Am-utils (4.4BSD Automounter Utilities)

For version 6.2a3, 27 November 2006

Erez Zadok

(Originally by Jan-Simon Pendry and Nick Williams)

Preface

This manual documents the use of the 4.4BSD automounter tool suite, which includes *Amd*, *Amq*, *Hlfsd*, and other programs. This is primarily a reference manual. While no tutorial exists, there are examples available. See Chapter 11 [Examples], page 103.

This manual comes in two forms: the published form and the Info form. The Info form is for on-line perusal with the INFO program which is distributed along with GNU texinfo package (a version of which is available for GNU Emacs).[1] Both forms contain substantially the same text and are generated from a common source file, which is distributed with the *Am-utils* source.

[1] GNU packages can be found in `ftp://ftp.gnu.org/pub/gnu/`.

License

Source Distribution

The *Am-utils* home page is located in

`http://www.am-utils.org/`

You can get the latest distribution version of *Am-utils* from

`ftp://ftp.am-utils.org/pub/am-utils/am-utils.tar.gz`

Additional alpha, beta, and release distributions are available in

`ftp://ftp.am-utils.org/pub/am-utils/`.

Revision 5.2 was part of the 4.3BSD Reno distribution.

Revision 5.3bsdnet, a late alpha version of 5.3, was part of the BSD network version 2 distribution

Revision 6.0 was made independently by Erez Zadok at the Computer Science Department of Columbia University (`http://www.cs.columbia.edu/`), as part of his PhD thesis work (`http://www.fsl.cs.sunysb.edu/docs/zadok-thesis-proposal/`). Am-utils (especially version 6.1) continues to be developed and maintained at the Computer Science Department (`http://www.cs.sunysb.edu/`) of Stony Brook University (`http://www.stonybrook.edu/`), as a service to the user community.

See [History], page 11, for more details.

Getting Additional Information

Bug Reports

Before reporting a bug, see if it is a known one in the bugs (`http://www.am-utils.org/docs/am-utils/BUGS.t`
file.

If you find a problem and hopefully you can reproduce it, please describe it in detail and submit a bug report (`https://bugzilla.filesystems.org/`) via Bugzilla (`http://www.bugzilla.org/`). Alternatively, you can send your bug report to the "am-utils" list (see `http://www.am-utils.org/` under "Mailing Lists") quoting the details of the release and your configuration. These details can be obtained by running the command '`amd -v`'. It would greatly help if you could provide a reproducible procedure for detecting the bug you are reporting.

Providing working patches is highly encouraged. Every patch incorporated, however small, will get its author an honorable mention in the authors file (`http://www.am-utils.org/docs/am-utils/AUTHORS.txt`).

Mailing Lists

There are several mailing lists for people interested in keeping up-to-date with developments.

1. The users mailing list, '`am-utils`' is for
   - announcements of alpha and beta releases of am-utils
   - reporting of bugs and patches
   - discussions of new features for am-utils
   - implementation and porting issues

   To subscribe, visit `http://www.am-utils.org/` under "Mailing Lists." After subscribing, you can post a message to this list. To avoid as much spam as possible, only subscribers to this list may post to it.

   Subscribers of '`am-utils`' are most helpful if they have the time and resources to test new and development versions of amd, on as many different platforms as possible. They should also be prepared to learn and use the GNU Autoconf, Automake, and Libtool packages, as needed; and of course, become familiar with the complex code in the am-utils package. In other words, subscribers on this list should hopefully be able to contribute meaningfully to the development of amd.

   Note that this '`am-utils`' list used to be called '`amd-dev`' before January 1st, 2004. Please use the new name, '`am-utils`'.
2. The announcements mailing list, '`am-utils-announce`' is for announcements only (mostly new releases). To subscribe, visit `http://www.am-utils.org/` under "Mailing Lists." This list is read-only: only am-utils developers may post to it.
3. We distribute nightly CVS snapshots in `ftp://ftp.am-utils.org/pub/am-utils/snapshots/daily/`. If you like to get email notices of commits to the am-utils CVS repository, subscribe to the CVS logs mailing list, '`am-utils-cvs`' at `http://www.am-utils.org/` under "Mailing Lists."

4. The older list which was used to user discussions, '`amd-workers`', is defunct as of January 2004. (Its last address was `amd-workers AT majordomo.glue.umd.edu`.) Don't use '`amd-workers`': use the newer, more active '`am-utils`' list.
5. For completeness, there's a developers-only closed list called '`am-utils-developers`' (see `http://www.am-utils.org/` under "Mailing Lists").

Am-utils Book

Erez Zadok (`http://www.cs.sunysb.edu/~ezk`) wrote a book (`http://www.fsl.cs.sunysb.edu/docs/amd-b`
titled *Linux NFS and Automounter Administration*, ISBN 0-7821-2739-8, (Sybex, 2001). The book is full of details and examples that go beyond what this manual has. The book also covers NFS in great detail. Although the book is geared toward Linux users, it is general enough for any Unix administrator and contains specific sections for non-Linux systems.

Introduction

An *automounter* maintains a cache of mounted filesystems. Filesystems are mounted on demand when they are first referenced, and unmounted after a period of inactivity.

Amd may be used as a replacement for Sun's automounter. The choice of which filesystem to mount can be controlled dynamically with *selectors*. Selectors allow decisions of the form "hostname is *this*," or "architecture is not *that*." Selectors may be combined arbitrarily. *Amd* also supports a variety of filesystem types, including NFS, UFS and the novel *program* filesystem. The combination of selectors and multiple filesystem types allows identical configuration files to be used on all machines thus reducing the administrative overhead.

Amd ensures that it will not hang if a remote server goes down. Moreover, *Amd* can determine when a remote server has become inaccessible and then mount replacement filesystems as and when they become available.

