd(8)
- nidomain(8)
- nidump(8)
- nifind(1)
- nigrep(1)
- niload(8)
- nipasswd(1)
- nireport(1)
- niutil(8)
- niwhich(1)
- niypd(8)

# Overview of NetInfo Programs

The following command line tools operate on the NetInfo database. The databases must be located in the `/etc/netinfo` directory. NetInfo is running when the **nibindd** and **niypd** daemon processes are running. Normally, these processes are started when the machine is booted.

### Daemons

There are several daemon processes associated with NetInfo.

**nibindd** The **nibindd** daemon must be running on each host in order for NetInfo to be used. If this daemon is not running, then the UNIX system will use NIS (yellow pages) if NIS is running. If not, the flat configuration files are consulted. Normally, this daemon is started up at boot time by an appropriate entry in the `rc.local` file.

**niypd** **niypd** handles requests for information from clients and passes them to **nibindd**. It must be running in conjunction with **nibindd** to use information in the NetInfo database.

**niypd** is a drop in replacement for **ypserv**, that translates NIS lookups to NetInfo lookups.

**niypd** currently does *not* cache requests. This is planned for a future release.

**netinfod** A **netinfod** daemon is started by **nibindd** for each domain known to the network hierarchy. **nibindd** checks the `/etc/netinfo` directory and starts a **netinfod** process for each database directory it finds.

**netinfod** is forked by **nibindd**. It should not be started by hand at any time, as **nibindd** handles the binding of domains into the hierarchy.

### Loading and Dumping Database Information

Two programs are supplied with the system in order to load information into and dump it out of the various NetInfo databases. Use these tools to keep the flat files up to date with the NetInfo database in case a machine has to be run without connecting to the network.

**niload** **niload** can be used to load information such as password files, into the database. The system understands the format of the flat configuration files and loads data according to the specified format.

**nidump** The dump program works in the opposite way to load. It copies the configuration information in the NetInfo into files formatted as specified.

**niload** and **nidump** understand the following formats:

| | | | |
|---|---|---|---|
| aliases | bootparams | bootptab | fstab |
| group | hosts | networks | passwd |
| printcap | protocols | rpc | services |

**niload** reads from standard input, and **nidump** writes to standard output. Information can be copied easily using these tools together.

### Creating and Managing Databases

**nidomain**

**nidomain** is used to create and delete databases. This utility does not bind the domain into the hierarchy. Both the uplink (to parent) and the downlink (to any child domains) have to be specified by setting the values of the appropriate properties.

When the **create** option is requested (**-m**), **nidomain** performs several tasks.

- It creates a directory for the database in `/etc/netinfo`, with a database file (`collection`).
- It starts an instance of the **netinfod** daemon.
- It enters some information in the database including:

**`/machines`** directory with a subdirectory for the host on which the domain is created. Three properties are created: name (value = host); ip_address (value obtained from host information); and serves property (one property only - self served from local database).

In order to make changes to the database information, a root user must be specified. This in done by creating a **`/users`** directory with a subdirectory for the root user.

The **delete** option removes the directory and stops the daemon process.

**niwhich**

**niwhich** is used to determine which host serves a given NetInfo domain. It outputs the hostname, Internet address and the database tag for the domain.

### Managing Database Information and Properties

**niutil**

The NetInfo utility program, **niutil**, is used to view, create and maintain subdirectories, properties, and values of the domain database.

The **list** option displays a list of directories in the database. **create** creates directories, and **destroy** removes them.

**read** displays a list of properties and their values. **addval**, **destroyval**, **createprop** and **destroyprop** are used to manage the properties and values.

### Querying the NetInfo Database

**nifind**

Finds a directory in the NetInfo hierarchy. It starts at the local domain and climbs up through the hierarchy until it reaches the root domain.

**nigrep**

Searchs for a regular expression in the NetInfo hierarchy. It searches the domain's directory hierarchy depth-first starting from the root directory

**nireport**

Prints a table of values of properties in all sub-directories of the directory given on the command line

**Passwords**

**nipasswd** — **nipasswd** allows users to change their NetInfo password. It behaves like the UNIX **passwd** program: it prompts for the old and new passwords, and validates the change before making the alteration to the NetInfo database.

