NFS Reference Manual Pages

IRIS-4D Series

SiliconGraphics
Computer Systems

Document number: 007-0627-030

NFS Reference Manual Pages

Document Version 3.0

Document Number 007-0627-030

8/90

NFS Reference Manual Pages
Document Version 3.0
Document Number 007-0627-030

Silicon Graphics, Inc.
Mountain View, California

Preface

Here are your NFS Reference Manual Pages. You may place them behind you *NFS User's Guide* or put them in the binder labelled *IRIS-4D Optional Manual Pages.* You received this binder with your IRIS-4D Series Reference Manuals.

NAME

domainname – set or display name of current YP domain

SYNOPSIS

domainname [*nameofdomain*]

DESCRIPTION

Without an argument, *domainname* displays the name of the current Yellow Pages domain. Only the super-user can set the domain name by giving an argument; this is usually done in the startup script */etc/init.d/network*. Currently, the Yellow Pages uses domains only to refer collectively to a group of hosts.

SEE ALSO

ypinit(1M)

NAME

on – execute a command remotely

SYNOPSIS

on [**–i**] [**–n**] [**–d**] *host command* [*argument*] ...

DESCRIPTION

The *on* program is used to execute commands on another system, in an environment similar to that invoking the program. All environment variables are passed, and the current working directory is preserved. To preserve the working directory, the working file system must be either already mounted on the host or be exported to it. Relative path names will only work if they are within the current file system; absolute path names may cause problems.

Standard input is connected to standard input of the remote command, and standard output and standard error from the remote command are sent to the corresponding files for the *on* command.

OPTIONS

–i Interactive mode: use remote echoing and special character processing. This option is needed for programs that expect to be talking to a terminal. All terminal modes and window size changes are propagated.

–n No Input: this option causes the remote program to get end-of-file when it reads from standard input, instead of passing standard input from the standard input of the *on* program. For example, **–n** is necessary when running commands in the background with job control.

–d Debug mode: print out some messages as work is being done.

SEE ALSO

rexd(1M), exports(4)

DIAGNOSTICS

unknown host
: Host name not found.

cannot connect to server
: Host down or not running the *rexd* server.

can't find .
: Problem finding the working directory.

can't locate mount point
: Problem finding current file system.

Other error messages may be passed back from the server.

BUGS

The window size is not set properly when executing interactively on Sun workstations.

NAME

rup – show host status of local machines (RPC version)

SYNOPSIS

rup [**–h**] [**–l**] [**–t**] [host ...]

DESCRIPTION

Rup gives a status similar to *uptime* for remote machines; it broadcasts on the local network, and displays the responses it receives.

Normally, the listing is in the order that responses are received, but this order can be changed by specifying one of the options listed below.

When *host* arguments are given, rather than broadcasting *rup* will only query the list of specified hosts.

A remote host will only respond if it is running the *rstatd* daemon, which is normally started up from *inetd*(1M).

OPTIONS

–h sort the display alphabetically by host name.

–l sort the display by load average.

–t sort the display by up time.

SEE ALSO

ruptime(1C), inetd(1M), rstatd(1M)

BUGS

Broadcasting does not work through gateways.

NAME

rusers – who's logged in on local machines (RPC version)

SYNOPSIS

rusers [**–a**] [**–h**] [**–i**] [**–l**] [**–u**] [host ...]

DESCRIPTION

The *rusers* command produces a listing of users on remote machines. It broadcasts on the local network, and prints the responses it receives. Normally, the listing is in the order that responses are received, but this order can be changed by specifying one of the options listed below. When *host* arguments are given, rather than broadcasting *rusers* will only query the list of specified hosts.

The default is to print out a listing with one line per machine. When the **–l** flag is given, a *who*(1) style listing is used. In addition, if a user hasn't typed to the system for a minute or more, the idle time is reported.

A remote host will only respond if it is running the *rusersd* daemon, which is normally started up from *inetd*.

OPTIONS

–a gives a report for a machine even if no users are logged on.

–h sort alphabetically by host name.

–i sort by idle time.

–l Give a longer listing in the style of *who*.

–u sort by number of users.

SEE ALSO

rwho(1C), inetd(1M), rusersd(1M)

BUGS

Broadcasting does not work through gateways.

NAME

rwall – write to all users over a network

SYNOPSIS

rwall host1 host2 ...
rwall **–n** netgroup1 netgroup2 ...
rwall **–h** host **–n** netgroup

DESCRIPTION

Rwall reads a message from standard input until end-of-file. It then sends this message, preceded by the line "Broadcast Message ...", to all users logged in on the specified host machines. With the **-n** option, it sends to the specified network groups, which are defined in *netgroup*(4).

A machine can only receive such a message if it is running *rwalld*(1m), which is normally started up by the daemon *inetd*(1m).

SEE ALSO

wall(1), rwalld(1M), netgroup(4)

BUGS

The timeout is fairly short in order to be able to send to a large group of machines (some of which may be down) in a reasonable amount of time. Thus the message may not get through to a heavily loaded machine.

NAME

ypcat – print values in a YP data base

SYNOPSIS

ypcat [**–k**] [**–t**] [**–d** *domainname*] *mname*
ypcat –x

DESCRIPTION

ypcat prints out values in a Yellow Pages (YP) map specified by *mname*, which may be either a *mapname* or a map *nickname*. Since *ypcat* uses the YP network services, no YP server is specified.

To look at the network-wide password database, *passwd.byname*, (with the nickname *passwd*), type in:

```
ypcat passwd
```

Refer to *ypfiles*(4) and *ypserv*(1M) for an overview of the Yellow Pages.

OPTIONS

–k Display the keys for those maps in which the values are null or the key is not part of the value. (None of the maps derived from files that have an ASCII version in */etc* fall into this class.)

–t Inhibit translation of *mname* to *mapname*. For example, *ypcat –t passwd* will fail because there is no map named *passwd*, whereas *ypcat passwd* will be translated to *ypcat passwd.byname*.

–d Specify a domain other that the default domain. The default domain is returned by *domainname*.

–x Display the map nickname table. This lists the nicknames (*mnames*) the command knows of, and indicates the *mapname* associated with each nickname.

SEE ALSO

ypfiles(4), ypserv(1M), ypmatch(1), domainname(1)

NAME

ypchpass – change selected Yellow Pages passwd fields

SYNOPSIS

ypchpass [**–f** *fullname*] [**–h** *home*] [**–s** *shell*] [*name*]

DESCRIPTION

Ypchpass changes selected *passwd*(4) fields associated with the user *name* (your own name by default) in the Yellow Pages. The Yellow Pages fields that can be modified with *ypchpass* may have different contents from those for the same user in the local */etc/passwd* file.

If invoked without options, *ypchpass* invokes the editor named by the environment variable **EDITOR**, or *vi*(1) if **EDITOR** is null or unset, to edit a template of selected *passwd* fields. After the user has edited this template and changed or added appropriate field contents, *ypchpass* checks the updated contents. If they are well-formed, it updates the Yellow Pages. Otherwise it prompts the user to re-edit the template.

If invoked with options, *ypchpass* updates the Yellow Pages based on the options' arguments and does not interactively acquire new field contents using an editor.

–f *fullname* Change the user's real name field to contain *fullname*. A useful convention, observed by *finger*(1), divides this field into four comma-separated parts: the user's real name, office, extension, and home phone.

–h *home* Change the initial working directory field to *home*.

–s *shell* Change the shell field to *shell*.

Only the owner of *name* may change its *passwd* fields.

SEE ALSO

yppasswd(1), ypfiles(4), rpc.passwd(1M)

NAME

ypmatch – print the value of one or more keys from a YP map

SYNOPSIS

ypmatch [**–d** *domain*] [**–k**] [**–t**] *key* ... *mname*
ypmatch –x

DESCRIPTION

ypmatch prints the values associated with one or more keys from the Yellow Pages (YP) map (database) specified by a *mname*, which may be either a *mapname* or an map *nickname*.

Multiple keys can be specified; the same map will be searched for all. The keys must be exact values insofar as capitalization and length are concerned. No pattern matching is available. If a key is not matched, a diagnostic message is produced.

OPTIONS

–d Specify a domain other that the default domain.

–k Before printing the value corresponding to a key, print the key itself, followed by a colon (':'). This is useful only if the keys are not duplicated in the values, or you've specified so many keys that the output could be confusing.

–t Inhibit translation of nickname to mapname. For example, *ypmatch –t zippy passwd* will fail because there is no map named *passwd*, while *ypmatch zippy passwd* will be translated to *ypmatch zippy passwd.byname*.

–x Display the map nickname table. This lists the nicknames (*mnames*) the command knows of, and indicates the *mapname* associated with each nickname.

SEE ALSO

ypcat(1), ypfiles(4)

NAME

yppasswd – change login password in Yellow Pages

SYNOPSIS

yppasswd [name]

DESCRIPTION

Yppasswd changes (or installs) a password associated with the user *name* (your own name by default) in the Yellow Pages. The Yellow Pages password may be different from the one on your own machine.

Yppasswd prompts for the old Yellow Pages password and then for the new one. The caller must supply both. The new password must be typed twice, to forestall mistakes.

New passwords must be at least four characters long if they use a sufficiently rich alphabet and at least six characters long if monocase. These rules are relaxed if you are insistent enough.

Only the owner of the name or the super-user may change a password; in either case you must prove you know the old password.

SEE ALSO

passwd(1), ypfiles(4), rpc.passwd(1M)

BUGS

The update protocol passes all the information to the server in one rpc call, without ever looking at it. Thus if you type in your old password incorrectly, you will not be notified until after you have entered your new password.

NAME

ypwhich – print the YP server or map master hostname

SYNOPSIS

ypwhich [**–d** *domain*] [**–V1** | **–V2**] [*hostname*]
ypwhich [**–t**] [**–d** *domain*] **–m** *mname*
ypwhich –x

DESCRIPTION

ypwhich tells which YP server supplies Yellow Pages services to a YP client, or which server is the master for a map. If invoked without arguments, it prints the YP server for the local machine. If *hostname* is specified, that machine is queried to find out which YP server it is using.

Refer to *ypfiles*(4) and *ypserv*(1M) for an overview of the Yellow Pages.

OPTIONS

–d *domain* Use *domain* instead of the default domain.

–V1 Which server is serving v.1 YP protocol-speaking client processes?

–V2 Which server is serving v.2 YP protocol client processes?

If neither version is specified, *ypwhich* attempts to locate the server that supplies the (current) v.2 services. If there is no v.2 server currently bound, *ypwhich* then attempts to locate the server supplying the v.1 services. Since YP servers and YP clients are both backward compatible, the user need seldom be concerned about which version is currently in use.

–m *mname* Find the master YP server for a map. No *hostname* can be specified with **–m**. *mname* can be a mapname, or a nickname for a map.

–t Inhibit nickname translation. This option is useful if there is a mapname identical to a nickname, which is not true of any SGI-supplied map.

–x Display the map nickname table. This lists the nicknames (*mnames*) the command knows of, and indicates the *mapname* associated with each nickname.

SEE ALSO

rpcinfo(1M), ypserv(1M), ypset(1M), ypfiles(4)

NAME

automount – automatically mount NFS filesystems

SYNOPSIS

/usr/etc/automount [**–mnTv**] [**–D** *name=value*] [**–f** *master-file*] [**–M** *mount-directory*] [**–tl** *duration*] [**–tm** *interval*] [**–tw** *interval*] [*directory mapname* [*–mount-options*]] ...

DESCRIPTION

automount is a daemon that automatically and transparently mounts an NFS filesystem as needed. It monitors attempts to access directories that are associated with an automount "map." You can assign a map to a directory using an entry in a direct automount map, or by specifying an indirect map on the command line.

The *automount* daemon is started during system initialization from the */etc/init.d/network* script if the configuration flags "nfs" and "automount" are set (see *chkconfig*(1M) and *network*(1M)). Site-dependent options and arguments to *automount* belong in the file */etc/config/automount.options*.

automount appears to be an NFS server to the kernel. *automount* uses the map contained in the *mapname* argument to determine a server, exported filesystem, and appropriate mount options for a given filesystem. It then mounts the filesystem in a temporary location, and creates a symbolic link to the temporary location. If the filesystem is not accessed within an appropriate interval (five minutes by default), the daemon unmounts the filesystem and removes the symbolic link.

By default, *automount* mounts everything under the directory */tmp_mnt*. For instance, if a user wants to mount a remote directory *src* under */usr/src*, the actual mount point will be */tmp_mnt/usr/src*, and */usr/src* will be a symbolic link to that location. As with any other kind of mount, a mount affected through the automounter on a non-empty mount point will hide the original contents of the mount point for as long as the mount is in effect.

The name-to-location binding is dynamic, making updates to a Yellow Pages (YP) map transparent to the user. This obviates the need to "premount" shared file systems for applications that have "hard-coded" references to files. Nor is there a need to maintain records of which hosts must be mounted for what applications.

If the *directory* argument does not exist, *automount* creates it, and it is removed automatically when *automount* exits.

If the *directory* argument is a pathname, the *map* argument must be an *indirect* map. In an indirect map, the key for each entry is a simple name (as opposed to a full pathname) that represents a symbolic link within *directory* to an NFS mount point.

If the *directory* argument is '/–', the map that follows must be a *direct* map. A direct map is not associated with a single directory. Instead, the key for each entry is a full pathname that will itself appear to be a symbolic link to an NFS mount point.

A map can be a file or a YP map; if a file, the *map* argument must be a full pathname.

The *–mount-options* argument, when supplied, consists of the leading dash and a comma-separated list of *mount*(1M) options. If these options are supplied, they become the default mount options for all entries in the map. Mount options provided within a map entry override these defaults.