Amd contains no proprietary source code and has been ported to numerous flavors of Unix.

History

The *Amd* package has been without an official maintainer since 1992. Several people have stepped in to maintain it unofficially. Most notable were the ‘upl’ (Unofficial Patch Level) releases of *Amd*, created by me (Erez Zadok (`http://www.cs.sunysb.edu/~ezk`)), and available from `ftp://ftp.am-utils.org/pub/amd/`. The last such unofficial release was ‘upl102’.

Through the process of patching and aging, it was becoming more and more apparent that *Amd* was in much need of revitalizing. Maintaining *Amd* had become a difficult task. I took it upon myself to cleanup the code, so that it would be easier to port to new platforms, add new features, keep up with the many new feature requests, and deal with the never ending stream of bug reports.

I have been working on such a release of *Amd* on and off since January of 1996. The new suite of tools is currently named `"`am-utils`"` (AutoMounter Utilities), in line with GNU naming conventions, befitting the contents of the package. In October of 1996 I had received enough offers to help me with this task that I decided to make a mailing list for this group of people. Around the same time, *Amd* had become a necessary part of my PhD thesis work, resulting in more work performed on am-utils.

Am-utils version 6.0 was numbered with a major new release number to distinguish it from the last official release of *Amd* (5.x). Many new features have been added such as a GNU `configure` system, NFS Version 3, a run-time configuration file (‘amd.conf’), many new ports, more scripts and programs, as well as numerous bug fixes. Another reason for the new major release number was to alert users of am-utils that user-visible interfaces may have changed. In order to make *Amd* work well for the next 10 years, and be easier to maintain, it was necessary to remove old or unused features, change various syntax files, etc. However, great care was taken to ensure the maximum possible backwards compatibility.

Am-utils version 6.1 has autofs support for Linux and Solaris 2.5`+` as *the* major new feature, in addition to several other minor new features. The autofs support is completely transparent to the end-user, aside from the fact that `/bin/pwd` now always returns the correct amd-ified path. The administrator can easily switch between NFS and autofs mounts by changing one parameter in `amd.conf`. Autofs support and maintenance was developed in conjunction with Ion Badulescu (`ionut AT badula.org`).

1 Overview

Amd maintains a cache of mounted filesystems. Filesystems are *demand-mounted* when they are first referenced, and unmounted after a period of inactivity. *Amd* may be used as a replacement for Sun's **automount**(8) program. It contains no proprietary source code and has been ported to numerous flavors of Unix. See Chapter 2 [Supported Platforms], page 17.

Amd was designed as the basis for experimenting with filesystem layout and management. Although *Amd* has many direct applications it is loaded with additional features which have little practical use. At some point the infrequently used components may be removed to streamline the production system.

Amd supports the notion of *replicated* filesystems by evaluating each member of a list of possible filesystem locations one by one. *Amd* checks that each cached mapping remains valid. Should a mapping be lost – such as happens when a fileserver goes down – *Amd* automatically selects a replacement should one be available.

1.1 Fundamentals

The fundamental concept behind *Amd* is the ability to separate the name used to refer to a file from the name used to refer to its physical storage location. This allows the same files to be accessed with the same name regardless of where in the network the name is used. This is very different from placing '`/n/hostname`' in front of the pathname since that includes location dependent information which may change if files are moved to another machine.

By placing the required mappings in a centrally administered database, filesystems can be re-organized without requiring changes to configuration files, shell scripts and so on.

1.2 Filesystems and Volumes

Amd views the world as a set of fileservers, each containing one or more filesystems where each filesystem contains one or more *volumes*. Here the term *volume* is used to refer to a coherent set of files such as a user's home directory or a TEX distribution.

In order to access the contents of a volume, *Amd* must be told in which filesystem the volume resides and which host owns the filesystem. By default the host is assumed to be local and the volume is assumed to be the entire filesystem. If a filesystem contains more than one volume, then a *sublink* is used to refer to the sub-directory within the filesystem where the volume can be found.

1.3 Volume Naming

Volume names are defined to be unique across the entire network. A volume name is the pathname to the volume's root as known by the users of that volume. Since this name uniquely identifies the volume contents, all volumes can be named and accessed from each host, subject to administrative controls.

Volumes may be replicated or duplicated. Replicated volumes contain identical copies of the same data and reside at two or more locations in the network. Each of the replicated volumes can be used interchangeably. Duplicated volumes each have the same name but contain different, though functionally identical, data. For example, '`/vol/tex`' might be the name of a TEX distribution which varied for each machine architecture.

Amd provides facilities to take advantage of both replicated and duplicated volumes. Configuration options allow a single set of configuration data to be shared across an entire network by taking advantage of replicated and duplicated volumes.

Amd can take advantage of replacement volumes by mounting them as required should an active fileserver become unavailable.

1.4 Volume Binding

Unix implements a namespace of hierarchically mounted filesystems. Two forms of binding between names and files are provided. A *hard link* completes the binding when the name is added to the filesystem. A *soft link* delays the binding until the name is accessed. An *automounter* adds a further form in which the binding of name to filesystem is delayed until the name is accessed.

The target volume, in its general form, is a tuple (host, filesystem, sublink) which can be used to name the physical location of any volume in the network.

When a target is referenced, *Amd* ignores the sublink element and determines whether the required filesystem is already mounted. This is done by computing the local mount point for the filesystem and checking for an existing filesystem mounted at the same place. If such a filesystem already exists then it is assumed to be functionally identical to the target filesystem. By default there is a one-to-one mapping between the pair (host, filesystem) and the local mount point so this assumption is valid.