**bootparams(5)**

## NAME

bootparams - boot parameter data base

## SYNOPSIS

/etc/bootparams

## DESCRIPTION

The bootparams file contains the list of client entries that diskless clients use for booting. For each diskless client the entry should contain the following information:

name of client
a list of keys, names of servers, and pathnames.

The first item of each entry is the name of the diskless client. The subsequent item is a list of keys, names of servers, and pathnames.

Items are separated by TAB characters.

## EXAMPLE

Here is an example of the /etc/bootparams file:

```
myclient  root=myserver:/nfsroot/myclient \
swap=myserver:/nfsswap/myclient \
dump=myserver:/nfsdump/myclient
```

## FILES

/etc/bootparams

## SEE ALSO

**nibootparamd(8)**

**netinfo(5)**

## NAME

netinfo - network administrative information

## DESCRIPTION

NetInfo stores its administration information in a hierarchical database. The hierarchy is composed of nodes called NetInfo directories. Each directory may have zero or more NetInfo properties associated with it. Each property has a name and zero or more values.

This man page describes those directories and properties which have meaning in the system as distributed. Users and third-parties may create other directories and properties, which of course cannot be described here.

### Search Policy

Virtually everything that utilises NetInfo for lookups adheres to the following convention. Search the local domain first. If found, return the answer. Otherwise, try the next level up and so on until the top of the domain hierarchy is reached. For compatibility with Yellow Pages and BIND, see **niypd(8)**.

### Database Format -Top Level

At the top level, the root directory contains a single property called “master”. This property indicates who is the master of this database, i.e., which server contains the master copy of the database.

The singular value of “master” contains two fields, a hostname and a domain tag separated by a '/' which uniquely identifies the machine and process serving as master of this data.

For example, the entry “eastman/network” says that the **netinfod(8)** process serving domain tag ***network*** on the machine *eastman* controls the master copy of the database.

For added security, a second property can be installed in the root directory to limit who can connect to the domain. By default, anybody can connect to the domain, which would allow them to read anything that is there (writes are protected however).

If this default is undesirable, a property called “trusted_networks” should be enabled in the root directory. Its values should be the network (or subnet) addresses which are assumed to contain trusted machines which are allowed to connect to the domain. Any other clients are assumed to be untrustworthy.

A name may be used instead of an address. If a name is given, then that name should be listed as a subdirectory of “**/networks**” within the same domain and resolve to the appropriate network address.

### Database Format -Second Level

At the second level, the following directories exist which have the following names (property named “name” has these values):

aliases
groups
machines
mounts

networks
printers
protocols
rpcs
services
users

These directories contain, for the most part, only the single property named "name".

The exception is the **/machines** directory which contains other properties having to do with automatic host installation. These properties are the following:

promiscuous — if it exists, the **bootpd(8)** daemon is promiscuous. Has no value.

assignable_ipaddr — a range of IP addresses to automatically assigned, specified with two values as endpoints.

configuration_ipaddr — the temporary IP address given to unknown machines in the process of booting.

default_bootfile — the default bootfile to assign to a new machine.

net_passwd — optional property. If it exists, it's the encrypted password for protecting automatic host installations.

The directory **/aliases** contains directories which refer to individual mailing aliases. The relevant properties are:

name — the name of the alias

members — a list of values, each of which is a member of this alias.

The directory **/groups** contains directories which refer to individual UNIX groups. The relevant properties are:

name — the name of the UNIX group

passwd — the associated password

gid — the associated group id

users — a list of values, each of which is a user who is a member of this UNIX group.

The directory **/machines** contains directories which refer to individual machines. The relevant properties are:

| | |
|---|---|
| name | the name of this machine. This property can have multiple values if the machine name has aliases. |
| ip_address | the Internet Protocol address of the machine. This property can have multiple values if the machine has multiple IP addresses. Note that the address MUST be stored in decimal-dot notation with no leading zeroes. |
| en_address | the Ethernet address of the machine. Note that the address MUST be stored in standard 6 field hex Ethernet notation, with no leading zeros. For example, “0:0:f:0:7:5a” is a valid Ethernet address, “00:00:0f:00:07:5a” is not. |
| serves | a list of values, each of which is information about which NetInfo domains this machine serves. Each value has the format *domain-name/domain-tag*. The domain name is the external name of the domain served by this machine as seen from this level of hierarchy. The domain tag is the internal name associated with the actual process on the machine that serves this information. |
| bootfile | the name of the kernel that this machine will use by default when NetBooting. |
| bootparams | a list of values, each of which is a Bootparams protocol key-value pair. For example, “root=eastman:/” has the Bootparams key “root” and Bootparams value “eastman:/”. |
| netgroups | a list of values, each of which is the name of a netgroup of which this machine is a member. |