OPTIONS

–D *var=value*
Assign *value* to the indicated automount (environment) variable.

–f *master-file*
Read a local file for initialization, before reading the *auto.master* YP map. The information in *master-file* will take precedence.

–m Suppress initialization of *directory–mapname* pairs listed in the *auto.master* YP database.

–M *mount-directory*
Mount temporary filesystems in the named directory, instead of */tmp_mnt*.

–n Disable dynamic mounts. With this option, references through the *automount* daemon only succeed when the target filesystem has been previously mounted.

–T Trace. Expand each NFS call and display it on the standard output.

–tl *duration*
Specify a *duration*, in seconds, that a looked up name remains cached when not in use. The default is 5 minutes.

–tm *interval*
Specify an *interval*, in seconds, between attempts to mount a filesystem. The default is 30 seconds.

–tw *interval*
Specify an *interval*, in seconds, between attempts to dismount filesystems that have exceeded their cached times. The default is 1 minute.

–v Verbose. Log status and/or warning messages to the console.

ENVIRONMENT

Environment variables can be used within an automount map. For instance, if $HOME appeared within a map, *automount* would expand it to its current value for the HOME variable. Environment variables are expanded only for the automounter's environment — not for the environment of a user using the automounter's services.

USAGE

Map Entry Format

A simple map entry (mapping) takes the form:

key [–mount-options] location ...

where *key* is the full pathname of the directory to mount when used in a direct map, or simple name in an indirect map. *mount-options* is a comma-separated list of mount options, and *location* specifies a remote filesystem from which the directory may be mounted. In the simple case, *location* takes the form:

hostname:pathname

Replicated Filesystems

Multiple *location* fields can be specified for replicated read-only filesystems, in which case *automount* sends multiple mount requests; *automount* mounts the filesystem from the first host that replies to the mount request. This request is first made to the local net or subnet. If there is no response, any connected server may respond. Since *automount* does not monitor the status of the server while the filesystem is mounted, it will not use another location in the list if the currently mounted server crashes. This support for replicated filesystems is available only at mount time. Once unmounted, another location may be used for subsequent mounts of the filesystem.

If each *location* in the list shares the same *pathname* then a single *location* may be used with a comma-separated list of hostnames.

hostname1,hostname2:pathname

Sharing Mounts

If *location* is specified in the form:

hostname:pathname:subdir

hostname is the name of the server from which to mount the filesystem, *pathname* is the pathname of the directory to mount, and *subdir*, when supplied, is the name of a subdirectory to which the symbolic link is made. This can be used to prevent duplicate mounts when multiple directories in the same remote filesystem may be accessed. With a map for **/home** such as:

bart homes:/home/simpsons:bart
homer homes:/home/simpsons:homer

and a user attempting to access a file in /home/bart, *automount* mounts *homes:/home/simpsons*, but creates a symbolic link called */home/bart* to the *bart* subdirectory in the temporarily-mounted filesystem. If a user immediately tries to access a file in */home/homer*, *automount* needs only to create a symbolic link that points to the *homer* subdirectory; */home/simpsons* is already mounted.

With the following map:

bart homes:/home/simpsons/bart
homer homes:/home/simpsons/homer

automount would have to mount the filesystem twice.

Comments and Quoting

A mapping can be continued across input lines by escaping the NEWLINE with a backslash. Comments begin with a # and end at the subsequent NEWLINE.

Characters that have special significance to the automount map parser may be protected either with double quotes (") or by escaping with a backslash (\). Pathnames with embedded whitespace, colons (:) or dollar ($) should be protected.

Directory Pattern Matching

The '&' character is expanded to the value of the *key* field for the entry in which it occurs. In this case:

bart homes:/home/simpsons:&

the & expands to *bart*. The '*' character, when supplied as the *key* field, is recognized as the catch-all entry. Such an entry will be used if any previous entry has not successfully matched the key being searched for. For instance, if the following entry appeared in the indirect map for */home*:

*** &:/home/&**

this would allow automatic mounts in */home* of any remote filesystem whose location could be specified as:

hostname:/home/hostname

Multiple Mounts

A multiple mount entry takes the form:

key [/[mountpoint [–mount-options] location ...] ...

The initial / within the */[mountpoint]* is required; the optional *mountpoint* is taken as a pathname relative to the destination of the symbolic link for *key*. If *mountpoint* is omitted in the first occurrence, a mount point of / is implied.

Given the direct map entry:

```
/tools \
  /          -ro dill:/tools          mint:/tools \
  /1.0       -ro mint:/tools/1.0      dill:/tools/1.0 \
  /1.0/man   -ro dill:/tools/1.0/man  mint:/tools/1.0/man
```

automount would automatically mount */tools*, */tools/1.0* and */tools/1.0/man*, as needed, from either *dill* or *mint*, whichever host responded first. If the mounts are hierarchically related, mounts closer to the root must appear before submounts. All the mounts of a multiple mount entry will occur together and will be unmounted together. This is important if the filesystems reference each other with relative symbolic links. Multiple mount entries can be used both in direct maps and in indirect maps.

Included Maps

The contents of another map can be included within a map with an entry of the form:

+mapname

mapname can either be a filename, or the name of a YP map, or one of the special maps described below. If the key being searched for is not located in an included map, the search continues with the next entry.

Special Maps

There are two special maps currently available: *–hosts*, and *–null*. The *–hosts* map uses the YP *hosts.byname* map to locate a remote host when the hostname is specified. This map specifies mounts of all exported filesystems from any host. For instance, if the following *automount* command is already in effect:

automount /net –hosts

then a reference to */net/lambada/usr* would initiate an automatic mount of all filesystems from *lambada* that *automount* can mount; references to a directory under */net/lambada* will refer to the corresponding directory relative to *lambada*'s root.

The *–null* map, when indicated on the command line, cancels any subsequent map for the directory indicated. It can be used to cancel a map given in *auto.master* or for a mount point specified as an entry in a direct map.

Configuration and the auto.master Map

automount normally consults the *auto.master* YP configuration map for a list of initial automount maps, and sets up automatic mounts for them in addition to those given on the command line. If there are duplications, the command-line arguments take precedence over a local –f master map and they both take precedence over a YP *auto.master* map. This configuration database contains arguments to the automount command, rather than mappings; unless –f is in effect, *automount* does *not* look for an *auto.master* file on the local host.

Maps given on the command line, or those given in a local *auto.master* file specified with –f override those in the YP *auto.master* map. For instance, given the command:

automount –f /etc/auto.master /home –null /– /etc/auto.direct

and a file named */etc/auto.master* that contains:

/home auto.home

automount would ignore the */home* entry in */etc/auto.master*.

FILES

/tmp_mnt
: directory under which filesystems are dynamically mounted

SEE ALSO

mount(1M), network(1M)

NOTE

The –hosts map must mount all the exported filesystems from a server. If frequent access to just a single filesystem is required it is more efficient to access the filesystem with a map entry that is tailored to mount just the filesystem of interest.

When it receives signal number 1, SIGHUP, *automount* rereads the */etc/mtab* file to update its internal record of currently-mounted filesystems. If a filesystem mounted with *automount* is unmounted with the *umount*(1M) command, *automount* should be forced to reread the file.

An *ls*(1) listing of the entries in the directory for an indirect map shows only the symbolic links for currently mounted filesystems. This restriction is intended to avoid unnecessary mounts as a side effect of programs that read the directory and *stat*(2) each of the names.

Mount points for a single automounter must not be hierarchically related. *automount* will not allow an automount mount point to be created within an automounted filesystem.

automount must not be terminated with the SIGKILL signal. Without an opportunity to unmount itself, the automount mount points will appear to the kernel to belong to a non-responding NFS server. The recommended way to terminate automount services is to send a SIGTERM signal to the daemon:

```
/etc/killall -TERM automount
```

This allows the automounter to catch the signal and unmount not only its daemon but also any mounts in */tmp_mnt*. Mounts in */tmp_mnt* that are busy will not be unmounted.

Since each direct map entry results in a separate mount for the mount daemon such maps should be kept short. Entries added to a direct map will have no effect until the automounter is restarted.

Entries in both direct and indirect maps can be modified at any time. The new information will be used when *automount* next uses the map entry to do a mount. *automount* does not cache map entries.

BUGS

The bg mount option is not recognized by the automounter.

Since *automount* is single-threaded, any request that is delayed by a slow or non-responding NFS server will delay all subsequent automatic mount requests until it completes.

Programs that read */etc/mtab* and then touch files that reside under automatic mount points will introduce further entries to the file.

NAME

bootparamd – boot parameter server

SYNOPSIS

/usr/etc/rpc.bootparamd [**–d**] [**–i**]

DESCRIPTION

bootparamd is a server process that provides information to diskless clients necessary for booting. It consults the **bootparams** database. If the client is not found there, or if the Yellow Pages service is not running, then the **/etc/bootparams** file is consulted.

bootparamd can be invoked either by **inetd**(1M) or by the user.

OPTIONS

–d Display the debugging information.

–i Ignore inter-domain "whoami" requests. If instances of *ypserv*(1M) on directly connected networks are invoked with the **-i** option, and if hosts in the local Yellow Pages domain have primary hostnames formed by concatenating a name containing no periods, a period, and the YP domain name, use **–i** with **bootparamd.**

NOTES

In the absence of **–i**, instances of **bootparamd** in different domains may receive a "whoami" broadcast, consult YP to find the requester's hostname by its address, receive the answer from a different domain (owing to **ypserv -i**), and reply with the wrong domain name.

FILES

/etc/bootparams

SEE ALSO

bootparams(4), **inetd**(1M), **ypserv**(1M)

NAME

cl_init – init program for diskless software installation

SYNOPSIS

/etc/cl_init

DESCRIPTION

cl_init is the *init*(1M) program in diskless share tree. When client workstation installs its software using PROM menu, the workstation will use the share tree as the root and process 1 will be running *cl_init* instead of the regular */etc/init*.

cl_init will prompt user for the following questions:

```
Do you want to use server XXX for software installation (y/n) ?
```

where *XXX* is the default server name used in the **tapedevice** or **bootfile** PROM variables. When doing installation, this default is the name of the server that contains the share tree. If the client tree will be installed on a different server, user should answer *n* here.

```
Enter server name :
```

Enter the correct server name for client tree. *cl_init* will check the server name with the */etc/hosts* file in the share tree. If the server name is not in this host file, *cl_init* will re-prompt the user for appropriate action.

```
Do you want to install client YYY on server XXX (y/n) ?
```

where *XXX* is the server name, and *YYY* is the hostname that is set in PROM variable **hostname**, or translated from **netaddr** using *bootparamd*(1M) service.

```
Enter password for autoinst :
```

If the *autoinst* login entry at server machine requires password, user should key in the password at this prompt. Once the password is accepted, the *inst(1M)* menu will appear on the screen.

```
Ready to exit (y/n) ?
```

cl_init will repeat the installation procedure again if the answer is *n*. Otherwise, it will set PROM variables, *netaddr* and *bootfile*, and reboot itself.

cl_init always login the server using default login name *autoinst*. User can set environment variable *dllogin* in the PROM to change the login name. For example, typing

```
setenv dllogin root
```

at prom manual mode will cause the diskless workstation login the server using *root*.

When using default login - *autoinst*, all messages before the *inst*(1M) menu are filtered. If there is a need for examining the login messages, such as for debugging purposes, it is possible to disable the feature by typing two or more escape characters before entering the password.

CAVEAT

cl_init is to be used only by diskless client installation package. If *cl_init* is invoked on a shell command line, the shell will hang. If the hung shell receives an *INTR* signal, the system will be shutdown silently.

SEE ALSO

init(1M), bootparamd(1M), inst(1M)

NAME

clinst – diskless client software installation tool

SYNOPSIS

clinst –c *class* [**–d**] **share**
clinst –c *class* **–h** *host* [**–d**] **client**
clinst –r

DESCRIPTION

Clinst is the tool for diskless workstation users to install the system software. When multiple users are using the same version of software, only one copy is needed. This single software tree is called a **share tree** which will be shared among the users. Each user has to create an private tree that contains mostly symbolic links to the share tree except for those files that can not be shared. This private tree is called a **client tree**. A share tree represents a **class.** Each use should select the **class** when installing the client tree. *clinst* serves the purposes of both installing and removing the share or client tree.

For each class, *clinst* needs a parameter file in the directory /usr/etc/boot to supply the necessary configuration information. *Class.dat* (where *class* should be replaced by the actual name of the class) should be created in /usr/etc/boot by copying the template *clinst.dat* from the same directory and modify the parameters according to the desired local configuration. Since *clinst.dat* is a shell script that will be invoked from within the *clinst*, it should always be executable. A good practice to modify the file is to change the strings within the double quotes only. *clinst.dat* contains the following variables:

DISKLESS optional directory prefix for share root, client root, swap file and dump file

CLROOT directory name for client root

SHAREHOST hostname for the share root. If share tree locates on different server then client tree, *clinst* should be run on shared server with target **share,** and on client tree server with target **client.**

SHARE directory name for share root

SWAP name of directory where swap file will be created. The swap directory must be on the same server as client root.

SWAPSIZE size of swap file. The default is set to 20 MB.

DUMP name of directory where dump file will be created. The dump directory must be on the same server as client root.

GFXBOARD type of graphics board

CPUBOARD type of cpu board

MACH type of machine

BOOTP_DIR the home directory of *bootp*(1M)

YP indicates whether yp is used

clinst is a shell script that will call *inst*(1M) to perform the software installation. It also modified the resulting tree so that diskless workstation can be brought up correctly. *clinst* creates the swap file, dump file, and client boot parameters for each installation, it also exports the directories created to the appropriate host.