1.5 Operational Principles

Amd operates by introducing new mount points into the namespace. These are called *automount* points. The kernel sees these automount points as NFS filesystems being served by *Amd*. Having attached itself to the namespace, *Amd* is now able to control the view the rest of the system has of those mount points. RPC calls are received from the kernel one at a time.

When a *lookup* call is received *Amd* checks whether the name is already known. If it is not, the required volume is mounted. A symbolic link pointing to the volume root is then returned. Once the symbolic link is returned, the kernel will send all other requests direct to the mounted filesystem.

If a volume is not yet mounted, *Amd* consults a configuration *mount-map* corresponding to the automount point. *Amd* then makes a runtime decision on what and where to mount a filesystem based on the information obtained from the map.

Amd does not implement all the NFS requests; only those relevant to name binding such as *lookup*, *readlink* and *readdir*. Some other calls are also implemented but most simply return an error code; for example *mkdir* always returns "read-only filesystem".

1.6 Mounting a Volume

Each automount point has a corresponding mount map. The mount map contains a list of key–value pairs. The key is the name of the volume to be mounted. The value is a list of locations describing where the filesystem is stored in the network. In the source for the map the value would look like

```
location1  location2  ...  locationN
```

Amd examines each location in turn. Each location may contain *selectors* which control whether *Amd* can use that location. For example, the location may be restricted to use by certain hosts. Those locations which cannot be used are ignored.

Amd attempts to mount the filesystem described by each remaining location until a mount succeeds or *Amd* can no longer proceed. The latter can occur in three ways:

- If none of the locations could be used, or if all of the locations caused an error, then the last error is returned.
- If a location could be used but was being mounted in the background then *Amd* marks that mount as being "in progress" and continues with the next request; no reply is sent to the kernel.
- Lastly, one or more of the mounts may have been *deferred*. A mount is deferred if extra information is required before the mount can proceed. When the information becomes available the mount will take place, but in the mean time no reply is sent to the kernel. If the mount is deferred, *Amd* continues to try any remaining locations.

Once a volume has been mounted, *Amd* establishes a *volume mapping* which is used to satisfy subsequent requests.

1.7 Automatic Unmounting

To avoid an ever increasing number of filesystem mounts, *Amd* removes volume mappings which have not been used recently. A time-to-live interval is associated with each mapping and when that expires the mapping is removed. When the last reference to a filesystem is removed, that filesystem is unmounted. If the unmount fails, for example the filesystem is still busy, the mapping is re-instated and its time-to-live interval is extended. The global default for this grace period is controlled by the `-w` command-line option (see Section 4.12 [-w Option], page 40) or the *amd.conf* parameter '`dismount_interval`' (see Section 6.5.8 [dismount_interval Parameter], page 59). It is also possible to set this value on a per-mount basis (see Section 3.3.4.4 [opts], page 31).

Filesystems can be forcefully timed out using the *Amq* command. See Chapter 7 [Run-time Administration], page 69. Note that on new enough systems that support forced unmounts, such as Linux, *Amd* can try to use the **umount2**(2) system call to force the unmount, if the regular **umount**(2) system call failed in a way that indicates that the mount point is hung or stale. See Section 6.5.11 [forced_unmounts Parameter], page 59.

1.8 Keep-alives

Use of some filesystem types requires the presence of a server on another machine. If a machine crashes then it is of no concern to processes on that machine that the filesystem is unavailable. However, to processes on a remote host using that machine as a fileserver this event is important. This situation is most widely recognized when an NFS server crashes and the behavior observed on client machines is that more and more processes hang. In order to provide the possibility of recovery, *Amd* implements a *keep-alive* interval timer for some filesystem types. Currently only NFS makes use of this service.

The basis of the NFS keep-alive implementation is the observation that most sites maintain replicated copies of common system data such as manual pages, most or all programs, system source code and so on. If one of those servers goes down it would be reasonable to mount one of the others as a replacement.

The first part of the process is to keep track of which fileservers are up and which are down. *Amd* does this by sending RPC requests to the servers' NFS `NullProc` and checking whether a reply is returned. While the server state is uncertain the requests are re-transmitted at three second intervals and if no reply is received after four attempts the server is marked down. If a reply is received the fileserver is marked up and stays in that state for 30 seconds at which time another NFS ping is sent. This interval is configurable and can even be turned off using the *ping* option. See Section 3.3.4.4 [opts Option], page 31.

Once a fileserver is marked down, requests continue to be sent every 30 seconds in order to determine when the fileserver comes back up. During this time any reference through *Amd* to the filesystems on that server fail with the error "Operation would block". If a replacement volume is available then it will be mounted, otherwise the error is returned to the user.

Although this action does not protect user files, which are unique on the network, or processes which do not access files via *Amd* or already have open files on the hung filesystem, it can prevent most new processes from hanging.

1.9 Non-blocking Operation

Since there is only one instance of *Amd* for each automount point, and usually only one instance on each machine, it is important that it is always available to service kernel calls. *Amd* goes to great lengths to ensure that it does not block in a system call. As a last resort *Amd* will fork before it attempts a system call that may block indefinitely, such as mounting an NFS filesystem. Other tasks such as obtaining filehandle information for an NFS filesystem, are done using a purpose built non-blocking RPC library which is integrated with *Amd*'s task scheduler. This library is also used to implement NFS keep-alives (see Section 1.8 [Keep-alives], page 15).

Whenever a mount is deferred or backgrounded, *Amd* must wait for it to complete before replying to the kernel. However, this would cause *Amd* to block waiting for a reply to be constructed. Rather than do this, *Amd* simply *drops* the call under the assumption that the kernel RPC mechanism will automatically retry the request.

2 Supported Platforms

Am-utils has been ported to a wide variety of machines and operating systems. *Am-utils*'s code works for little-endian and big-endian machines, as well as 32 bit and 64 bit architectures. Furthermore, when *Am-utils* ports to an Operating System on one architecture, it is generally readily portable to the same Operating System on all platforms on which it is available.