The directory **/mounts** contains directories which refer to filesystems. The relevant properties are:

| | |
|---|---|
| name | the name of the filesytem. For example, “/dev/od0a” or “eastman:/”. |
| dir | the directory upon which this filesystem is mounted. |
| type | the filesystem type of the mount. |
| opts | a list of values, each of which is a **mount(8)** option associated with the mounting of this filesystem. |
| passno | pass number on parallel **fsck(8)**. |
| freq | dump frequency, in days. |

The directory **/networks** contains directories which refer to Internet networks. The relevant properties are:

| | |
|---|---|
| name | the name of the network. If the network has aliases, there may be more than one value for this property. |
| address | the network number of this network. The value MUST be in decimal-dot notation with no leading zeroes. |

The directory **/printers** contains directories which refer to printer entries. The relevant properties are:

| | |
|---|---|
| name | the name of the printer. If the printer has alias, this property will have multiple values. |
| lp, sd, etc. | the names of **printcap(5)** properties associated with this printer. If the value associated with the property name is numeric, the number has a leading "#" prepended to it. |

The directory **/protocols** contains directories which refer to transport protocols. The relevant properties are:

| | |
|---|---|
| name | the name of the protocol. If the protocol has aliases, the property will have multiple values. |
| number | the associated protocol number. |

The directory **/services** contains directories which refer to ARPA services. The relevant properties are:

| | |
|---|---|
| name | the name of the service. If the service has aliases, the property will have multiple values. |
| protocol | the name of the protocol upon which the service runs. If the service runs on multiple protocols, this property will have multiple values. |
| port | the associated port number of the service. |

The directory **/users** contains information which refer to users. The relevant properties are:

| | |
|---|---|
| name | the login name of the user. |
| passwd | the encrypted password of the user. |
| uid | the user id of the user. |
| gid | the default group id of the user. |
| realname | the real name of the user. |
| home | the home directory of the user. |
| shell | the login shell of the user. |

**AUTHOR**

NeXT Computer Inc.

**NON-NEXTSTEP SUPPORT**

Xedoc Software Development Pty. Ltd.

**SEE ALSO**

**aliases(5), bootparams(5), bootptab(5), fstab(5), group(5), hosts(5), niypd(8), netinfod(8), netgroup(5), networks(5), passwd(5), printcap(5), protocols(5), services(5)**

**netinfod(8)**

## NAME

netinfod - NetInfo daemon

## SYNOPSIS

**netinfod** *domain-tag*

## DESCRIPTION

**netinfod** daemon serves the information in the given *domain-tag* to the network. It is normally executed automatically by **nibindd(8)** and should not be run manually.

## FILES

**/etc/netinfo/*domain_tag*.nidb**

where the actual information served is stored.

## AUTHOR

NeXT Computer Inc.

## NON-NEXTSTEP SUPPORT

Xedoc Software Development Pty. Ltd.

## SEE ALSO

**netinfo(5)**

**nibindd(8)**

## NAME

nibindd - NetInfo binder

## SYNOPSIS

**nibindd**

## DESCRIPTION

The **nibindd** daemon is responsible for finding, creating and destroying NetInfo (**netinfod(8)**) servers. When it starts up, it reads the directory /etc/netinfo for directories with the extension ".nidb" and starts up a **netinfod(8)** server for each NetInfo database it finds. If **nibindd** is sent a hangup signal, SIGHUP, it kills all running **netinfod** processes and rebinds the NetInfo domain hierarchy (note that this does not affect the connections established by **niypd(8)**). This is useful for getting the system to conform to new network configuration changes without rebooting. nibindd writes a file with its process ID number (pid file) in **/etc/nibindd.pid**.

The **nibindd** daemon will automatically destroy the registration for a **netinfod** server if it disappears for any reason. It will take the **netinfod** servers down if it is shut down by sending it a terminate signal, SIGTERM.

**nidomain**(8) is the user interface to **nibindd**.

## AUTHOR

NeXT Computer Inc.