There are two modes for *clinst* to operate, the manual mode and automatic mode. In manual mode, *clinst* is used as a regular UNIX command with command line parameters described in the following sections. In automatic mode, *clinst* is invoked from remote login with user name *autoinst*. Diskless client will automatically enter this mode when software installation option is selected at prom level. Server should have an entry for *autoinst* in its password file with user id and group id set to 0 and */usr/etc/boot/rclinst*, which invokes *clinst* with –r flag, designated as shell. System administrator should determine whether the password is necessary.

OPTIONS

–c *class* Using class *class*. The file *class.dat* should exist in /usr/etc/boot directory.

–h *host* Indicates the client tree is created for workstation named *host*. The *host* must be a valid hostname, i.e. it should have already been assigned an IP address. This parameter does not have any effect when creating share tree.

–d To remove the diskless tree. The default is to install the tree.

share | client To create a share tree if *share* is specified, and to create a client tree when *client* is specified. When creating client tree, the *host* must be supplied.

EXAMPLE

```
/usr/etc/boot/clinst -c 4D20 share
```

will create a share tree for class *4D20*. The */usr/etc/boot/4D20.dat* file should exist before you run the command.

FILES

/usr/etc/boot/clinst

/usr/etc/boot/clinst.dat Template for class.dat

SEE ALSO

inst(1M), rclinst(1M)

NAME

exportfs – export and unexport directories to NFS clients

SYNOPSIS

/usr/etc/exportfs [**–aiuv**] [**–o** *options*] [**directory**]

DESCRIPTION

Exportfs makes a local directory (or file) available for mounting over the network by NFS clients. It is normally invoked at boot time by the */etc/init.d/network* script, and uses information contained in the */etc/exports* file to export a *directory* (which must be specified as a full pathname). The super-user can run *exportfs* at any time to alter the list or characteristics of exported directories. Directories that are currently exported are listed in the file */etc/xtab*.

With no options or arguments, *exportfs* prints out the list of directories currently exported.

OPTIONS

–a All. Export all directories listed in */etc/exports*, or if **–u** is specified, unexport all of the currently exported directories.

–v Verbose. Print each directory as it is exported or unexported.

–u Unexport the indicated directories.

–i Ignore the options in */etc/exports*. Normally, *exportfs* will consult */etc/exports* for the options associated with the exported directory.

–o *options*

Specify a comma-separated list of optional characteristics for the directory being exported. *Options* are described in *exports*(4).

FILES

/etc/exports	static export information
/etc/xtab	current state of exported directories
/etc/netgroup	

SEE ALSO

exports(4), netgroup(4)

WARNINGS

You cannot export a directory that is either a parent- or a sub-directory of one that is currently exported and *within the same filesystem.* It would be illegal, for example, to export both */usr* and */usr/local* if both directories resided in the same disk partition.

NAME

lockd – network lock daemon

SYNOPSIS

/usr/etc/rpc.lockd [**–t** *timeout*] [**–g** *graceperiod*]

DESCRIPTION

lockd provides the inherently stateful locking services within the stateless NFS environment. It allows the locking of records and files between applications running on different physical machines.

Locks are presently advisory only. The lock style implemented by **lockd** is that specified in the SVID (see *lockf*(3C) and *fcntl*(2)). There is no interaction between the **lockd**'s locks and *flock*(3B) style locks.

lockd is started from **inetd**(1M). It processes lock requests that are either sent locally by the kernel or remotely by another lock daemon. In the case of local lock requests for remote data, **lockd** forwards the lock requests to the server site's lock daemon through the RPC/XDR(3R) package. **lockd** then requests the local status monitor daemon, **statd**(1M), for monitor service of the server. The reply to the lock request will not be sent to the kernel until the status daemon and the server site's lock daemon have replied.

When a server recovers, it waits for a grace period for all client site **lockd**s to submit reclaim requests. Client site **lockd**s are notified by the **statd** of the server recovery and promptly resubmit previously granted lock requests. If a **lockd** fails to secure a previously granted lock at the server site, it sends SIGUSR1 to the application process.

OPTIONS

–t *timeout* *lockd* uses *timeout* (seconds) as the interval instead of the default value (15 seconds) to retransmit lock request to the remote server.

–g *graceperiod*
lockd uses *graceperiod* (seconds) as the grace period duration instead of the default value (45 seconds).

NOTE

The reply to a lock request for remote data is delayed until all daemons become available.

In the Sun implementation, **lockd** sends SIGLOST. The IRIX implementation sends SIGUSR1.

SEE ALSO

fcntl(2), inetd(1M), lockf(3C), signal(2), statd(1M)

NAME

makedbm – make a Yellow Pages dbm file

SYNOPSIS

/usr/etc/yp/makedbm [**–i** *yp_input_file*] [**–o** *yp_output_name*]
[**–d** *yp_domain_name*] [**–m** *yp_master_name*]
infile outfile
/usr/etc/yp/makedbm [**–u** *dbmfilename*]

DESCRIPTION

Makedbm takes *infile* and converts it to a pair of files in *dbm*(3B) format, namely *outfile*.pag and *outfile*.dir. Each line of the input file is converted to a single *dbm* record. All characters up to the first tab or space form the key, and the rest of the line is the data. If a line ends with \, then the data for that record is continued on to the next line. It is left for the clients of the Yellow Pages to interpret #; *makedbm* does not itself treat it as a comment character. *infile* can be –, in which case standard input is read.

Makedbm is meant to be used in generating *dbm* files for the Yellow Pages, and it generates a special entry with the key *yp_last_modified*, which is the date of *infile* (or the current time, if *infile* is –).

OPTIONS

–i Create a special entry with the key *yp_input_file*.

–o Create a special entry with the key *yp_output_name*.

–d Create a special entry with the key *yp_domain_name*.

–m Create a special entry with the key *yp_master_name*. If no master host name is specified, *yp_master_name* will be set to the local host name.

–u Undo a *dbm* file. That is, print out a *dbm* file one entry per line, with a single space separating keys from values.

EXAMPLE

It is easy to write shell scripts to convert standard files such as */etc/passwd* to the key value form used by *makedbm*. For example,

```
#!/usr/bin/awk -f
BEGIN { FS = ":"; OFS = "\t"; }
{ print $1, $0 }
```

takes the */etc/passwd* file and converts it to a form that can be read by *makedbm* to make the Yellow Pages file *passwd.byname*. That is, the key is a username, and the value is the remaining line in the */etc/passwd* file.

SEE ALSO

yppasswd(1M), dbm(3B)

NAME

mountd – NFS mount request server

SYNOPSIS

/usr/etc/rpc.mountd [**–n**]

DESCRIPTION

Mountd is an *rpc*(4) server that answers file system mount requests. It reads the file */etc/exports*, described in *exports*(4), to determine which file systems are available to which machines and users. It also provides information as to which clients have file systems mounted. This information can be printed using the *showmount*(1M) command.

Normally, *mountd* only accepts requests from clients using a privileged (i.e., secure) port. The **–n** option disables this check and allows *mountd* to accept requests from any port.

The *mountd* daemon is normally invoked by *inetd*(1m).

SEE ALSO

inetd(1M), showmount(1M), exports(4), services(4)

NAME

nfsd, biod – NFS daemons

SYNOPSIS

/usr/etc/nfsd [*nservers*]
/usr/etc/biod [*nservers*]

DESCRIPTION

nfsd starts the *nfs*(4) server daemons that handle client filesystem requests. *Nservers* is the number of file system request daemons to start. This number should be based on the load expected on this server. Four seems to be a good number.

Biod starts *nservers* asynchronous block I/O daemons. This command is used on a NFS client to buffer cache handle read-ahead and write-behind. The magic number for *nservers* in here is also four.

These daemons are started during system initialization from the /etc/init.d/network script if the configuration flag "nfs" is set on (see *network*(1M)).

When a file that is opened by a client is unlinked (by the server), a file with a name of the form *.nfsXXX* (where *XXX* is a number) is created by the client. When the open file is closed, the *.nfsXXX* file is removed. If the client crashes before the file can be closed, the *.nfsXXX* file is not removed.

FILES

.nfsXXX client machine pointer to an open-but-unlinked file

SEE ALSO

exportfs(1M), mountd(1M), network(1M), exports(4)

NAME

nfsstat – display Network File System statistics

SYNOPSIS

/usr/etc/nfsstat [**–csnrdz**] [*unix*] [*core*]

DESCRIPTION

Nfsstat displays statistical information about the Network File System (NFS) and Remote Procedure Call (RPC) interfaces to the kernel. It can also be used to reinitialize this information. If no options are given the default is

nfsstat –csnr

That is, print everything and reinitialize nothing. The optional arguments *unix* and *core* may be used to indicate another system namelist and kernel memory image, respectively.

OPTIONS

–c Display client information. Only the client side NFS and RPC information will be printed. Can be combined with the **–n** and **–r** options to print client NFS or client RPC information only.

–s Display server information. Works like the **–c** option above.

–n Display NFS information. NFS information for both the client and server side will be printed. Can be combined with the **–c** and **–s** options to print client or server NFS information only.

–r Display RPC information. Works like the **–n** option above.

–z Zero (reinitialize) statistics. Can be combined with any of the above options to zero particular sets of statistics after printing them. The user must have super-user privilege for this option to work.

FILES

/unix	system namelist
/dev/kmem	kernel memory

NAME

rarpd – DARPA Reverse Address Resolution Protocol daemon

SYNOPSIS

/usr/etc/rarpd [**–d**] [**–l** *logfile*] [*interface...*]

DESCRIPTION

Rarpd responds to Reverse Address Resolution Protocol (Reverse ARP, RARP) requests. It puts itself in the background, and requires root privileges.

The Reverse ARP protocol is used by machines at boot time to discover their 32-bit Internet Protocol (IP) address given their 48-bit Ethernet address. In order for a RARP request to be answered, the requesting machine's name-to-IP-address entry must exist in the **/etc/hosts** file and its name-to-Ethernet-address entry must exist in the **/etc/ethers** file. Note that if the server machine running *rarpd* is using the Yellow Pages, the server's **/etc** files are ignored and the appropriate Yellow Pages maps are queried.

Normally *rarpd* serves all configured IP interfaces which support broadcasting. Optional *interface* arguments restrict service to only those interfaces. The **–d** option causes *rarpd* to run in the foreground and log diagnostics on its standard error output. The **–l** option causes *rarpd* to record requests in *logfile*.

FILES

/usr/adm/SYSLOG	system log
/etc/init.d/network	networking start-up script
/etc/config/rarpd	configuration switch
/etc/config/rarpd.options	configuration options

SEE ALSO

chkconfig(1M), ifconfig(1M), ethers(4), hosts(4).

Finlayson, Ross, Timothy Mann, Jeffrey Mogul, and Marvin Theimer, *A Reverse Address Resolution Protocol*, RFC 903, Network Information Center, SRI International, Menlo Park, CA, June 1984.

NAME

rclinst – diskless client software auto-installation tool

SYNOPSIS

/usr/etc/boot/rclinst

DESCRIPTION

rclinst is the script that should be used for the *autoinst* entry in the /etc/passwd file. The *autoinst* entry in the password file is needed in order to enable diskless workstation to install the software from the local machine. The format of the entry should look like:

```
autoinst::0:0:/usr/etc/boot:/usr/etc/boot/rclinst
```

will invoke *clinst(1M)* using the **–r** option.

SEE ALSO

clinst(1M), cl_init(1M)

NAME

registrar – IP address update command for yp hosts data base

SYNOPSIS

/usr/etc/yp/registrar host-file "make hosts"

DESCRIPTION

registrar, which runs on yp master only, is a YP hostname data base update program. A hostname registration request can be sent from either *yp_host*(1M) command, or from the PROM IP address auto-registration function. This requests is initially sent to a rpc daemon process *rpc.ypupdated*(1M). *registrar* is invoked indirectly by *rpc.ypupdated*(1M) via an intermediate make file *updaters*(4).

The *host-file* parameter should be the host file that is used in YP data base makefile, */usr/etc/yp/Makefile*. Usually, it is /etc/hosts.

registrar knows how to receive the input parameters from *rpc.ypupdated*(1M). The input parameters tell *registrar* whether to allocate a new IP address, to change the existing hostname entry, or to delete the entry, along with the necessary information to service the request.

When adding new hostname entry, there is no authentication checking. As long as the new hostname and the aliases are not yet used, the registration request will always be executed. When changing or deleting hostname entry, the yp master root password has to be passed along with the command.

The new IP address will be selected with the first available address that is in the same network/subnet specified in the request. System administrator can, however, mark a specific range in the */etc/hosts* for address allocation by a special comment line. The format of this line should be

 # registrar start=xxx end=yyy mask=0xzzzzzzzz

where *xxx* is the IP address with the smallest local network address in the range, *yyy* is the IP address with largest local network address in the range, and *zzzzzzzz* is the 8-byte hexadecimal mask value. The *xxx* and *yyy* must fall into the same network, or subnet if netmask is specified.

There can be several lines of comment for the address allocation for the same network or subnet. The *registrar* will use the first available address by searching through the comment lines sequentially. In case all address ranges described by the comment lines are used, *registrar* will choose the lowest available IP address that is in the same network or subnet.

FILES

/etc/hosts
/usr/etc/yp/updaters

EXAMPLE

Use

#registrar start=192.26.1.10 end=192.26.1.20 mask=0xffffff00

to reserve 192.26.1.10 through 192.26.1.20 for network 192.26.1.