See the '`INSTALL`' in the distribution for more specific details on building and/or configuring for some systems.

3 Mount Maps

Amd has no built-in knowledge of machines or filesystems. External *mount-maps* are used to provide the required information. Specifically, *Amd* needs to know when and under what conditions it should mount filesystems.

The map entry corresponding to the requested name contains a list of possible locations from which to resolve the request. Each location specifies filesystem type, information required by that filesystem (for example the block special device in the case of UFS), and some information describing where to mount the filesystem (see Section 3.3.4.3 [fs Option], page 31). A location may also contain *selectors* (see Section 3.3.3 [Selectors], page 26).

3.1 Map Types

A mount-map provides the run-time configuration information to *Amd*. Maps can be implemented in many ways. Some of the forms supported by *Amd* are regular files, ndbm databases, NIS maps, the *Hesiod* name server, and even the password file.

A mount-map *name* is a sequence of characters. When an automount point is created a handle on the mount-map is obtained. For each map type configured, *Amd* attempts to reference the map of the appropriate type. If a map is found, *Amd* notes the type for future use and deletes the reference, for example closing any open file descriptors. The available maps are configured when *Amd* is built and can be displayed by running the command '`amd -v`'.

When using an *Amd* configuration file (see Chapter 6 [Amd Configuration File], page 55) and the keyword '`map_type`' (see Section 6.4.5 [map_type Parameter], page 56), you may force the map used to any type.

By default, *Amd* caches data in a mode dependent on the type of map. This is the same as specifying '`cache:=mapdefault`' and selects a suitable default cache mode depending on the map type. The individual defaults are described below. The *cache* option can be specified on automount points to alter the caching behavior (see Section 5.18 [Automount Filesystem], page 51).

The following map types have been implemented, though some are not available on all machines. Run the command '`amd -v`' to obtain a list of map types configured on your machine.

3.1.1 File maps

When *Amd* searches a file for a map entry it does a simple scan of the file and supports both comments and continuation lines.

Continuation lines are indicated by a backslash character ('\') as the last character of a line in the file. The backslash, newline character *and any leading white space on the following line* are discarded. A maximum line length of 2047 characters is enforced after continuation lines are read but before comments are stripped. Each line must end with a newline character; that is newlines are terminators, not separators. The following examples illustrate this:

```
key     valA   valB;   \
          valC
```

specifies *three* locations, and is identical to

```
key     valA   valB;   valC
```

However,

```
key     valA   valB;\
          valC
```

specifies only *two* locations, and is identical to

```
key     valA   valB;valC
```

After a complete line has been read from the file, including continuations, *Amd* determines whether there is a comment on the line. A comment begins with a hash ("`#`") character and continues to the end of the line. There is no way to escape or change the comment lead-in character.

Note that continuation lines and comment support *only* apply to file maps, or ndbm maps built with the `mk-amd-map` program.

When caching is enabled, file maps have a default cache mode of `all` (see Section 5.18 [Automount Filesystem], page 51).

3.1.2 ndbm maps

An ndbm map may be used as a fast access form of a file map. The program, `mk-amd-map`, converts a normal map file into an ndbm database. This program supports the same continuation and comment conventions that are provided for file maps. Note that ndbm format files may *not* be sharable across machine architectures. The notion of speed generally only applies to large maps; a small map, less than a single disk block, is almost certainly better implemented as a file map.

ndbm maps have a default cache mode of '`all`' (see Section 5.18 [Automount Filesystem], page 51).

3.1.3 NIS maps

When using NIS (formerly YP), an *Amd* map is implemented directly by the underlying NIS map. Comments and continuation lines are *not* supported in the automounter and must be stripped when constructing the NIS server's database.

NIS maps have a default cache mode of `all` (see Section 5.18 [Automount Filesystem], page 51).

The following rule illustrates what could be added to your NIS '`Makefile`', in this case causing the '`amd.home`' map to be rebuilt:

```
$(YPTSDIR)/amd.home.time: $(ETCDIR)/amd.home
        -@sed -e "s/#.*$$//" -e "/^$$/d" $(ETCDIR)/amd.home | \
          awk '{  \
             for (i = 1; i <= NF; i++) \
                 if (i == NF) { \
                 if (substr($$i, length($$i), 1) == "\\") \
                     printf("%s", substr($$i, 1, length($$i) - 1)); \
                 else \
                     printf("%s\n", $$i); \
                 } \
                 else \
                printf("%s ", $$i); \
            }' | \
        $(MAKEDBM) - $(YPDBDIR)/amd.home; \
        touch $(YPTSDIR)/amd.home.time; \
        echo "updated amd.home"; \
        if [ ! $(NOPUSH) ]; then \
            $(YPPUSH) amd.home; \
            echo "pushed amd.home"; \
        else \
            : ; \
        fi
```

Here `$(YPTSDIR)` contains the time stamp files, and `$(YPDBDIR)` contains the dbm format NIS files.

3.1.4 NIS+ maps

NIS+ maps do not support cache mode '`all`' and, when caching is enabled, have a default cache mode of '`inc`'.

XXX: FILL IN WITH AN EXAMPLE.

3.1.5 Hesiod maps

When the map name begins with the string '`hesiod.`' lookups are made using the *Hesiod* name server. The string following the dot is used as a name qualifier and is prepended with the key being located. The entire string is then resolved in the `automount` context, or the *amd.conf* parameter '`hesiod_base`' (see Section 6.5.14 [hesiod_base Parameter], page 60). For example, if the key is '`jsp`' and map name is '`hesiod.homes`' then *Hesiod* is asked to resolve '`jsp.homes.automount`'.