## NON-NEXTSTEP SUPPORT

Xedoc Software Development Pty. Ltd.

## SEE ALSO

**netinfod(8)**, **nidomain(8)**

**nibootparamd(8)**

## NAME

nibootparamd - boot parameter server

## SYNOPSIS

**nibootparamd** [ **-d** ]

## DESCRIPTION

**nibootparamd** is a server process that provides information to NetBoot clients necessary for booting. It consults the NetInfo database (the **/machines** directory) if NetInfo is running, and examines the boot and address properties. If the client's information is not found, NIS is consulted if it is running.

**nibootparamd** can be invoked either by **inetd**(8) or by the user.

## OPTIONS

**-d** Display the debugging information

## FILES

/etc/bootparams - if NetInfo is not running.

## AUTHOR

Xedoc Software Development Pty. Ltd.

## SEE ALSO

**bootparams(5), inetd(8), nidump(8), niload(8)**, **netinfo(5)**

**nibootpd(8)**

## NAME

nibootpd - boot protocol daemon

## SYNOPSIS

**nibootpd** [ **-d** ]

## DESCRIPTION

**nibootpd** is the bootstrap protocol server daemon described in RFC 951. It is used by diskless hosts to resolve their protocol addresses and to determine which bootfile to netload. **nibootpd** is normally run as a subprocess of **inetd**(8) daemon.

The file `/etc/bootptab` is the standard database for **nibootpd**. When NetInfo is running, this file is **not** consulted, and all nibootpd information comes from NetInfo. However, even when NetInfo is running, this file **must** exist.

Blank lines and lines beginning with '#' are ignored. The first section of the file contains default parameters, one per line. The first parameter is the default directory of the bootfiles. The second parameter is the name of the default bootfile. A line beginning with '%%' marks the end of the parameter section.

The remainder of the file contains one line per client interface, with the information show below. The 'host' name is also tried as a suffix for the 'bootfile' when searching the home directory, e.g. 'bootfile.host'. For 10MB Ethernet 'htype' is always '1'.

```
# host    htype     haddr              iaddr       bootfile
tc101g    1         02:60:8c:06:35:05  36.44.0.65  seagate
```

## OPTIONS

**-d** The **-d** flag enables debugging output.

## FILES

/etc/bootptab

**en_address**, **bootfile** and **bootparams** properties of the **/machines/*hostname*** directory in NetInfo.

## AUTHOR

Xedoc Software Development Pty. Ltd.

## SEE ALSO

**niload(8)**, **nidump(8), niutil(8)**, **netinfo(5)**

Bootstrap Protocol (BOOTP), RFC 951, Croft and Gilmore.

**nidomain(8)**

## NAME

nidomain - NetInfo domain utility

## SYNOPSIS

**nidomain -l** *[ hostname ]*
**nidomain -m** *tag*
**nidomain -d** *tag*
**nidomain -c** *tag master/remotetag*

## DESCRIPTION

The **nidomain** utility is an interface to **nibindd(8)**, to which it sends all of its requests about the domains served on a given machine. It also can be used to create and destroy NetInfo databases.

The **nidomain** utility will allow one to create multiple levels of NetInfo hierarchy, but it is not a particularly easy way to do it. One should use the NetInfo Manager application for setting up multilevel hierarchies.

## OPTIONS

**-l** [ *hostname* ]

List the domains by tag served on the given hostname. If hostname is unspecified, the local host is used.

**-m** *tag*

Create a new NetInfo database and server on the local machine for the domain tag of *tag*.

**-d** *tag*

Destroy the local NetInfo database and server associated with the domain tagged *tag*. If the database was associated with a clone, the machine's “serves” property on the master is NOT modified to reflect the fact that the database has been deleted.

**-c** *tag master/remotetag*

Creates a clone NetInfo database with the domain tagged *tag*. The database is cloned from the machine *master* and remote tag *remotetag*. The “serves” property on the master machine should be set up prior to running this command to contain the entry “./tag”.

## AUTHOR

NeXT Computer Inc.

## NON-NEXTSTEP SUPPORT

Xedoc Software Development Pty. Ltd.