SEE ALSO

yp_host(1M), updaters(4), ypupdated(1M)

NAME

rexd – RPC-based remote execution server

SYNOPSIS

/usr/etc/rpc.rexd

DESCRIPTION

rexd is the *rpc*(4) server for remote program execution. This daemon is started by *inetd*(1M) whenever a remote execution request is made (see the note below). For non-interactive programs standard file descriptors are connected directly to TCP connections. Interactive programs involve pseudo-terminals, similar to the login sessions provided by *rlogin(1C)*. This daemon may use the NFS to mount file systems specified in the remote execution request.

rexd should be used on trusted networks only. It is not automatically enabled in the */usr/etc/inetd.conf* file. To enable *rexd*, edit inetd.conf and remove the comment character preceding the rexd entry and signal *inetd* to read the file:

```
/etc/killall -HUP inetd
```

DIAGNOSTICS

Diagnostic messages are logged to *syslogd*(1M), and returned to the requester.

RESTRICTIONS

The super-user cannot execute commands using *rexd* client programs such as *on*(1C).

FILES

/dev/ttyq*n*	pseudo-terminals used for interactive mode.
/etc/passwd	authorized users.
/usr/tmp_rex/rexd?????	temporary mount points for remote file systems.

SEE ALSO

on(1C), exports(4), rpc(4), inetd(1M)

BUGS

Access control is not secure.

Does not properly handle window size information sent by Sun workstations.

NAME

rpc.passwd – server for modifying Yellow Pages password file

SYNOPSIS

/usr/etc/rpc.passwd *file* [**–m** *arg1 arg2* ...]

DESCRIPTION

rpc.passwd is a server that handles password change requests from *yppasswd*(1). It changes a password entry in *file*, which is assumed to be in the format of *passwd*(4). An entry in *file* will only be changed if the password presented by *yppasswd*(1) matches the encrypted password of that entry.

If the **–m** option is given, then after *file* is modified, a *make*(1) will be performed in */usr/etc/yp*. Any arguments following the flag will be passed to *make*.

This server should be run on the host serving as the Yellow Pages master. It is started from the */etc/init.d/network* startup script if the "yp" and "ypmaster" configuration flags are set on (see *network*(1M)). The startup script invokes the server using */etc/passwd* as the Yellow Pages password file and causes password changes to be propagated immediately. To use a different YP passwd file, put the file's name in */etc/config/rpc.passwd.options* and change the PWFILE variable in */usr/etc/yp/Makefile*.

FILES

/usr/etc/yp/Makefile

SEE ALSO

network(1M), ypmake(1M), yppasswd(1), passwd(4), ypfiles(4)

CAVEAT

This server will eventually be replaced with a more general service for modifying any map in the Yellow Pages

NAME

rstatd – kernel statistics server

SYNOPSIS

/usr/etc/rpc.rstatd

DESCRIPTION

Rstatd is an *rpc*(4) server which returns performance statistics obtained from the kernel. The *rstatd* daemon is normally invoked by *inetd*(1M).

SEE ALSO

inetd(1M)

NAME

rusersd – network username server

SYNOPSIS

/usr/etc/rpc.rusersd

DESCRIPTION

Rusersd is an *rpc*(4) server that returns a list of users on the network. The *rusersd* daemon is normally invoked by *inetd*(1M).

SEE ALSO

rusers(1C), services(4), inetd(1M)

NAME

rwalld – network rwall server

SYNOPSIS

/usr/etc/rpc.rwalld

DESCRIPTION

Rwalld is a server that handles *rwall*(1) and *shutdown*(1) requests. It is implemented by calling *wall*(1) to all the appropriate network machines. The *rwalld* daemon is normally invoked by *inetd*(1M).

SEE ALSO

rwall(1), wall(1), inetd(1M)

NAME

showmount – show all remote mounts

SYNOPSIS

/etc/showmount [**–a**] [**–d**] [**–e**] [**–x**] [host]

DESCRIPTION

Showmount lists all the clients that have remotely mounted a filesystem from *host*. This information is maintained by the *mountd*(1M) server on *host*, and is saved across crashes in the file */etc/rmtab*. The default value for *host* is the value returned by *hostname*(1).

OPTIONS

–a Print all remote mounts in the format *hostname:directory* where *hostname* is the name of the client, and *directory* is the root of the file system that has been mounted.

–d List directories that have been remotely mounted by clients.

–e Print the list of exported file systems. For each file system, list the clients given mount access.

–x Print the list of exported file systems and all of their export options, in the format described by *exports*(4). This option option overrides the **–e** option.

SEE ALSO

mountd(1M), exportfs(1M), exports(4)

BUGS

If a client crashes, its entry will not be removed from */etc/rmtab* until it reboots and executes *umount –a*.

NAME

spray – spray packets

SYNOPSIS

/usr/etc/spray *host* [**–c** *count*] [**–d** *delay*] [**–i** *delay*] [**–l** *length*]

DESCRIPTION

spray sends a one-way stream of packets to *host* using RPC, and then reports how many were received by *host* and what the transfer rate was. The host name can be either a name or an Internet address.

OPTIONS

–c *count* Specifies how many packets to send. The default value of *count* is the numbers of packets required to make the total stream size 100000 bytes.

–d *delay* Specifies how may microseconds to pause between sending each packet. The default is 0.

–i Use ICMP echo packets rather than RPC. Since ICMP automatically echos, this creates a two way stream.

–l *length* The *length* parameter is the numbers of bytes in the ethernet packet that holds the RPC call message. Since the data is encoded using XDR, and XDR only deals with 32 bit quantities, not all values of *length* are possible, and *spray* rounds up to the nearest possible value. When *length* is greater than 1514, then the RPC call can no longer be encapsulated in one Ethernet packet, so the *length* field no longer has a simple correspondence to Ethernet packet size. The default value of *length* is 86 bytes (the size of the RPC and UDP headers)

SEE ALSO

icmp(7P), ping(1M), sprayd(1M)

NAME

sprayd – spray server

SYNOPSIS

/usr/etc/rpc.sprayd

DESCRIPTION

rpc.sprayd is a server which records the packets sent by *spray*(1M). The *rpc.sprayd* daemon is normally invoked by *inetd*(1M).

SEE ALSO

inetd(1M), spray(1M).

NAME

statd – network status monitor daemon

SYNOPSIS

/usr/etc/rpc.statd

DESCRIPTION

statd is an intermediate version of the status monitor. It implements a simple protocol which allows applications to monitor the status of other machines. *lockd*(1M) uses *statd* to detect both client and server failures.

statd is started during system initialization if the *chkconfig*(1M) "lockd" flag is set on.

Applications use RPC to register machines they want monitored by *statd*. The status monitor maintains a database of machines to track and the corresponding applications to notify of crashes. It also maintains a database of machines to notify upon recovery of its own host machine and a counter of the number of times it has "recovered".

FILES

/usr/etc/statd.d/sm	machines to monitor
/usr/etc/statd.d/sm.bak	machines to notify upon recovery
/usr/etc/statd.d/state	recovery counter (a.k.a. version number)

SEE ALSO

network(1M), lockd(1M), statmon(4)

BUGS

The crash of a site is only detected upon its recovery.

NAME

updbootparam – YP bootparams database update program

SYNOPSIS

/usr/etc/yp/updbootparam bootparams-file "make bootparams"

DESCRIPTION

Updbootparam, which runs on yp master only, is the YP *bootparams*(4) update program that will modify the data base upon request. An update request is sent from *yp_bootparam*(1M) command using rpc call. *Updbootparam* is invoked indirectly by rpc daemon *rpc.ypupdated*(1M) via an intermediate makefile *updaters*(4).

Updbootparam is designed to be used in the make file *updaters*(4). The parameter **bootparams-file** should be the bootparams file used in YP data base makefile, */usr/etc/yp/Makefile*. Usually, it is */etc/bootparams*. The update request tells *updbootparam* whether to add or delete an bootparams entry.

FILES

/usr/etc/yp/updaters

SEE ALSO

bootparams(4), updaters(4), ypupdated(1M)

NAME

yp_bootparam – update yp bootparams data base

SYNOPSIS

yp_bootparam **–h** *host* **–a** [**–b**] [**–k** *key_file*] **–f** *file*
yp_bootparam **–h** *host* **–a** [**–b**] [**–k** *key_file*] params
yp_bootparam **–h** *host* **–d** [**–b**] [**–k** *key_file*]

DESCRIPTION

yp_bootparam is the user interface tool to update yp bootparams(4) data base directly on client workstation without running on ypmaster. *yp_bootparam* uses the *ypupdated*(1M) service on ypmaster to update the bootparams data base.

User can either add an entry to data base, or delete an entry. As long as the hostname is registered in the yp hosts data base and the bootparam entry does not exist, user can always add an entry to the bootparams data base. In this process, a security key can be requested on return. When deleting an entry, the same key (if there is one) must be submitted in the same command line. Otherwise, the request will be rejected. The update daemon on ypmaster stores the keys in /usr/etc/boot/keystore, while *yp_bootparam* saves the key in the file specified by the input parameter.

OPTIONS

–h *host* Use *host* as the key in bootparams data base. The *host* will be checked against yp hosts data base unless *–b* is specified. If *host* is an alias, the real key found in hosts data base will be used instead.

–a To add an entry

–d To remove an entry.

–b use *host* as key without checking yp hosts data base.

–k *key_file* When adding an entry, the *key_file* is used to save the returned key. When deleting an entry, it is used to pass the original key.

–f *file* The *file* will contain the data part in the bootparam entry. It should have the following format:

```
root=root_server:root_path
share=share_server:share_path
swap=root_server:swap_path
dump=root_server:dump_path
```

The dump value is optional.

params If *–f* is not used, the boot parameters should be passed at command level.

EXAMPLE

yp_bootparam –a –h bonnie –k /mykey
root=clyde:/bonnie share=clyde:/share
swap=clyde:/swap/bonnie

will create a bootparam entry in ypmaster. The security key returned will be left in /mykey.

FILES

/usr/etc/yp/yp_bootparam
/usr/etc/boot/keystore
/usr/include/rpcsvc/ypclnt.h Error code listing

SEE ALSO

bootparams(4), updbootparam(1M), ypupdated(1M)

NAME

yp_host – update yp hosts data base

SYNOPSIS

yp_host **–r** **–h** *host* [**–n** *net* [**–m** *mask*]] [**–a** *aliases*]
yp_host **–c** **–h** *host* **–w** *newname* [**–a** *aliases*]
yp_host **–d** **–h** *host*

DESCRIPTION

yp_host is the user command to update yp hosts data base directly on client workstation without running on ypmaster. *yp_host* uses the *ypupdated*(1M) service on ypmaster to update the hosts data base.

Users can add an entry to data base, change existing data base or delete an entry. As long as the host name is not used in the current data base, there are no restrictions for creating an entry. However, to modify or delete an entry, users will be prompted for the root password of ypmaster.

yp_host is provided to avoid using editor on /etc/hosts directly when modifying the hosts file. The single threaded nature of *ypupdated*(1M) guarantees the data base is consistent under multiple updating. In network with Silicon Graphics diskless workstations, this feature is especially important because the automatic registration request will be received by ypmaster at random time due to installation of a new diskless workstation.

OPTIONS

–r To register the host name in yp hosts data base.

–h *host* Use *host* as the key in updating hosts data base.

–n *net* To specify the network that *host* will be in. This parameter should follow the "." notation of Internet address. If **–m** is used, the *net* should be a four bytes Internet address. If **–m** is not used, the *net* should be an Internet network number.

–m *mask* To specify the network mask that will be used. This mask should have the form of *0xffffffff*, where "f" must be a valid hexadecimal character. This parameter "and"ed with *net* represents the target Internet subnet number.

–a *aliases* To specify the aliases of the new host. Multiple aliases should be quoted (") in command line.

–c To change the existing entry.

–w *newname* To modify the entry to use *newname* as the new key for hosts data base.

–d To delete an entry.

EXAMPLE

yp_host –a –h bonnie –n 192.26.88 –a "bonnie.1 bonnie.2" will create an entry in ypmaster. The network that new IP address will be using is "192.26.88".

yp_host –a –h bonnie –n 192.26.88.200 –m 0xffffffc0 –a "bonnie.1 bonnie.2". The subnet that new IP address will be in is "192.26.88.192".

FILES

/etc/hosts
/usr/etc/yp/yp_host
/usr/include/rpcsvc/ypclnt.h Error code listing

SEE ALSO

registrar(1M), ypupdated(1M)

NAME

ypinit – build and install Yellow Pages database

SYNOPSIS

/usr/etc/yp/ypinit –m
/usr/etc/yp/ypinit –s master_name

DESCRIPTION

ypinit sets up a Yellow Pages database on a YP server. It can be used to set up a master or a slave server. You must be the superuser to run it. It asks a few, self-explanatory questions, and reports success or failure to the terminal.

It sets up a master server using the simple model in which that server is master to all maps in the data base. This is the way to bootstrap the YP system; later if you want you can change the association of maps to masters. All databases are built from scratch, either from information available to the program at runtime, or from the ASCII data base files in */etc*. These files are listed below under **FILES** . All such files should be in their "traditional" form, rather than the abbreviated form used on client machines.

A YP database on a slave server is set up by copying an existing database from a running server. The *master_name* argument should be the hostname of YP server (either the master server for all the maps, or a server on which the data base is up-to-date and stable).

Refer to *ypfiles*(4) and *ypserv*(1m) for an overview of the Yellow Pages.

OPTIONS

–m Indicates that the local host is to be the YP master.

–s Set up a slave database.