Hesiod maps do not support cache mode '`all`' and, when caching is enabled, have a default cache mode of '`inc`' (see Section 5.18 [Automount Filesystem], page 51).

The following is an example of a *Hesiod* map entry:

```
jsp.homes.automount HS TXT "rfs:=/home/charm;rhost:=charm;sublink:=jsp"
njw.homes.automount HS TXT "rfs:=/home/dylan/dk2;rhost:=dylan;sublink:=njw"
```

3.1.6 Password maps

The password map support is unlike the four previous map types. When the map name is the string '`/etc/passwd`' *Amd* can lookup a user name in the password file and re-arrange the home directory field to produce a usable map entry.

Amd assumes the home directory has the format '`/`*anydir*`/`*dom1*`/../`*domN*`/`*login*'. It breaks this string into a map entry where `${rfs}` has the value '`/`*anydir*`/`*domN*', `${rhost}` has the value '*domN*`.`...`.`*dom1*', and `${sublink}` has the value *login*.

Thus if the password file entry was

```
/home/achilles/jsp
```

the map entry used by *Amd* would be

```
rfs:=/home/achilles;rhost:=achilles;sublink:=jsp
```

Similarly, if the password file entry was

```
/home/cc/sugar/mjh
```

the map entry used by *Amd* would be

```
rfs:=/home/sugar;rhost:=sugar.cc;sublink:=mhj
```

3.1.7 Union maps

The union map support is provided specifically for use with the union filesystem, see Section 5.20 [Union Filesystem], page 53.

It is identified by the string '`union:`' which is followed by a colon separated list of directories. The directories are read in order, and the names of all entries are recorded in the map cache. Later directories take precedence over earlier ones. The union filesystem type then uses the map cache to determine the union of the names in all the directories.

3.1.8 LDAP maps

LDAP (Lightweight Directory Access Protocol) maps do not support cache mode '`all`' and, when caching is enabled, have a default cache mode of '`inc`'.

For example, an *Amd* map '`amd.home`' that looks as follows:

```
/defaults    opts:=rw,intr;type:=link

zing         -rhost:=shekel \
             host==shekel \
             host!=shekel;type:=nfs
```

when converted to LDAP (see Section 10.3 [amd2ldif], page 97), will result in the following LDAP database:

```
$ amd2ldif amd.home CUCS < amd.home
dn: cn=amdmap timestamp, CUCS
cn             : amdmap timestamp
objectClass    : amdmapTimestamp
amdmapTimestamp: 873071363

dn: cn=amdmap amd.home[/defaults], CUCS
cn          : amdmap amd.home[/defaults]
objectClass : amdmap
amdmapName  : amd.home
amdmapKey   : /defaults
amdmapValue : opts:=rw,intr;type:=link

dn: cn=amdmap amd.home[], CUCS
cn          : amdmap amd.home[]
objectClass : amdmap
amdmapName  : amd.home
amdmapKey   :
amdmapValue :

dn: cn=amdmap amd.home[zing], CUCS
cn          : amdmap amd.home[zing]
```

```
objectClass : amdmap
amdmapName  : amd.home
amdmapKey   : zing
amdmapValue : -rhost:=shekel host==shekel host!=shekel;type:=nfs
```

3.1.9 Executable maps

An executable map is a dynamic map in which the keys and values for the maps are generated on the fly by a program or script. The program is expected to take a single parameter argument which is the key to lookup. If the key is found, the program should print on stdout the key-value pair that were found; if the key was not found, nothing should be printed out. Below is an sample of such a map script:

```
#!/bin/sh
# executable map example
case "$1" in
    "/defaults" )
echo "/defaults   type:=nfs;rfs:=filer"
;;
    "a" )
echo "a   type:=nfs;fs:=/tmp"
;;
    "b" )
echo "b   type:=link;fs:=/usr/local"
;;
    * )  # no match, echo nothing
;;
esac
```

See Section 6.5.10 [exec_map_timeout Parameter], page 59.

3.2 How keys are looked up

The key is located in the map whose type was determined when the automount point was first created. In general the key is a pathname component. In some circumstances this may be modified by variable expansion (see Section 3.3.2 [Variable Expansion], page 25) and prefixing. If the automount point has a prefix, specified by the *pref* option, then that is prepended to the search key before the map is searched.

If the map cache is a '`regexp`' cache then the key is treated as an egrep-style regular expression, otherwise a normal string comparison is made.

If the key cannot be found then a *wildcard* match is attempted. *Amd* repeatedly strips the basename from the key, appends '`/*`' and attempts a lookup. Finally, *Amd* attempts to locate the special key '`*`'.

For example, the following sequence would be checked if '`home/dylan/dk2`' was being located:

```
home/dylan/dk2
home/dylan/*
home/*
*
```

At any point when a wildcard is found, *Amd* proceeds as if an exact match had been found and the value field is then used to resolve the mount request, otherwise an error code is propagated back to the kernel. (see Chapter 5 [Filesystem Types], page 45).

3.3 Location Format

The value field from the lookup provides the information required to mount a filesystem. The information is parsed according to the syntax shown below.

location-list:
: *location-selection*
: *location-list white-space* `||` *white-space location-selection*

location-selection:
: *location*
: *location-selection white-space location*

location:
: *location-info*
: `-`*location-info*
: `-`

location-info:
: *sel-or-opt*
: *location-info*`;`*sel-or-opt*
: `;`

sel-or-opt:
: *selection*
: *opt-ass*

selection:
: selector`==`*value*
: selector`!=`*value*

opt-ass:
: option`:=`*value*

white-space:
: space
: tab

Note that unquoted whitespace is not allowed in a location description. White space is only allowed, and is mandatory, where shown with non-terminal *white-space*.