## SEE ALSO

**nibindd(8)**

**nidump(8)**

## NAME

nidump - extract a UNIX-format file out of NetInfo

## SYNOPSIS

**nidump** [ **-t** ] *{-r directory | format } domain*

## DESCRIPTION

**nidump** reads the specified NetInfo domain and dumps a portion of its contents to standard output. When a UNIX administration file *format* is specified, **nidump** provides output using the syntax of the corresponding UNIX flat file. The allowed values for format are *aliases*, *bootparams*, *bootptab*, *exports*, *fstab*, *group*, *hosts*, *networks*, *passwd*, *printcap*, *protocols*, *rpc*, and *services*.

## OPTIONS

**-t** Interpret the domain as a tagged domain. For example, “trotter/network” refers to the database tagged “network” on the machine “trotter”. You may supply an IP address instead of a machine name.

**-r** Dump the specified directory in "raw" format. Directories are delimited by curly braces, and properties within a directory are listed in the form "property = value;". Parentheses introduce a comma-separated list of items. The special property name CHILDREN is used to hold a directory's children, if any are present. Spacing and line breaks are significant only within double quotes, which may be used to protect any names that might contain metacharacters.

## EXAMPLE

`nidump passwd .` - dumps a password file from the local NetInfo domain.

`nidump -r /locations /` dumps the /locations directory of the root domain.

`nidump -t -r /name=users/uid=530 trotter/network` dumps the directory for the user whose UID is 530.

## RESTRICTIONS

The -r option is not supported by Xedoc NetInfo Editions.

## AUTHOR

NeXT Computer Inc.

## NON-NEXTSTEP SUPPORT

Xedoc Software Development Pty. Ltd.

## SEE ALSO

**niload(8), niutil(8), netinfo(5)**

**nifind(1)**

## NAME

nifind - find a directory in the NetInfo hierarchy

## SYNOPSIS

**nifind [ -anvp ] [ -t** *timeout* **]** *directory [ domain ]*

## DESCRIPTION

**nifind** searches for the named directory in the NetInfo hierarchy. It starts at the local domain and climbs up through the hierarchy until it reaches the root domain. Any occurances of *directory* are reported by directory ID number. If the optional *domain* argument is given, then **nifind** stops climbing at that point in the hierarchy. The *domain* argument must be specified by an absolute or relative domain name.

When invoked with the **-a** option, **nifind** searches for the named directory in the entire NetInfo hierarchy (or the sub- tree with *domain* as the root if *domain* is specified). The **-n** option exempts local domains from the search.

**nifind** uses a default two second connection timeout when contacting a domain. The connection timeout can be overriden with the **-t** option.

## OPTIONS

**-a** Search for *directory* in the entire NetInfo hierarchy.

**-n** Exempt local directories from the search.

**-p** Print directory contents.

**-v** Produce verbose output.

**-t** *timeout*

Specify an integer value as the connection timeout (in seconds).

## EXAMPLES

```
% nifind /printers/inky
/printers/inky found in /software, id = 202
/printers/inky found in /, id = 357

% nifind  -a /machines/mailhost /sales
/machines/mailhost found in /sales, id = 234

% nifind -v /users/uid=164
/users/uid=164 not found in /sales/polaris
/users/uid=164 not found in /sales
/users/uid=164 found in /, id = 451
```

```
% nifind -p /machines/mailhost
/machines/mailhost found in /sales, id=171
name: zippy mailhost
ip_address: 192.42.172.5

/machines/mailhost found in /, id = 350
name:  zorba mailhost
ip_address: 192.42.172.1
```

**SEE ALSO**

**netinfo(5)**

**AUTHOR**

Marc Majka, NeXT Computer Inc.

**NON-NEXTSTEP SUPPORT**

Xedoc Software Development Pty. Ltd.

**BUGS**

**nifind** does not complain if *domain* is not an ancestor specified in a normal search.

**nigrep(1)**

## NAME

nigrep - search for a regular expression in the NetInfo hierarchy

## SYNOPSIS

**nigrep** *expression* **[ -t ]** *domain [ directory ... ]*

## DESCRIPTION

**nigrep** searches through the specified domain argument for a regular expression. It searches the domain's directory hierarchy depth-first starting from the root directory. It can also start from each directory specified on the command line.

The *domain* argument can be specified as an absolute or relative domain name. The *domain* argument can be specified as a network address or hostname and tag if preceded by the **-t** option.