FILES

/etc/passwd
/etc/group
/etc/hosts
/etc/networks
/etc/services
/etc/protocols
/etc/netgroup
/etc/ethers

SEE ALSO

makedbm(1M), ypfiles(4), yppush(1M), ypxfr(1M), ypmake(1M), ypserv(1M)

NAME

ypmake – rebuild Yellow Pages database

SYNOPSIS

cd /usr/etc/yp; ypmake [*map*]

DESCRIPTION

On YP master machines, the file called *Makefile* in */usr/etc/yp* is used by **ypmake** to build the Yellow Pages databases. With no arguments, **ypmake** creates *dbm*(3B) databases for any YP maps that are out-of-date, and then executes *yppush*(1M) to notify slave servers that there has been a change.

If invoked with *map*, **ypmake** will update that map only. Typing *ypmake passwd* will create and *yppush* the password database (assuming it is out of date). Likewise, *ypmake hosts* and *ypmake networks* will create databases from the host and network files, */etc/hosts* and */etc/networks* and *yppush* the databases to the slave servers.

cron(1M) executes **ypmake** at regular intervals in order to maintain consistency between YP servers' databases. Once a day, **ypmake** rebuilds and transfers copies of all of the YP databases to the slave servers, and moves the log file */usr/etc/yp/ypmake.log* to */usr/etc/yp/ypmake.log.old* to keep it from growing too large.

ypmake reads the file */etc/config/ypmaster.options* so that users may configure the following variables:

ALIASES
full pathname of the aliases file used to build the aliases database. (Default location is /usr/lib/aliases.)

DIR the directory of the source files. (Default is /etc.)

DOM used to construct a domain other than the master's default domain;

NOPUSH
when non-null, inhibits doing a *yppush* of the new database files. (Default is the null string.)

PWFILE
full pathname of the password file used to build the passwd database. (Default location is /etc/passwd.)

YPDIR directory containing YP programs (e.g., makedbm, yppush, etc.). (Default is /usr/etc/yp.)

For instance, to change the location of the password file used by **ypmake** to */etc/passwd.yp*, include:

PWFILE=/etc/passwd.yp

in */etc/config/ypmaster.options*.

Refer to *ypfiles*(4) and *ypserv*(1M) for an overview of the Yellow Pages.

FILES

/usr/etc/yp/ypmake.log

SEE ALSO

cron(1M), make(1), makedbm(1M), ypserv(1M)

NAME

yppoll – what version of a YP map is at a YP server host

SYNOPSIS

/usr/etc/yp/yppoll [**–h** *host*] [**–d** *domain*] *mapname*

DESCRIPTION

Yppoll asks a *ypserv* process what the order number is, and which host is the master YP server for the named map. If the server is a v.1 YP protocol server, *yppoll* uses the older protocol to communciate with it. In this case, it also uses the older diagnostic messages in case of failure.

OPTIONS

–h *host* Ask the *ypserv* process at *host* about the map parameters. If *host* isn't specified, the YP server for the local host is used. That is, the default host is the one returned by *ypwhich*(1M).

–d *domain* Use *domain* instead of the default domain.

SEE ALSO

ypserv(1M), ypfiles(4)

NAME

yppush – force propagation of a changed YP map

SYNOPSIS

/usr/etc/yp/yppush [**–d** *domain*] [**–v**] *mapname*

DESCRIPTION

Yppush copies a new version of a Yellow Pages (YP) map from the master YP server to the slave YP servers. It is normally run only on the master YP server by the *Makefile* in */usr/etc/yp* after the master databases are changed. It first constructs a list of YP server hosts by reading the YP map *ypservers* within the *domain*. Keys within the map *ypservers* are the ASCII names of the machines on which the YP servers run.

A "transfer map" request is sent to the YP server at each host, along with the information needed by the transfer agent (the program which actually moves the map) to call back the *yppush* . When the attempt has completed (successfully or not), and the transfer agent has sent *yppush* a status message, the results may be printed to stdout. Messages are also printed when a transfer is not possible; for instance when the request message is undeliverable, or when the timeout period on responses has expired.

Refer to *ypfiles*(4) and *ypserv*(1M) for an overview of the Yellow Pages.

OPTIONS

–d Specify a *domain*.

–v Verbose. This causes messages to be printed when each server is called, and for each response. If this flag is omitted, only error messages are printed.

FILES

/usr/etc/yp/*domainname*/ypservers.{dir, pag}

SEE ALSO

ypserv(1M) ypxfr(1M), ypfiles(4)

BUGS

In the current implementation (version 2 YP protocol), the transfer agent is *ypxfr* , which is started by the *ypserv* program. If *yppush* detects that it is speaking to a version 1 YP protocol server, it uses the older protocol, sending a version 1 YPPROC_GET request and issues a message to that effect. Unfortunately, there is no way of knowing if or when the map transfer is performed for version 1 servers. *yppush* prints a message saying that an "old-style" message has been sent. The system administrator should later check to see that the transfer has actually taken place.

NAME

ypserv, ypbind – Yellow Pages server and binder processes

SYNOPSIS

/usr/etc/ypserv [**–iv**] [**–L** *logflags*]
/usr/etc/ypbind

DESCRIPTION

The Yellow Pages (YP) provides a simple network lookup service consisting of databases and processes. The databases are files in a directory tree rooted at */usr/etc/yp*. These files are described in *ypfiles*(4). The processes are **/usr/etc/ypserv**, the YP database lookup server, and **/usr/etc/ypbind**, the YP binder. The programmatic interface to YP is described in *ypclnt*(3N). Administrative tools are described in *yppush*(1M) *ypxfr*(1M) *yppoll*(1M) *ypwhich*(1), and *ypset*(1M). Tools to see the contents of YP maps are described in *ypcat*(1), and *ypmatch*(1). Database generation and maintenance tools are described in *ypinit*(1M), *ypmake*(1M), and *makedbm*(1M).

Both **ypbind** and **ypserv** are daemon processes typically activated at system startup time from */etc/init.d/network* if the configuration flags "yp" and "ypserv" are set on (see *network*(1M)). The *yp* configuration state must be on for *ypserv* to be on.

ypserv runs only on YP server machines with a complete YP database. **ypbind** runs on all machines using YP services, both YP servers and clients.

The **ypserv** daemon's primary function is to look up information in its local database of YP maps. The operations performed by **ypserv** are defined for the implementor by the *YP protocol specification*, and for the programmer by the header file *<rpcsvc/yp_prot.h>*. Communication to and from **ypserv** is by means of RPC calls. Lookup functions are described in *ypclnt*(3N), and are supplied as C-callable funtions in */usr/lib/libsun.a*. There are four lookup functions, all of which are performed on a specified map within some YP domain: *Match*, *Get_first*, *Get_next*, and *Get_all*. The *Match* operation takes a key, and returns the associated value. The *Get_first* operation returns the first key-value pair from the map, and *Get_next* can be used to enumerate the remainder. *Get_all* ships the entire map to the requester as the response to a single RPC request.

Two other functions supply information about the map, rather than map entries: *Get_order_number*, and *Get_master_name*. In fact, both order number and master name exist in the map as key-value pairs, but the server will not return either through the normal lookup functions. (If you examine the map with *makedbm*(1M), however, they will be visible.) Other functions are used within the YP subsystem itself, and are not of general interest

to YP clients. They include *Do_you_serve_this_domain?*, *Transfer_map*, and *Reinitialize_internal_state*.

The function of **ypbind** is to remember information that lets client processes on a single node communicate with some **ypserv** process.

ypbind must run on every machine which has YP client processes; **ypserv** may or may not be running on the same node, but must be running somewhere on the network.

The information **ypbind** remembers is called a *binding* — the association of a domain name with the internet address of the YP server, and the port on that host at which the **ypserv** process is listening for service requests. The process of binding is driven by client requests. As a request for an unbound domain comes in, the **ypbind** process broadcasts on the net trying to find a **ypserv** process that serves maps within that domain. Since the binding is established by broadcasting, there must be at least one **ypserv** process on every net. Once a domain is bound by a particular **ypbind**, that same binding is given to every client process on the node. The **ypbind** process on the local node or a remote node may be queried for the binding of a particular domain by using the *ypwhich*(1) command.

Bindings are verified before they are given out to a client process. If **ypbind** is unable to speak to the **ypserv** process it's bound to, it marks the domain as unbound, tells the client process that the domain is unbound, and tries to bind the domain once again. Requests received for an unbound domain will fail immediately. In general, a bound domain is marked as unbound when the node running **ypserv** crashes or gets overloaded. In such a case, **ypbind** will to bind any YP server (typically one that is less-heavily loaded) available on the net.

ypbind also accepts requests to set its binding for a particular domain. The request is usually generated by the YP subsystem itself. *ypset*(1M) is a command to access the *Set_domain* facility. It is for unsnarling messes, not for casual use.

YPSERV OPTIONS

–f *forklimit*
: limits the number of processes **ypserv** can fork at any given time. (The default is 20.)

–i
: allows **ypserv** to resolve non-local host name and address lookups with the 4.3BSD Internet domain name server, *named*(1M).

–L *logflags*
: specifies the type(s) of information to be logged in */usr/etc/yp/ypserv.log* (see below), in addition to error messages. *logflags* is a comma-separated list of one or more of: **dispatch**,

interdomain and **querycache**.

–v "Verbose" – display messages to stderr instead of the logfile.

FILES

If the file */usr/etc/yp/ypserv.log* exists when **ypserv** starts up, log information will be written to this file.

SEE ALSO

named(1M), network(1M), ypcat(1), ypmatch(1), yppush(1M), ypwhich(1), ypxfr(1M), ypset(1M), ypclnt(3N), ypfiles(4), YP protocol specification

NAME

ypset – point ypbind at a particular server

SYNOPSIS

/usr/etc/yp/ypset [**–V1**|**–V2**] [**–h** *host*] [**–d** *domain*] *server*

DESCRIPTION

Ypset tells *ypbind* to get YP services for the specified *domain* from the *ypserv* process running on *server*. If *server* is down, or isn't running *ypserv*, this is not discovered until a YP client process tries to get a binding for the domain. At this point, the binding set by *ypset* will be tested by *ypbind*. If the binding is invalid, *ypbind* will attempt to rebind for the same domain.

Ypset is useful for binding a client node which is not on a broadcast net, or is on a broadcast net which isn't running a YP server host. It also is useful for debugging YP client applications, for instance where a YP map only exists at a single YP server host.

In cases where several hosts on the local net are supplying YP services, it is possible for *ypbind* to rebind to another host even while you attempt to find out if the *ypset* operation succeeded. That is, you can type "ypset host1", and then "ypwhich", which replies: "host2", which can be confusing. This is a function of the YP subsystem's attempt to load-balance among the available YP servers, and occurs when *host1* does not respond to *ypbind* because it is not running *ypserv* (or is overloaded), and *host2*, running *ypserv*, gets the binding.

Server indicates the YP server to bind to, and can be specified as a name or an IP address. If specified as a name, *ypset* will attempt to use YP services to resolve the name to an IP address. This will work only if the node has a current valid binding for the domain in question. In most cases, *server* should be specified as an IP address.

Refer to *ypfiles*(4) and *ypserv*(1M) for an overview of the Yellow Pages.

OPTIONS

–V1 Bind *server* for the (old) v.1 YP protocol.

–V2 Bind *server* for the (current) v.2 YP protocol.

If no version is supplied, *ypset*, first attempts to set the domain for the (current) v.2 protocol. If this attempt fails, *ypset*, then attempts to set the domain for the (old) v.1 protocol.

–h *host* Set ypbind's binding on *host*, instead of locally. *host* can be specified as a name or as an Internet address.

-d *domain*
Use *domain* instead of the default domain.

SEE ALSO
ypwhich(1), ypserv(1M), ypfiles(4)

NAME

ypupdated – server for changing YP information

SYNOPSIS

/usr/etc/rpc.ypupdated

DESCRIPTION

ypupdated is a daemon that updates information in the Yellow Pages, normally started up by *inetd*(1M). *ypupdated* consults the file *updaters*(4) in the directory */usr/etc/yp* to determine which YP maps should be updated and how to change them.

By default, the daemon requires the most secure method of authentication available to it, which currently is AUTH_UNIX. The DES authentication method is not implemented at the time.

FILES

/usr/etc/yp/updaters

SEE ALSO

inetd(1M), *updaters*(4)

BUGS

Access control is insecure. Use only on a trusted network.

NAME

ypxfr – transfer a YP map from some YP server to here

SYNOPSIS

/usr/etc/yp/ypxfr [**–f**] [**–h** *host*] [**–d** *domain*]
[**–c**] [**–C** *tid prog ipadd port*] *mapname*

DESCRIPTION

Ypxfr moves a YP map to the local host by making use of normal YP services. It creates a temporary map in the directory /usr/etc/yp/*domain* (which must already exist), fills it by enumerating the map's entries, fetches the map parameters (master and order number) and loads them. It then deletes any old versions of the map and moves the temporary map to the real mapname.

If *ypxfr* is run interactively, it writes its output to the terminal. However, if it's invoked without a controlling terminal, and if the log file */usr/etc/yp/ypxfr.log* exists, it will append all its output to that file. Since *ypxfr* is run from */usr/spool/cron/crontabs/root*, or by *ypserv*, you can use the log file to retain a record of what was attempted, and what the results were.