A *location-selection* is a list of possible volumes with which to satisfy the request. Each *location-selection* is tried sequentially, until either one succeeds or all fail. This, by the way, is different from the historically documented behavior, which claimed (falsely, at least for last 3 years) that *Amd* would attempt to mount all *location-selection*s in parallel and the first one to succeed would be used.

*location-selection*s are optionally separated by the '`||`' operator. The effect of this operator is to prevent use of location-selections to its right if any of the location-selections on its left were selected, whether or not any of them were successfully mounted (see Section 3.3.3 [Selectors], page 26).

The location-selection, and singleton *location-list*, '`type:=ufs;dev:=/dev/xd1g`' would inform *Amd* to mount a UFS filesystem from the block special device '`/dev/xd1g`'.

The *sel-or-opt* component is either the name of an option required by a specific filesystem, or it is the name of a built-in, predefined selector such as the architecture type. The value may be quoted with double quotes '`"`', for example '`type:="ufs";dev:="/dev/xd1g"`'. These quotes are stripped when the value is parsed and there is no way to get a double quote into a value field. Double quotes are used to get white space into a value field, which is needed for the program filesystem (see Section 5.14 [Program Filesystem], page 49).

3.3.1 Map Defaults

A location beginning with a dash '`-`' is used to specify default values for subsequent locations. Any previously specified defaults in the location-list are discarded. The default string can be empty in which case no defaults apply.

The location '`-fs:=/mnt;opts:=ro`' would set the local mount point to '`/mnt`' and cause mounts to be read-only by default. Defaults specified this way are appended to, and so override, any global map defaults given with '`/defaults`').

3.3.2 Variable Expansion

To allow generic location specifications *Amd* does variable expansion on each location and also on some of the option strings. Any option or selector appearing in the form `$var` is replaced by the current value of that option or selector. For example, if the value of `${key}` was '`bin`', `${autodir}` was '`/a`' and `${fs}` was '`${autodir}/local/${key}`' then after expansion `${fs}` would have the value '`/a/local/bin`'. Any environment variable can be accessed in a similar way.

Two pathname operators are available when expanding a variable. If the variable name begins with '`/`' then only the last component of the pathname is substituted. For example, if `${path}` was '`/foo/bar`' then `${/path}` would be expanded to '`bar`'. Similarly, if the variable name ends with '`/`' then all but the last component of the pathname is substituted. In the previous example, `${path/}` would be expanded to '`/foo`'.

Two domain name operators are also provided. If the variable name begins with '`.`' then only the domain part of the name is substituted. For example, if `${rhost}` was '`swan.doc.ic.ac.uk`' then `${.rhost}` would be expanded to '`doc.ic.ac.uk`'. Similarly, if the variable name ends with '`.`' then only the host component is substituted. In the previous example, `${rhost.}` would be expanded to '`swan`'.

Variable expansion is a two phase process. Before a location is parsed, all references to selectors, *eg* `${path}`, are expanded. The location is then parsed, selections are evaluated and option assignments recorded. If there were no selections or they all succeeded the location is used and the values of the following options are expanded in the order given: *sublink*, *rfs*, *fs*, *opts*, *remopts*, *mount* and *unmount*.

Note that expansion of option values is done after *all* assignments have been completed and not in a purely left to right order as is done by the shell. This generally has the desired effect but care must be taken if one of the options references another, in which case the ordering can become significant.

There are two special cases concerning variable expansion:

1. before a map is consulted, any selectors in the name received from the kernel are expanded. For example, if the request from the kernel was for '`${arch}.bin`' and the machine architecture was '`vax`', the value given to `${key}` would be '`vax.bin`'.

2. the value of `${rhost}` is expanded and normalized before the other options are expanded. The normalization process strips any local sub-domain components. For example, if `${domain}` was '`Berkeley.EDU`' and `${rhost}` was initially '`snow.Berkeley.EDU`', after the normalization it would simply be '`snow`'. Hostname normalization is currently done in a *case-dependent* manner.

3.3.3 Selectors

Selectors are used to control the use of a location. It is possible to share a mount map between many machines in such a way that filesystem location, architecture and operating system differences are hidden from the users. A selector of the form '`arch==sun3;os==sunos4`' would only apply on Sun-3s running SunOS 4.x.

Selectors can be negated by using '`!=`' instead of '`==`'. For example to select a location on all non-Vax machines the selector '`arch!=vax`' would be used.

Selectors are evaluated left to right. If a selector fails then that location is ignored. Thus the selectors form a conjunction and the locations form a disjunction. If all the locations are ignored or otherwise fail then *Amd* uses the *error* filesystem (see Section 5.21 [Error Filesystem], page 54). This is equivalent to having a location '`type:=error`' at the end of each mount-map entry.

The default value of many of the selectors listed here can be overridden by an *Amd* command line switch or in an *Amd* configuration file. See Chapter 6 [Amd Configuration File], page 55.

The following selectors are currently implemented.

3.3.3.1 arch Selector Variable

The machine architecture which was automatically determined at compile time. The architecture type can be displayed by running the command '`amd -v`'. You can override this value also using the `-A` command line option. See Chapter 2 [Supported Platforms], page 17.

3.3.3.2 autodir Selector Variable

The default directory under which to mount filesystems. This may be changed by the `-a` command line option. See Section 3.3.4.3 [fs Option], page 31.

3.3.3.3 byte Selector Variable

The machine's byte ordering. This is either '`little`', indicating little-endian, or '`big`', indicating big-endian. One possible use is to share '`rwho`' databases (see Section 11.5 [rwho servers], page 106). Another is to share ndbm databases, however this use can be considered a courageous juggling act.

3.3.3.4 cluster Selector Variable

This is provided as a hook for the name of the local cluster. This can be used to decide which servers to use for copies of replicated filesystems. `${cluster}` defaults to the value of `${domain}` unless a different value is set with the `-C` command line option.