On output, **nigrep** prints the directory ID number of the directory which contains the regular expression, and the property key and values where it was found. A line is printed for each property that contains the regular expression.

## OPTIONS

**-t** Specify domain as a network address or hostname and tag.

## EXAMPLES

```
% nigrep '[Nn]et' /
% nigrep '[Nn]et' -t 192.42.172.17/local
% nigrep '192.[0-9]+.172' -t astra/network /machines
% nigrep '/Net/server.*/Users' .. /users /mounts
```

## SEE ALSO

**netinfo(5)**

## AUTHOR

Marc Majka, NeXT Computer Inc.

## NON-NEXTSTEP SUPPORT

Xedoc Software Development Pty. Ltd.

**niload(8)**

## NAME

niload - load UNIX-format file into NetInfo

## SYNOPSIS

**niload [ -v ] [ -d ] [ -p ] [ -t ]** *format domain*

## DESCRIPTION

**niload** loads information from standard input into the given NetInfo domain.

If format is specified, the input is interpreted according to the UNIX file format of the same name. The allowed values for format are *aliases*, *bootparams*, *bootptab*, *exports*, *fstab*, *group*, *hosts*, *networks*, *passwd*, *printcap*, *protocols*, *rpc*, and *services*.

If **-r** *directory* is specified instead of a UNIX file format, the input is interpreted as "raw" NetInfo data, as generated by **nidump -r**, and loaded into *directory*.

**niload** overwrites entries in the existing directory with those given in the input. Entries that are in the directory aren't deleted if they don't exist in the input, unless the **-d** option is specified. **niload** must be run as superuser on the master NetInfo server for the given domain, unless one specifies the **-p** option, which allows one to run from anywhere in the network.

## OPTIONS

**-v** Verbose. Prints a "+" for each entry loaded, a "-" for each entry deleted. (UNIX formats only)

**-d** Delete entries which are in the existing directory, but not in the input.

**-p** Prompt for the root password of the given domain so that one can run from other locations in the network besides the master.

**-t** Interpret the domain as a tagged domain. For example, "trotter/network" refers to the database tagged "network" on the machine "trotter". You may supply an IP address instead of a machine name.

**-r** Load entries in "raw" format, as generated by nidump -r. The first argument should be the path of a NetInfo directory into which the information is loaded. Since the input often specifies properties (including "name") at its topmost level, the directory you specify may be renamed as a result of this operation. If the directory you specify does not exist, it will be created.

## EXAMPLE

`niload passwd . < /etc/passwd` load the local /etc/passwd file into the local NetInfo database.

`niload -d -r /locations .` replaces the contents of /locations in the local domain with input given in nidump "raw" format.

## RESTRICTIONS

The -r option is not supported by Xedoc NetInfo Editions.

**AUTHOR**

NeXT Computer Inc.

**NON-NEXTSTEP SUPPORT**

Xedoc Software Development Pty. Ltd.

**SEE ALSO**

**nidump(8), niutil(8), netinfo(5), aliases(5), bootparams(5), bootptab(5), exports(5), fstab(5), group(5), hosts(5), networks(5), passwd(5), printcap(5), protocols(5), rpc(5), services(5)**

**nipasswd(1)**

## NAME

nipasswd - change NetInfo password information

## SYNOPSIS

**nipasswd** [ username ]

## DESCRIPTION

**nipasswd** changes a user's password in the NetInfo database.

The superuser may change anyone's password without being required to enter the old password. Ordinary users may only change their own password.

When changing a password, **nipasswd** prompts for the old password and then for the new one. If the old password is not entered correctly, it will not be changed. The new password must be typed twice to forestall mistakes.

**nipasswd** will search up the NetInfo domain hierarchy starting from the local domain until it finds a password entry for a user. The password is then changed at that point.

In other words, **nipasswd** changes the password for a user at the lowest possible level of the NetInfo domain hierarchy. The command will not search further up the hierarchy to change further password entries.

New passwords should be at least six characters long. If you persist in entering a shorter password it will eventually be accepted. Users should be warned that this may result in compromising system security.

## NOTES

Password algorithms do not work with 8-bit characters. This maintains consistency with login file naming rules, which do not allow 8-bit characters in login names. See login(1) for explanations about why login is not 8-bit clean.

## AUTHOR

Xedoc Software Development Pty. Ltd.