For consistency between servers, *ypxfr* should be run periodically for every map in the YP data base. Different maps change at different rates: the *services.byname* map may not change for months at a time, for instance, and may therefore be checked only once a day in the wee hours. You may know that *mail.aliases* or *hosts.byname* changes several times per day. In such a case, you may want to check hourly for updates. A *crontab* entry can be used to perform periodic updates automatically (see *cron*(1M). Rather than having a separate *crontab* entry for each map, you can group comands to update several maps in a shell script. Examples are in */usr/etc/yp*: *ypxfr_1pd.sh*, (transfer once per day) *ypxfr_2pd.sh*, (transfer twice per day) and *ypxfr_1phr.sh* (transfer once per hour). They can serve as reasonable first cuts.

Refer to *ypfiles*(4) and *ypserv*(1M) for an overview of the Yellow Pages.

OPTIONS

–f Force the transfer to occur even if the version at the master is not more recent than the local version.

–c Don't send a "Clear current map" request to the local *ypserv* process. Use this flag if *ypserv* is not running locally at the time you are running *ypxfr*. Otherwise, *ypxfr* will complain that it can't talk to the local *ypserv*, and the transfer will fail.

–h *host* Get the map from *host*, regardless of what the map says the master is. If *host* is not specified, *ypxfr* will ask the

YP service for the name of the master, and try to get the map from there. *host* may be a name or an Internet address in the form *a.b.c.d* (see *inet*(3N)).

–d *domain* Specify a domain other than the default domain.

–C *tid prog ipadd port*

This option is **only** for use by *ypserv*. When *ypserv* invokes *ypxfr*, it specifies that *ypxfr* should call back a *yppush* process at the host with Internet address *ipaddr*, registered as program number *prog*, listening on port *port*, and waiting for a response to transaction *tid*.

FILES

/usr/etc/yp/ypxfr.log
/usr/etc/yp/ypxfr_1pd.sh
/usr/etc/yp/ypxfr_2pd.sh
/usr/etc/yp/ypxfr_1ph.sh
/usr/spool/cron/crontabs/root

SEE ALSO

ypserv(1M), yppush(1M), ypfiles(4)

NAME

nfssvc, async_daemon – NFS daemons

SYNOPSIS

nfssvc(sock)
int sock;

async_daemon()

DESCRIPTION

Nfssvc starts an NFS daemon listening on socket *sock*. The socket must be AF_INET, and SOCK_DGRAM (protocol UDP/IP). The system call will return only if the process is killed.

Async_daemon implements the NFS daemon that handles asynchronous I/O for an NFS client. The system call never returns.

BUGS

These two system calls allow kernel processes to have user context.

SEE ALSO

mountd(1M)

NAME

ether_ntoa, ether_aton, ether_ntohost, ether_hostton, ether_line – ethernet address mapping operations

SYNOPSIS

```
#include <sys/types.h>
#include <sys/socket.h>
#include <net/if.h>
#include <netinet/in.h>
#include <netinet/if_ether.h>

char *
ether_ntoa(e)
        struct ether_addr *e;

struct ether_addr *
ether_aton(s)
        char *s;

ether_ntohost(hostname, e)
        char *hostname;
        struct ether_addr *e;

ether_hostton(hostname, e)
        char *hostname;
        struct ether_addr *e;

ether_line(l, e, hostname)
        char *l;
        struct ether_addr *e;
        char *hostname;
```

DESCRIPTION

These routines are useful for mapping 48 bit ethernet numbers to their ASCII representations or their corresponding host names, and vice versa.

The function *ether_ntoa* converts a 48 bit ethernet number pointed to by *e* to its standard ACSII representation; it returns a pointer to the ASCII string. The representation is of the form: "x:x:x:x:x:x" where *x* is a hexadecimal number between 0 and ff. The function *ether_aton* converts an ASCII string in the standard representation back to a 48 bit ethernet number; the function returns NULL if the string cannot be scanned successfully.

The function *ether_ntohost* maps an ethernet number (pointed to by *e*) to its associated hostname. The string pointed to by *hostname* must be long enough to hold the hostname and a null character. The function returns zero upon success and non-zero upon failure. Inversely, the function

ether_hostton maps a hostname string to its corresponding ethernet number; the function modifies the ethernet number pointed to by *e*. The function also returns zero upon success and non-zero upon failure.

The function *ether_line* scans a line (pointed to by *l*) and sets the hostname and the ethernet number (pointed to by *e*). The string pointed to by *hostname* must be long enough to hold the hostname and a null character. The function returns zero upon success and non-zero upon failure.

SEE ALSO

ethers(4)

NAME

exportent, getexportent, setexportent, addexportent, remexportent, endexportent, getexportopt – get exported file system information

SYNOPSIS

```
#include <stdio.h>
#include <exportent.h>
FILE *setexportent( )
struct exportent *getexportent(filep)
        FILE *filep;
int addexportent(filep, dirname, options)
        FILE *filep;
        char *dirname;
        char *options;
int remexportent(filep, dirname)
        FILE *filep;
        char *dirname;
char *getexportopt(xent, opt)
        struct exportent *xent;
        char *opt;
void endexportent(filep)
        FILE *filep;
```

DESCRIPTION

These routines access the exported filesystem information in */etc/xtab* .

setexportent opens the export information file and returns a file pointer to use with *getexportent*, *addexportent*, *remexportent*, and *endexportent*. *getexportent* reads the next line from *filep* and returns a pointer to an object with the following structure containing the broken-out fields of a line in the file, */etc/xtab*. The fields have meanings described in *exports*(4).

```
#define ACCESS_OPT   ''access''  /* machines that can mount fs */
#define ROOT_OPT     ''root''    /* machines with root access of fs */
#define RO_OPT       ''ro''      /* export read-only */
#define RW_OPT       ''rw''      /* export read-mostly */
#define ANON_OPT     ''anon''    /* uid for anonymous requests */
#define NOHIDE_OPT   ''nohide''  /* visible from upper-exported fs */
struct exportent {
     char *xent_dirname;         /* directory (or file) to export */
     char *xent_options;         /* options, as above */
};
```

addexportent adds the *exportent* to the end of the open file *filep*. It returns 0 if successful and −1 on failure. *remexportent* removes the indicated entry from the list. It also returns 0 on success and −1 on failure. *getexportopt* scans the *xent_options* field of the *exportent* structure for a substring that matches *opt*. It returns the string value of *opt*, or NULL if the option is not found.

endexportent closes the file.

NOTE

The **NOHIDE_OPT** option is specific to IRIX.

To compile and link a program that calls these routines, follow the procedures for section (3Y) routines as described in *intro*(3).

FILES

/etc/exports
/etc/xtab

SEE ALSO

exportfs(1M), exports(4).

DIAGNOSTICS

NULL pointer (0) returned on EOF or error.

BUGS

The returned *exportent* structure points to static information that is overwritten in each call.

NAME

setmntent, getmntent, addmntent, endmntent, hasmntopt – get file system descriptor file entry

SYNOPSIS

```
#include <stdio.h>
#include <mntent.h>

FILE *setmntent(filep, type)
char *filep;
char *type;

struct mntent *getmntent(filep)
FILE *filep;

int addmntent(filep, mnt)
FILE *filep;
struct mntent *mnt;

char *hasmntopt(mnt, opt)
struct mntent *mnt;
char *opt;

int endmntent(filep)
FILE *filep;
```

DESCRIPTION

These routines replace the *getfsent* routines for accessing the file system description file */etc/fstab*. They are also used to access the mounted file system description file */etc/mtab*.

Setmntent opens a file system description file and returns a file pointer which can then be used with *getmntent*, *addmntent*, or *endmntent*. The *type* argument is the same as in *fopen*(3S). *Getmntent* reads the next line from *filep* and returns a pointer to an object with the following structure containing the broken-out fields of a line in the filesystem description file, *<mntent.h>*. The fields have meanings described in *fstab*(4).

```
struct mntent {
        char    *mnt_fsname;    /* file system name */
        char    *mnt_dir;       /* file system path prefix */
        char    *mnt_type;      /* dbg, efs, nfs */
        char    *mnt_opts;      /* ro, hide, etc. */
        int     mnt_freq;       /* dump frequency, in days */
        int     mnt_passno;     /* pass number on parallel fsck */
};
```

Addmntent adds the *mntent* structure *mnt* to the end of the open file *filep*. Note that *filep* has to be opened for writing if this is to work. *Hasmntopt* scans the *mnt_opts* field of the *mntent* structure *mnt* for a substring that matches *opt*. It returns the address of the substring if a match is found, 0 otherwise. *Endmntent* closes the file.

NOTE

To compile and link a program that calls these routines, follow the procedures for section (3Y) routines as described in *intro*(3).

FILES

/etc/fstab
/etc/mtab

SEE ALSO

fstab(4)

DIAGNOSTICS

Null pointer (0) returned on EOF or error.

BUGS

The returned *mntent* structure points to static information that is overwritten in each call.

NAME

getnetgrent, setnetgrent, endnetgrent, innetgr – get network group entry

SYNOPSIS

innetgr(netgroup, machine, user, domain)
char *netgroup, *machine, *user, *domain;

setnetgrent(netgroup)
char *netgroup

endnetgrent()

getnetgrent(machinep, userp, domainp)
char **machinep, **userp, **domainp;

DESCRIPTION

Innetgr returns 1 or 0, depending on whether *netgroup* contains the machine, user, or domain triple as a member. Any of the three strings machine, user, or domain can be NULL, in which case it signifies a wild card.

Getnetgrent returns the next member of a network group. After the call, machinep will contain a pointer to a string containing the name of the machine part of the network group member, and similarly for userp and domainp. If any of machinep, userp or domainp is returned as a NULL pointer, it signifies a wild card. *Getnetgrent* will *malloc*(3) space for the name. This space is released when a *endnetgrent* call is made. *Getnetgrent* returns 1 if it succeeding in obtaining another member of the network group, 0 if it has reached the end of the group.

Setnetgrent establishes the network group from which *getnetgrent* will obtain members, and also restarts calls to *getnetgrent* from the beginning of the list. If the previous *setnetgrent* call was to a different network group and there has been no intervening call to *endnetgrent*, an *endnetgrent* call is implied. *Endnetgrent* frees the space allocated during the *getnetgrent* calls.

FILES

/usr/etc/netgroup

NAME

mount – keep track of remotely mounted filesystems

SYNOPSIS

#include <rpcsvc/mount.h>

RPC INFORMATION

Program number:

```
MOUNTPROG
```

XDR routines:

```
xdr_exportbody(xdrs, ex)
        XDR *xdrs;
        struct exports *ex;
xdr_exports(xdrs, ex);
        XDR *xdrs;
        struct exports **ex;
xdr_nexportbody(xdrs, nex)
        XDR *xdrs;
        struct nexports *nex;
xdr_nexports(xdrs, nex);
        XDR *xdrs;
        struct nexports **nex;
xdr_fhandle(xdrs, fh);
        XDR *xdrs;
        fhandle_t *fp;
xdr_fhstatus(xdrs, fhs);
        XDR *xdrs;
        struct fhstatus *fhs;
xdr_groups(xdrs, gr);
        XDR *xdrs;
        struct groups *gr;
xdr_mountbody(xdrs, ml)
        XDR *xdrs;
        struct mountlist *ml;
xdr_mountlist(xdrs, ml);
        XDR *xdrs;
        struct mountlist **ml;
xdr_path(xdrs, path);
        XDR *xdrs;
        char **path;
```

Procedures:

```
MOUNTPROC_MNT
        Argument of xdr_path, returns fhstatus.
        Requires unix authentication.
MOUNTPROC_DUMP
        No arguments, returns struct mountlist.
MOUNTPROC_UMNT
        Argument of xdr_path, no results.
        Requires unix authentication.
MOUNTPROC_UMNTALL
        No arguments, no results.
        Requires unix authentication.
        Unmounts all remote mounts of sender.
MOUNTPROC_EXPORT
MOUNTPROC_EXPORTALL
        No arguments, returns struct exports if program version
        is MOUNTPROG_ORIG, struct nexports if program version
        is MOUNTPROG_NEWSGI.
```

Versions:

```
MOUNTVERS_ORIG
        Universal program version.
MOUNTVERS_NEWSGI
        SGI enhanced version for complete exports inquiry.
```

Structures:

```
struct mountlist {              /* what is mounted */
        char *ml_name;
        char *ml_path;
        struct mountlist *ml_nxt;
};
struct fhstatus {
        int fhs_status;
        fhandle_t fhs_fh;
};
/*
 * List of exported directories.
 * An export entry with ex_groups NULL indicates an entry
 * which is exported to the world.
 */
struct exports {
        dev_t           ex_dev;         /* dev of directory */
        char            *ex_name;       /* name of directory */
        struct groups   *ex_groups;     /* groups given access */
```

```
        struct exports *ex_next;
};
struct groups {
        char            *g_name;
        struct groups   *g_next;
};
/*
 * List of exported directories with all options documented
 * in exports(4).
 */
struct nexports {
        char            *nex_name;
        struct groups   *nex_access;
        struct groups   *nex_root;
        bool_t          nex_ro;
        struct groups   *nex_rw;
        int             nex_anon;
        bool_t          nex_nohide;
        struct nexports*nex_next;
};
```

SEE ALSO

exportfs(1M), mount(1M), showmount(1M), mountd(1M), exports(4).