3.3.3.5 domain Selector Variable

The local domain name as specified by the `-d` command line option. See Section 3.3.3.7 [host Selector Variable], page 27.

3.3.3.6 dollar Selector Variable

This is a special variable, whose sole purpose is to produce a literal dollar sign in the value of another variable. For example, if you have a remote file system whose name is '`/disk$s`', you can mount it by setting the remote file system variable as follows:

```
rfs:=/disk${dollar}s
```

3.3.3.7 host Selector Variable

The local hostname as determined by **gethostname**(2). If no domain name was specified on the command line and the hostname contains a period '`.`' then the string before the period is used as the host name, and the string after the period is assigned to `${domain}`. For example, if the hostname is '`styx.doc.ic.ac.uk`' then `host` would be '`styx`' and `domain` would be '`doc.ic.ac.uk`'. `hostd` would be '`styx.doc.ic.ac.uk`'.

3.3.3.8 hostd Selector Variable

This resolves to the `${host}` and `${domain}` concatenated with a '`.`' inserted between them if required. If `${domain}` is an empty string then `${host}` and `${hostd}` will be identical.

3.3.3.9 karch Selector Variable

This is provided as a hook for the kernel architecture. This is used on SunOS 4 and SunOS 5, for example, to distinguish between different '`/usr/kvm`' volumes. `${karch}` defaults to the "machine" value gotten from **uname**(2). If the **uname**(2) system call is not available, the value of `${karch}` defaults to that of `${arch}`. Finally, a different value can be set with the `-k` command line option.

3.3.3.10 os Selector Variable

The operating system. Like the machine architecture, this is automatically determined at compile time. The operating system name can be displayed by running the command '`amd -v`'. See Chapter 2 [Supported Platforms], page 17.

3.3.3.11 osver Selector Variable

The operating system version. Like the machine architecture, this is automatically determined at compile time. The operating system name can be displayed by running the command '`amd -v`'. See Chapter 2 [Supported Platforms], page 17.

3.3.3.12 full_os Selector Variable

The full name of the operating system, including its version. This value is automatically determined at compile time. The full operating system name and version can be displayed by running the command '`amd -v`'. See Chapter 2 [Supported Platforms], page 17.

3.3.3.13 vendor Selector Variable

The name of the vendor of the operating system. This value is automatically determined at compile time. The name of the vendor can be displayed by running the command '`amd -v`'. See Chapter 2 [Supported Platforms], page 17.

The following selectors are also provided. Unlike the other selectors, they vary for each lookup. Note that when the name from the kernel is expanded prior to a map lookup, these selectors are all defined as empty strings.

3.3.3.14 key Selector Variable

The name being resolved. For example, if '`/home`' is an automount point, then accessing '`/home/foo`' would set `${key}` to the string '`foo`'. The key is prefixed by the *pref* option set in the parent mount point. The default prefix is an empty string. If the prefix was '`blah/`' then `${key}` would be set to '`blah/foo`'.

3.3.3.15 map Selector Variable

The name of the mount map being used.

3.3.3.16 netnumber Selector Variable

This selector is identical to the '`in_network`' selector function, see Section 3.3.3.26 [in_network Selector Function], page 30. It will match either the name or number of *any* network interface on which this host is connected to. The names and numbers of all attached interfaces are available from the output of '`amd -v`'.

3.3.3.17 network Selector Variable

This selector is identical to the '`in_network`' selector function, see Section 3.3.3.26 [in_network Selector Function], page 30. It will match either the name or number of *any* network interface on which this host is connected to. The names and numbers of all attached interfaces are available from the output of '`amd -v`'.

3.3.3.18 path Selector Variable

The full pathname of the name being resolved. For example '`/home/foo`' in the example above.

3.3.3.19 wire Selector Variable

This selector is identical to the '`in_network`' selector function, see Section 3.3.3.26 [in_network Selector Function], page 30. It will match either the name or number of *any* network interface on which this host is connected to. The names and numbers of all attached interfaces are available from the output of '`amd -v`'.

3.3.3.20 uid Selector Variable

This selector provides the numeric effective user ID (UID) of the user which last accessed an automounted path name. This simple example shows how floppy mounting can be assigned only to machine owners:

```
floppy  -type:=pcfs \
        uid==2301;host==shekel;dev:=/dev/floppy \
        uid==6712;host==titan;dev=/dev/fd0 \
        uid==0;dev:=/dev/fd0c \
        type:=error
```

The example allows two machine owners to mount floppies on their designated workstations, allows the root user to mount on any host, and otherwise forces an error.

3.3.3.21 gid Selector Variable

This selector provides the numeric effective group ID (GID) of the user which last accessed an automounted path name.

The following boolean functions are selectors which take an argument *ARG*. They return a value of true or false, and thus do not need to be compared with a value. Each of these may be negated by prepending '`!`' to their name.

3.3.3.22 exists Selector Function

If the file listed by *ARG* exists (via **lstat**(2)), this function evaluates to true. Otherwise it evaluates to false.

3.3.3.23 false Selector Function

Always evaluates to false. *ARG* is ignored.

3.3.3.24 netgrp Selector Function

The argument *ARG* of this selector is a netgroup name followed optionally by a comma and a host name. If the host name is not specified, it defaults to `${host}`. If the host name (short name) is a member of the netgroup, this selector evaluates to true. Otherwise it evaluates to false.