## SEE ALSO

**finger(1), login(1), yppasswd(1), crypt(1), passwd(1), netinfo(5)**

**nireport(1)**

## NAME

nireport - print tables from the NetInfo hierarchy

## SYNOPSIS

**nireport [ -t ]** *domain directory [ property ...]*

## DESCRIPTION

**nireport** prints a table of values of properties in all sub-directories of the directory given on the command line (see "Examples"). Multiple values of a property are printed in a comma-separated list.

The *domain* argument can be specified as an absolute or relative domain name. The *domain* argument can also be specified as a network address or host name and tag if it is preceded by the **-t** option.

## OPTIONS

**-t** Specify *domain* as a network address or hostname and tag.

## EXAMPLES

```
% nireport /software/duck /users name uid passwd
root      0    0NNGzihc9ILeg
nobody    -2   *
agent     1    *
daemon    1    *
uucp      4    *
news      6    *
sybase    8    *
me        20   DJJ.KMEC.OgIY

% nireport -t crow/network /machines name ip_address serves
crow      129.18.10.221  ./network,crow/local
robin     129.18.10.216  robin/local
raven     129.18.21.6    ./network,raven/local
duck      129.18.10.210  duck/local
```

## AUTHOR

Marc Majka, NeXT Computer Inc.

## NON-NEXTSTEP SUPPORT

Xedoc Software Development Pty. Ltd.

**niutil(8)**

## NAME

niutil - NetInfo utility

## SYNOPSIS

**niutil [ opts ] -create** *domain path*

**niutil [ opts ] -destroy** *domain path*

**niutil [ opts ] -createprop** *domain path propkey [ val ... ]*

**niutil [ opts ] -appendprop** *domain path propkey val ...*

**niutil [ opts ] -mergeprop** *domain path propkey val ...*

**niutil [ opts ] -insertval** *domain path propkey val index*

**niutil [ opts ] -destroyprop** *domain path propkey ...*

**niutil [ opts ] -destroyval** *domain path propkey val ...*

**niutil [ opts ] -renameprop** *domain path oldkey newkey*

**niutil [ opts ] -read** *domain path*

**niutil [ opts ] -list** *domain path [ propkey ]*

**niutil [ opts ] -rparent** *domain*

**niutil [ opts ] -resync** *domain*

**niutil [ opts ] -statistics** *domain*

**opts: [ -t ] [ -p ] [ -u** *user***] [ -P** *password* **] [ -T** *timeout***]**

## DESCRIPTION

**niutil** lets you to do arbitrary reads and writes on the given NetInfo domain. In order to perform writes, **niutil** must be run as root on the NetInfo master for the database, unless the **-p** option is given.

The database directory specified by *path* is separated by "/"s, similar to UNIX. The property names may be given in the path using a "=", but will default to the property name "name".

For example, the following refers to a user with the user ID 3.

```
/name=users/uid=3
```

The following shorter form would also be sufficient:

```
/users/uid=3
```

You may specify a numeric ID for the directory instead of the string path.

## OPTIONS

**-t** Interpret the domain as a tagged domain.

For example, "eastman/network" refers to the database tagged "network" on the machine "eastman". You may supply an IP address instead of a machine name.

**-p** Prompt for the root password of the given domain so that one can run from other locations in the network besides the master.

**-u** Authenticate as another user (implies -p).

**-P** Password supplied on command line (overrides -p).

**-T** Read & write timeout in seconds (default 30).

**-create** *domain path*

Create a new directory with the given path.

**-destroy** *domain path*

Destroy the directory with the given path.

**-createprop** *domain path propkey [ val ... ]*

Create a new property in the directory path. *propkey* refers to the name of the property; 0 or more property values may be specified. If the named property already exists, it is overwritten. Use **-appendprop** to add new values without overwriting existing ones.

**-appendprop** *domain path propkey val ...*

Append the value *val* to the property *propkey* in the given path. The property will be created if it doesn't already exist. If *val* already exists, a duplicate entry will be created.

**-mergeprop** *domain path propkey val ...*

Merge values into the property *propkey* in the given path. The property will be created if it doesn't already exist. If *val* already exists, it isn't duplicated.

**-insertval** *domain path propkey val index*

Insert value *val* at the given index position of the property *propkey*.