NAME

rnusers, rusers – return information about users on remote machines

SYNOPSIS

```
#include <rpcsvc/rusers.h>

rnusers(host)
        char *host

rusers(host, up)
        char *host
        struct utmpidlearr *up;
```

DESCRIPTION

Rnusers returns the number of users logged on to *host* (–1 if it cannot determine that number). *Rusers* fills the *utmpidlearr* structure with data about *host*, and returns 0 if successful. The relevant structures are:

```
struct utmparr {                        /* RUSERSVERS_ORIG */
        struct utmp **uta_arr;
        int uta_cnt
};

struct utmpidle {
        struct utmp ui_utmp;
        unsigned ui_idle;
};

struct utmpidlearr {                    /* RUSERSVERS_IDLE */
        struct utmpidle **uia_arr;
        int uia_cnt
};
```

RPC INFO

program number:

RUSERSPROG

xdr routines:

```
int xdr_utmp(xdrs, up)
        XDR *xdrs;
        struct utmp *up;
int xdr_utmpidle(xdrs, ui);
        XDR *xdrs;
        struct utmpidle *ui;
int xdr_utmpptr(xdrs, up);
        XDR *xdrs;
        struct utmp **up;
int xdr_utmpidleptr(xdrs, up);
```

```
        XDR *xdrs;
        struct utmpidle **up;
int xdr_utmparr(xdrs, up);
        XDR *xdrs;
        struct utmparr *up;
int xdr_utmpidlearr(xdrs, up);
        XDR *xdrs;
        struct utmpidlearr *up;
```

procs:

RUSERSPROC_NUM
No arguments, returns number of users as an *unsigned long*.

RUSERSPROC_NAMES
No arguments, returns *utmparr* or *utmpidlearr*, depending on version number.

RUSERSPROC_ALLNAMES
No arguments, returns *utmparr* or *utmpidlearr*, depending on version number.
Returns listing even for *utmp* entries satisfying *nonuser()* in *utmp.h*.

versions:

RUSERSVERS_ORIG
RUSERSVERS_IDLE

NAME

rwall – write to specified remote machines

SYNOPSIS

#include <rpcsvc/rwall.h>

rwall(host, msg);
char *host, *msg;

DESCRIPTION

Rwall causes *host* to print the string *msg* to all its users. It returns 0 if successful.

RPC INFO

program number:
WALLPROG

procs:
WALLPROC_WALL
Takes string as argument (wrapstring), returns no arguments. Executes *wall* on remote host with string.

versions:
RSTATVERS_ORIG

SEE ALSO

rwall(1), shutdown(1m), rwalld(1m)

NAME

yp_update – changes yp information

SYNOPSIS

#include <rpcsvc/ypclnt.h>

yp_update(domain, map, ypop, key, keylen, data, datalen)
char *domain;
char *map;
unsigned ypop
char *key;
int keylen;
char *data;
int datalen;

DESCRIPTION

yp_update() is used to make changes to the YP database. The syntax is the same as that of **yp_match()** except for the extra parameter *ypop* which may take on one of four values. If it is **YPOP_CHANGE,** then the data associated with the key will be changed to the new value. If the key is not found in the database, then **yp_update()** will return **YPERR_KEY**. If *ypop* has the value **YPOP_INSERT,** then the key-value pair will be inserted into the database. The error **YPERR_KEY** is returned if the key already exists in the database. To store an item into the database without concern for whether it exists already or not, pass *ypop* as **YPOP_STORE** and no error will be returned if the key already or does not exist. To delete an entry, the value of *ypop* should be **YPOP_DELETE.**

SEE ALSO

ypupdated(1M), updaters(1M)

NAME

ypclnt yp_get_default_domain yp_bind yp_unbind yp_match yp_first yp_next yp_all yp_order yp_master yperr_string ypprot_err – yellow pages client interface

SYNOPSIS

```
#include <rpcsvc/ypclnt.h>

yp_bind(indomain);
char *indomain;

void yp_unbind(indomain)
char *indomain;

yp_get_default_domain(outdomain);
char **outdomain;

yp_match(indomain, inmap, inkey, inkeylen, outval, outvallen)
char *indomain;
char *inmap;
char *inkey;
int inkeylen;
char **outval;
int *outvallen;

yp_first(indomain, inmap, outkey, outkeylen, outval, outvallen)
char *indomain;
char *inmap;
char **outkey;
int *outkeylen;
char **outval;
int *outvallen;

yp_next(indomain, inmap, inkey, inkeylen, outkey,
        outkeylen, outval, outvallen);
char *indomain;
char *inmap;
char *inkey;
int inkeylen;
char **outkey;
int *outkeylen;
char **outval;
int *outvallen;

yp_all(indomain, inmap, incallback);
char *indomain;
char *inmap;
struct ypall_callback incallback;
```

yp_order(indomain, inmap, outorder);
char *indomain;
char *inmap;
int *outorder;

yp_master(indomain, inmap, outname);
char *indomain;
char *inmap;
char **outname;

char *yperr_string(incode)
int incode;

ypprot_err(incode)
unsigned int incode;

DESCRIPTION

This package of functions provides an interface to the Yellow Pages (YP) network lookup service. The package can be loaded from the library, */usr/lib/libsun.a*. Refer to *ypfiles*(4) and *ypserv*(1M) for an overview of the yellow pages, including the definitions of *map* and *domain* , and a description of the various servers, databases, and commands that comprise the YP.

All input parameters names begin with **in**. Output parameters begin with **out**. Output parameters of type *char* ** should be addresses of uninitialized character pointers. Memory is allocated by the YP client package using *malloc*(3), and may be freed if the user code has no continuing need for it. For each *outkey* and *outval*, two extra bytes of memory are allocated at the end that contain NEWLINE and NULL, respectively, but these two bytes are not reflected in *outkeylen* or *outvallen* . *indomain* and *inmap* strings must be non-null and null-terminated. String parameters which are accompanied by a count parameter may not be null, but may point to null strings, with the count parameter indicating this. Counted strings need not be null-terminated.

All functions in this package of type **int** return 0 if they succeed, and a failure code (YPERR_*xxxx*) otherwise. Failure codes are described under DIAGNOSTICS below.

The YP lookup calls require a map name and a domain name, at minimum. It is assumed that the client process knows the name of the map of interest. Client processes should fetch the node's default domain by calling **yp_get_default_domain()** , and use the returned *outdomain* as the *indomain* parameter to successive YP calls.

To use the YP services, the client process must be "bound" to a YP server that serves the appropriate domain using *yp_bind*. Binding need not be done explicitly by user code; this is done automatically whenever a YP lookup function is called. *yp_bind* can be called directly for processes that make use of a backup strategy (e.g., a local file) in cases when YP services are not available.

Each binding allocates (uses up) one client process socket descriptor; each bound domain costs one socket descriptor. However, multiple requests to the same domain use that same descriptor. *yp_unbind()* is available at the client interface for processes that explicitly manage their socket descriptors while accessing multiple domains. The call to *yp_unbind()* make the domain *unbound*, and free all per-process and per-node resources used to bind it.

If an RPC failure results upon use of a binding, that domain will be unbound automatically. At that point, the ypclnt layer will retry forever or until the operation succeeds, provided that *ypbind* is running, and either

a) the client process can't bind a server for the proper domain, or

b) RPC requests to the server fail.

If an error is not RPC-related, or if *ypbind* is not running, or if a bound *ypserv* process returns any answer (success or failure), the ypclnt layer will return control to the user code, either with an error code, or a success code and any results.

yp_match returns the value associated with a passed key. This key must be exact; no pattern matching is available.

yp_first returns the first key-value pair from the named map in the named domain.

yp_next() returns the next key-value pair in a named map. The *inkey* parameter should be the *outkey* returned from an initial call to *yp_first()* (to get the second key-value pair) or the one returned from the nth call to *yp_next()* (to get the nth + second key-value pair).

The concept of *first* (and, for that matter, of *next)* is particular to the structure of the YP map being processed; there is no relation in retrieval order to either the lexical order within any original (non-YP) data base, or to any obvious numerical sorting order on the keys, values, or key-value pairs.

The only ordering guarantee made is that if the *yp_first()* function is called on a particular map, and then the *yp_next()* function is repeatedly called on the same map at the same server until the call fails with a reason of YPERR_NOMORE, every entry in the data base will be seen exactly once. Further, if the same sequence of operations is performed on the same map at the same server, the entries will be seen in the same order.

Under conditions of heavy server load or server failure, it is possible for the domain to become unbound, then bound once again (perhaps to a different server) while a client is running. This can cause a break in one of the enumeration rules; specific entries may be seen twice by the client, or not at all. This approach protects the client from error messages that would otherwise be returned in the midst of the enumeration. The next paragraph describes a better solution to enumerating all entries in a map.

yp_all provides a way to transfer an entire map from server to client in a single request using TCP (rather than UDP as with other functions in this package). The entire transaction takes place as a single RPC request and response. You can use *yp_all* just like any other YP procedure, identify the map in the normal manner, and supply the name of a function which will be called to process each key-value pair within the map. You return from the call to *yp_all* only when the transaction is completed (successfully or unsuccessfully), or your *"foreach"* function decides that it doesn't want to see any more key-value pairs.

The third parameter to *yp_all* is

```
struct ypall_callback *incallback {
        int (*foreach)();
        char *data;
};
```

The function *foreach* is called

```
foreach(instatus, inkey, inkeylen, inval, invallen, indata);
int instatus;
char *inkey;
int inkeylen;
char *inval;
int invalllen;
char *indata;
```

The *instatus* parameter will hold one of the return status values defined in *<rpcsvc/yp_prot.h>* — either *YP_TRUE* or an error code. (See *ypprot_err*, below, for a function which converts a YP protocol error code to a ypclnt layer error code.)

The key and value parameters are somewhat different than defined in the synopsis section above. First, the memory pointed to by the *inkey* and *inval* parameters is private to the *yp_all* function, and is overwritten with the arrival of each new key-value pair. It is the responsibility of the *foreach* function to do something useful with the contents of that memory, but it does not own the memory itself. Key and value objects presented to the *foreach* function look exactly as they do in the server's map — if they were not newline-terminated or null-terminated in the map, they won't be here

either.

The *indata* parameter is the contents of the *incallback->data* element passed to *yp_all* . The *data* element of the callback structure may be used to share state information between the *foreach* function and the mainline code. Its use is optional, and no part of the YP client package inspects its contents — cast it to something useful, or ignore it as you see fit.

The *foreach* function is a Boolean. It should return zero to indicate that it wants to be called again for further received key-value pairs, or non-zero to stop the flow of key-value pairs. If *foreach* returns a non-zero value, it is not called again; the functional value of *yp_all* is then 0.

yp_order returns the order number for a map.

yp_master returns the machine name of the master YP server for a map.

yperr_string returns a pointer to an error message string that is null-terminated but contains no period or newline.

ypprot_err takes a YP protocol error code as input, and returns a ypclnt layer error code, which may be used in turn as an input to *yperr_string* .

FILES

/usr/include/rpcsvc/ypclnt.h
/usr/include/rpcsvc/yp_prot.h

SEE ALSO

ypfiles(4), ypserv(1M),

DIAGNOSTICS

All integer functions return 0 if the requested operation is successful, or one of the following errors if the operation fails.

```
#define YPERR_BADARGS   1   /* args to function are bad */
#define YPERR_RPC       2   /* RPC failure - domain has been unbound */
#define YPERR_DOMAIN    3   /* can't bind to server on this domain */
#define YPERR_MAP       4   /* no such map in server's domain */
#define YPERR_KEY       5   /* no such key in map */
#define YPERR_YPERR     6   /* internal yp server or client error */
#define YPERR_RESRC     7   /* resource allocation failure */
#define YPERR_NOMORE    8   /* no more records in map database */
#define YPERR_PMAP      9   /* can't communicate with portmapper */
#define YPERR_YPBIND    10  /* can't communicate with ypbind */
#define YPERR_YPSERV    11  /* can't communicate with ypserv */
#define YPERR_NODOM     12  /* local domain name not set */
```

NAME

yppasswd – update user password in Yellow Pages

SYNOPSIS

#include <rpcsvc/yppasswd.h>

yppasswd(oldpass, newpw)
char *oldpass
struct passwd *newpw;

DESCRIPTION

If *oldpass* is indeed the old user password, this routine replaces the password entry with *newpw*. It returns 0 if successful.

RPC INFO

program number:
YPPASSWDPROG

xdr routines:

```
xdr_ppasswd(xdrs, yp)
        XDR *xdrs;
        struct yppasswd *yp;
xdr_yppasswd(xdrs, pw)
        XDR *xdrs;
        struct passwd *pw;
```

procs:

YPPASSWDPROC_UPDATE
Takes *struct yppasswd* as argument, returns integer.
Same behavior as *yppasswd()* wrapper.
Uses UNIX authentication.

versions:
YPPASSWDVERS_ORIG

structures:

```
struct yppasswd {
        char *oldpass;     /* old (unencrypted) password */
        struct passwd newpw;       /* new pw structure */
};
```

SEE ALSO

yppasswd(1), rpc.passwd(1M)

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 or SPACE 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

bootparamd(1m)

NAME

ethers – ethernet address to hostname database

DESCRIPTION

The *ethers* file contains information regarding the known (48 bit) ethernet addresses of hosts on the internet. For each host on an ethernet, a single line should be present with the following information:

ethernet address
official host name

Items are separated by any number of blanks and/or tabs. A '#' indicates the beginning of a comment extending to the end of line.

The standard form for ethernet addresses is "x:x:x:x:x:x" where *x* is a hexadecimal number between 0 and ff, representing one byte. The address bytes are always in network order. Host names may contain any printable character other than a space, tab, newline, or comment character. It is intended that host names in the *ethers* file correspond to the host names in the *hosts*(4) file.

The *ether_line*() routine from the ethernet address manipulation library, *ethers*(3Y) may be used to scan lines of the *ethers* file.

FILES

/etc/ethers

SEE ALSO

ethers(3Y), hosts(4)

NAME

exports – list of NFS filesystems being exported

SYNOPSIS

/etc/exports

DESCRIPTION

The file */etc/exports* describes the filesystems which are being exported to NFS clients. It is created by the system administrator using a text editor and processed by *exportfs*(1M) at system startup and by the *mount* request daemon, *mountd*(1M), each time a mount request is received. *Exportfs* should be re-executed after making changes to the file.