**-destroyprop** *domain path propkey*

Destroy the property with name *propkey* in the given path.

**-destroyval** *domain path propkey val ...*

Remove a property value from the property *propkey* in directory *path*.

**-renameprop** *domain path oldkey newkey*

Rename a property key in the given path.

**-read** *domain path*

Read the properties associated with the directory specified in the given path.

**-list** *domain path*

List the directories in the given domain/path. The directory ID's are listed along with any names they may have.

**-rparent** *domain*

Get a server's current NetInfo parent.

**-resync** *domain*

Force master and clone servers to resynchronize databases.

**-statistics** *domain*

Print server version number and database checksum.

**EXAMPLE**

```
niutil -list . /
```

list the directories at the top level in the local NetInfo database.

**AUTHOR**

NeXT Computer Inc.

**NON-NEXTSTEP SUPPORT**

Xedoc Software Development Pty. Ltd.

**SEE ALSO**

**niload(8), nidump(8), netinfo(5)**

**niwhich(1)**

## NAME

niwhich - return host information for NetInfo domains

## SYNOPSIS

**niwhich -d** *domain* **[ -p | -h | -i ]**

## DESCRIPTION

**niwhich** is used to determine which host serves a given NetInfo domain. Given a domain name, **niwhich** will output the hostname, IP address and database tag for the domain. The **-h** and **-i** options are for convenience, and are mostly useful in writing shell scripts to create new domains.

**niwhich** returns a status code to indicate success/failure to contact the given domain.

## OPTIONS

**-d** Specify domain.

The special name "." refers to the local domain, while the name ".." refers to the parent of the local domain. The name "/" refers to the root of the NetInfo domain hierarchy. The full domain pathname uses "/" as a separator.

**-h** Only output the name of host that serves database for specified domain.

**-i** Only output the IP address of the host that serves the specified domain.

**-p** Do not output any information at all. Useful to 'probe' a binding to make sure it has come up correc tly.

## EXAMPLE

```
niwhich -d ..
```

outputs host information for the host that serves the parent of the local domain.

## AUTHOR

Xedoc Software Development Pty. Ltd.

## SEE ALSO

**niutil(8), netinfo(5)**

**niypd(8)**

## NAME

niypd - NIS emulation server

## SYNOPSIS

**niypd [ -V ] [ -d ] [ -D ] [ -l ]** *[ domain_name ]*

## DESCRIPTION

**niypd** is a server process that provides information to any process that makes calls to the NIS client side routines. Such processes include any process that uses the standard libc calls such as `getpwent()`, `gethostent()` etc. and also, the special tools **ypcat** and **ypmatch** provided as part of the standard NIS distribution.

**niypd** emulates the equivalent process **ypserv** by providing an RPC call-compatible interface. Rather than consulting 'map' files as **ypserv** does, however, **niypd** draws its data from NetInfo databases.

Communication to and from **niypd** is by means of RPC calls. Lookup functions are described in ypclnt(3N), and are supplied as C-callable functions in `/lib/libc`.

**niypd** is capable of using data from NIS databses as well as NetInfo, through support for the concept of a 'parent' NIS domain. All NIS derived data will appear AFTER NetInfo data in **ypcat**, **ypmatch** or other queries.

If the *domain_name* parameter is not provided on the command line, the parent domain is derived by removing the first component of the local domain name.

e.g. if the local domain is `alpha.xyz.com` then the parent domain will be `xyz.com`

If the *domain_name* parameter is supplied on the command line, it will be used instead.

Finally, if the *domain_name* parameter is the special token `local` then **niypd** will NOT make calls to the underlying NIS system for parent lookups. This allows a systems administrator to clearly separate those systems administered by NetInfo from those using NIS.

You *cannot* run niypd and ypserv on the same host.

**OPTIONS**

-V Show version of the software.

-d Turn on verbose debugging output.

-D Do not detach and daemonize process. Logging is to standard output.

-l Override system default location of ypserv.log file.

*domain_name*

Either `local` to override NIS fall-through or an actual NIS domain name to use as a parent domain. Omitting this parameter causes niypd to derive the parent domain using the approach above.

**NON-NEXTSTEP SUPPORT**

Xedoc Software Development Pty. Ltd.

**SEE ALSO**

**ypserv(8), ypbind(8), ypinit(8), netinfo(5)**

# Index