The file consists of a list of filesystems, the *netgroup*(4) or machine names allowed to remote mount each filesystem, and possibly a list of options. The filesystem names are left justified and followed by a list of names separated by white space. The names will be looked up in */etc/netgroup* and then in */etc/hosts*. A hyphen indicates the start of the options list. Multiple options are separated by commas. The default options are **rw,hide,anon=nobody**.

ro Export the directory read-only. If not specified, the directory is exported read-write.

rw=*hostname*[**:***hostname*]...
Export the directory read-mostly. Read-mostly means exported read-only to most machines, but read-write to those specified. If no hosts are specified, the directory is exported read-write to all.

anon=*uid*
If a request comes from an unknown user, use *uid* as the effective user ID. *uid* may be either a name or an integer user-id from */etc/passwd*. The default value for this option is ''nobody'' (uid –2). Setting the value of ''anon'' to –1 disables anonymous access. Note: root users (uid 0) are always considered "unknown" by the NFS server, unless they are included in the "root" option below.

root=*hostname*[**:***hostname*]...
Give root access only to the root users from a specified *hostname*. The default is for no hosts to be granted root access.

access=*client*[**:***client*]...
Give mount access to each *client* listed. A *client* can either be a hostname, or a netgroup (see *netgroup*(4)). Each *client* in the list is first checked for in the */etc/netgroup* database, and then the */etc/hosts* database. The default value allows any machine to mount the given directory.

hide Prevents a client who mounts this entry's parent filesystem from accessing files in this filesystem. Instead, clients who mount a filesystem containing a hidden filesystem access the directory on which the hidden child is mounted, not the child filesystem's root directory.

nohide Allows a client who mounts this entry's parent filesystem to access files in this filesystem.

wsync Causes all writes to this file system to be performed synchronously to the disk. With this option, the server waits until the data is safely written to the disk before sending a positive response to the client. Without this option, the server performs delayed-writes (i.e., responds to the client then writes the data at its convenience or when a *sync*(2) is executed). Delaying writes provides a great performance boost, but also introduces the risk of losing data should the server crash before the data is written to the disk. Use the wsync option if this risk is unacceptable.

A filesystem name which is not followed by a name list is exported to everyone. A "#" anywhere in the file indicates a comment extending to the end of the line on which it appears. Lines beginning with white space are continuation lines.

EXAMPLES

```
/usr/local                      # export to the world
/usr      clients               # export to my clients
/usr2     bonnie clyde          # export to only these machines
/usr3     -anon=guest           # map client root & anonymous to guest
/         -ro                   # export the root and usr filesystems
/usr      -ro,nohide            # export all local filesystems read-only
```

Exporting all your machine's local filesystems requires enumerating all local mount points, and using "nohide" for each root filesystem:

```
/         -ro
/usr      -ro,nohide
/d        -ro,nohide
```

NOTE

The **rootid** option is a backward-compatible IRIX synonym for **anon**. The **hide**, **nohide** and **wsync** options are specific to IRIX.

FILES

/etc/exports

SEE ALSO

exportfs(1M), mountd(1M), netgroup(4)

NAME

hosts.equiv – list of trusted hosts

DESCRIPTION

The */etc/hosts.equiv* file contains a list of trusted hosts. When an *rcp*(1C), *rdist*(1C), *rlogin*(1C) or *rsh*(1C) request from such a host is made, and the initiator of the request is in */etc/passwd*, then, no further validity checking is done. That is, *rlogin* does not prompt for a password, and *rsh* completes successfully. So a remote user is "equivalenced" to a local user with the same user name when the remote user is in *hosts.equiv*.

The format of *hosts.equiv* is a list of names, as in this example:

```
host1
host2
+@group1
-@group2
```

A line consisting of a simple host name means that anyone logging in from that host is trusted. A line consisting of *+@group* means that all hosts in that network group (see *netgroup*(4)) are trusted. A line consisting of *–@group* means that hosts in that group are not trusted. Programs scan *hosts.equiv* linearly, and stop at the first hit (either positive for hostname and +@ entries, or negative for –@ entries). A line consisting of a single + means that everyone is trusted.

The *.rhosts* file has the same format as *hosts.equiv*. When user *XXX* executes *rcp*, *rdist*, *rlogin*, or *rsh*, the *.rhosts* file from *XXX*'s home directory is conceptually concatenated onto the end of *hosts.equiv* for permission checking. However, –@ entries are not sticky. If a user is excluded by a minus entry from *hosts.equiv* but included in *.rhosts*, then that user is considered trusted. In the special case when the user is root, then only the */.rhosts* file is checked.

It is also possible to have two names (separated by white space) on a line of these files. In this case, if the remote host is equivalenced by the first name, then the user named by the second name is allowed to log in as anyone, that is, specify any name to the **–l** flag (provided that name is in the */etc/passwd* file, of course). Thus the entry

```
gotham batman
```

in */etc/hosts.equiv* allows *batman* to log in from gotham as anyone. The usual usage would be to put this entry in the *.rhosts* file in the home directory for *robin* .

Then *batman* may log in as *robin* when coming from gotham. The second entry may be a netgroup, thus

+@group1 +@group2

allows any user in *group2* coming from a host in *group1* to log in as anyone.

FILES

/etc/hosts.equiv
~/.rhosts

WARNING

The references to network groups (+@ and –@ entries) in *hosts.equiv* and *.rhosts* are only supported when the *netgroup* file is supplied by the Yellow Pages.

SEE ALSO

rcp(1C), rdist(1C), rlogin(1C), rsh(1C), ruserok(3N), netgroup(4), rhosts(4)

NAME

netgroup – list of network groups

DESCRIPTION

Netgroup defines network wide groups, used for permission checking when doing remote mounts, remote logins, and remote shells. For remote mounts, the information in *netgroup* is used to classify machines; for remote logins and remote shells, it is used to classify users. Each line of the *netgroup* file defines a group and has the format

groupname member1 member2 ....

where member*i* is either another group name, or a triple:

(hostname, username, domainname)

Any of three fields can be empty, in which case it signifies a wild card. Thus

universal (,,)

defines a group to which everyone belongs. Field names that begin with something other than a letter, digit or underscore (such as "-") work in precisely the opposite fashion. For example, consider the following entries:

justmachines	(analytica,-,sun)
justpeople	(-,babbage,sun)

The machine *analytica* belongs to the group *justmachines* in the domain *sun*, but no users belong to it. Similarly, the user *babbage* belongs to the group *justpeople* in the domain *sun*, but no machines belong to it.

Network groups are contained in the yellow pages, and are accessed through these files:

/etc/yp/*domainname*/netgroup.dir
/etc/yp/*domainname*/netgroup.pag
/etc/yp/*domainname*/netgroup.byuser.dir
/etc/yp/*domainname*/netgroup.byuser.pag
/etc/yp/*domainname*/netgroup.byhost.dir
/etc/yp/*domainname*/netgroup.byhost.pag

These files can be created from */etc/netgroup* using *makedbm*(1M).

FILES

/etc/netgroup
/etc/yp/*domainname*/netgroup.dir
/etc/yp/*domainname*/netgroup.pag
/etc/yp/*domainname*/netgroup.byuser.dir
/etc/yp/*domainname*/netgroup.byuser.pag

/etc/yp/*domainname*/netgroup.byhost.dir
/etc/yp/*domainname*/netgroup.byhost.pag

SEE ALSO

getnetgrent(3), makedbm(1M), ypserv(1M)

NAME

rmtab – remotely mounted file system table

DESCRIPTION

Rmtab resides in the directory */etc* and contains a record of all clients that have done remote mounts of file systems from this machine. Whenever a remote *mount* is done, an entry is made in the *rmtab* file of the machine serving up that file system. *Umount* removes entries of a remotely mounted file system. *Umount –a* broadcasts to all servers, and informs them that they should remove all entries from *rmtab* created by the sender of the broadcast message (this is done automatically during system startup). The table is a series of lines of the form:

hostname:directory

This table is used only to preserve information between crashes, and is read only by *mountd*(1M) when it starts up. *Mountd* keeps an in-core table, which it uses to handle requests from programs like *showmount*(1) and *shutdown*(1M).

FILES

/etc/rmtab

SEE ALSO

showmount(1), mountd(1M), mount(1M), umount(1M), shutdown(1M)

BUGS

Although the *rmtab* table is close to the truth, it is not always 100% accurate.

NAME

sm, sm.bak, state – statd directories and file structures

SYNOPSIS

/usr/etc/statd.d/sm
/usr/etc/statd.d/sm.bak
/usr/etc/state

DESCRIPTION

/usr/etc/statd.d/sm and **/usr/etc/statd.d/sm.bak** are directories generated by **statd**. Each entry in **/usr/etc/statd.d/sm** represents the name of a machine to be monitored by **statd**. Each entry in **/usr/etc/statd.d/sm.bak** represents the name of a machine to be notified of **statd**'s recovery.

/usr/etc/statd.d/state is a file generated by **statd** to record its version number. This version number is incremented each time a crash or recovery takes place.

FILES

/usr/etc/statd.d/sm
/usr/etc/statd.d/sm.bak
/usr/etc/statd.d/state

SEE ALSO

lockd(1M), statd(1M)

NAME

updaters – configuration file for YP updating

SYNOPSIS

/usr/etc/yp/updaters

DESCRIPTION

The file */usr/etc/yp/updaters* is a makefile (see *make*(1)) which is used for updating YP databases. Each entry in the file is a make target for a particular YP database. For example, if there is a YP database named **passwd.byname** that can be updated, there should be a *make* target named **passwd.byname** in the *updaters* file with the command to update the file.

The information necessary to make the update is passed to the update command through standard input. The information passed is described below (all items are followed by a NEWLINE, except for 4 and 6)

- Network name of client wishing to make the update (a string)
- Kind of update (an integer)
- Number of bytes in key (an integer)
- Actual bytes of key
- Number of bytes in data (an integer)
- Actual bytes of data

After getting this information through standard input, the command to update the particular database should decide whether the user is allowed to make the change. If not, it should exit with the status YPERR_ACCESS. If the user is allowed to make the change, the command should make the change and exit with a status of zero. If there are any errors that may prevent the updater from making the change, it should exit with the status that matches a valid YP error code described in *<rpcsvc/ypclnt.h>*.

FILES

/usr/etc/yp/updaters

SEE ALSO

make(1), *ypupdated*(1M)

BUGS

Access control is insecure. Use only on a trusted network.

NAME

ypfiles – the Yellow Pages database and directory structure

DESCRIPTION

The yellow pages (YP) network lookup service uses a database of *dbm*(3B) files in the directory hierarchy at */usr/etc/yp*. A *dbm* database consists of two files, created by calls to the *dbm* library package. One has the filename extension *.pag* and the other has the filename extension *.dir*. For instance, the database named *hosts.byname*, is implemented by the pair of files *hosts.byname.pag* and *hosts.byname.dir*. A *dbm* database served by the YP is called a YP *map*. A YP *domain* is a named set of YP maps. Each YP domain is implemented as a subdirectory of */usr/etc/yp* containing the map. Any number of YP domains can exist. Each may contain any number of maps.

No maps are required by the YP lookup service itself, although they may be required for the normal operation of other parts of the system. There is no list of maps which YP serves - if the map exists in a given domain, and a client asks about it, the YP will serve it. For a map to be accessible consistently, it must exist on all YP servers that serve the domain. To provide data consistency between the replicated maps, entries to run *ypxfr* periodically exist in */usr/spool/cron/crontabs/root* on each server. More information on this topic is in *ypxfr*(1M)..

YP maps should contain two distinguished key-value pairs. The first is the key YP_LAST_MODIFIED, having as a value a ten-character ASCII order number. The order number should be the UNIX time in seconds when the map was built. The second key is YP_MASTER_NAME, with the name of the YP master server as a value. *makedbm*(1M) generates both key-value pairs automatically. A map that does not contain both key-value pairs can be served by the YP, but the *ypserv* process will not be able to return values for "Get order number" or "Get master name" requests. In addition, values of these two keys are used by *ypxfr* when it transfers a map from a master YP server to a slave. If *ypxfr* cannot figure out where to get the map, or if it is unable to determine whether the local copy is more recent than the copy at the master, you must set extra command line switches when you run it.

YP maps must be generated and modified only at the master server. They are copied to the slaves using *ypxfr*(1M) to avoid potential byte-ordering problems among YP servers running on machines with different architectures, and to minimize the amount of disk space required for the *dbm* files. The YP database can be initially set up for both masters and slaves by using *ypinit*(1M).

After the server databases are set up, it is probable that the contents of some maps will change. In general, some ASCII source version of the database exists on the master, and it is changed with a standard text editor. The update is incorporated into the YP map and is propagated from the master to the slaves by running */usr/etc/yp/ypmake*. *ypmake* executes the file */usr/etc/yp/Makefile* and logs its activity in */usr/etc/yp/ypmake.log*. */usr/etc/yp/Makefile* contains entries for all supplied maps; if you add a YP map, edit this file to support the new map. The makefile uses *makedbm* to generate the YP map on the master, and *yppush* to propagate the changed map to the slaves. *yppush* is a client of the map *ypservers* , which lists all the YP servers. For more information on this topic, see *yppush*(1M).

SEE ALSO

makedbm(1M), ypinit(1M), ypmake(1M), ypxfr(1M), yppush(1M), yppoll(1M), ypserv(1M), rpcinfo(1M), dbm(3B)