For example, suppose you have a netgroup '`ppp-hosts`', and for reasons of performance, these have a local '`/home`' partition, while all other clients on the faster network can access a shared home directory. A common map to use for both might look like the following:

```
home/*  netgrp(ppp-hosts);type:=link;fs:=/local/${key} \
        !netgrp(ppp-hosts);type:=nfs;rhost:=serv1;rfs:=/remote/${key}
```

A more complex example that takes advantage of the two argument netgrp mount selector is given in the following scenario. Suppose one wants to mount the local scratch space from a each host under '`scratch/<hostname>`' and some hosts have their scratch space in a different path than others. Hosts in the netgroup '`apple-hosts`' have their scratch space in the '`/apple`' path, where hosts in the netgroup '`cherry-hosts`' have their scratch space in the '`/cherry`' path. For hosts that are neither in the '`apple-hosts`' or '`cherry-hosts`' netgroups we want to make a symlink pointing to nowhere but provide a descriptive error message in the link destination:

```
scratch/* netgrp(apple-hosts,${/key});type:=nfs;rhost:=${/key};\
    rfs:="/apple" \
netgrp(cherry-hosts,${/key});type:=nfs;rhost:=${/key};\
    rfs:="/cherry" \
type:=link;rfs:="no local partition for ${/key}"
```

3.3.3.25 netgrpd Selector Function

The argument *ARG* of this selector is a netgroup name followed optionally by a comma and a host name. If the host name is not specified, it defaults to `${hostd}`. If the host name (fully-qualified name) is a member of the netgroup, this selector evaluates to true. Otherwise it evaluates to false.

The ‘netgrpd’ function uses fully-qualified host names to match netgroup names, while the ‘netgrp’ function (see Section 3.3.3.24 [netgrp Selector Function], page 29) uses short host names.

3.3.3.26 in_network Selector Function

This selector matches against any network name or number with an optional netmask. First, if the current host has any network interface that is locally attached to the network specified in *ARG* (either via name or number), this selector evaluates to true.

Second, ‘in_network’ supports a network/netmask syntax such as ‘128.59.16.0/255.255.255.0’, ‘128.59.16.0/24’, ‘128.59.16.0/0xffffff00’, or ‘128.59.16.0/’. Using the last form, *Amd* will match the specified network number against the default netmasks of each of the locally attached interfaces.

If the selector does not match, it evaluates to false.

For example, suppose you have two servers that have an exportable ‘/opt’ that smaller clients can NFS mount. The two servers are say, ‘serv1’ on network ‘foo-net.site.com’ and ‘serv2’ on network ‘123.4.5.0’. You can write a map to be used by all clients that will attempt to mount the closest one as follows:

```
opt in_network(foo-net.site.com);rhost:=serv1;rfs:=/opt \
    in_network(123.4.5.0);rhost:=serv2;rfs:=/opt \
    rhost:=fallback-server
```

3.3.3.27 true Selector Function

Always evaluates to true. *ARG* is ignored.

3.3.3.28 xhost Selector Function

This function compares *ARG* against the current hostname, similarly to the Section 3.3.3.7 [host Selector Variable], page 27. However, this function will also match if *ARG* is a CNAME (DNS Canonical Name, or alias) for the current host’s name.

3.3.4 Map Options

Options are parsed concurrently with selectors. The difference is that when an option is seen the string following the ‘:=’ is recorded for later use. As a minimum the *type* option must be specified. Each filesystem type has other options which must also be specified. See Chapter 5 [Filesystem Types], page 45, for details on the filesystem specific options.

Superfluous option specifications are ignored and are not reported as errors.

The following options apply to more than one filesystem type.

3.3.4.1 addopts Option

This option adds additional options to default options normally specified in the ‘/defaults’ entry or the defaults of the key entry being processed (see Section 3.3.4.4 [opts Option], page 31). Normally when you specify ‘opts’ in both the ‘/defaults’ and the map entry, the latter overrides the former completely. But with ‘addopts’ it will append the options and override any conflicting ones.

‘addopts’ also overrides the value of the ‘remopts’ option (see Section 3.3.4.5 [remopts Option], page 36), which unless specified defaults to the value of ‘opts’.

Options which start with '`no`' will override those with the same name that do not start with '`no`' and vice verse. Special handling is given to inverted options such as '`soft`' and '`hard`', '`bg`' and '`fg`', '`ro`' and '`rw`', etc.

For example, if the default options specified were

```
opts:=rw,nosuid,intr,rsize=1024,wsize=1024,quota,posix
```

and the ones specified in a map entry were

```
addopts:=grpid,suid,ro,rsize=2048,quota,nointr
```

then the actual options used would be

```
wsize=1024,posix,grpid,suid,ro,rsize=2048,quota,nointr
```

3.3.4.2 delay Option

The delay, in seconds, before an attempt will be made to mount from the current location. Auxiliary data, such as network address, file handles and so on are computed regardless of this value.

A delay can be used to implement the notion of primary and secondary file servers. The secondary servers would have a delay of a few seconds, thus giving the primary servers a chance to respond first.

3.3.4.3 fs Option

The local mount point. The semantics of this option vary between filesystems.

For NFS and UFS filesystems the value of `${fs}` is used as the local mount point. For other filesystem types it has other meanings which are described in the section describing the respective filesystem type. It is important that this string uniquely identifies the filesystem being mounted. To satisfy this requirement, it should contain the name of the host on which the filesystem is resident and the pathname of the filesystem on the local or remote host.

The reason for requiring the hostname is clear if replicated filesystems are considered. If a fileserver goes down and a replacement filesystem is mounted then the *local* mount point *must* be different from that of the filesystem which is hung. Some encoding of the filesystem name is required if more than one filesystem is to be mounted from any given host.

If the hostname is first in the path then all mounts from a particular host will be gathered below a single directory. If that server goes down then the hung mount points are less likely to be accidentally referenced, for example when **getcwd**(3) traverses the namespace to find the pathname of the current directory.

The '`fs`' option defaults to `${autodir}/${rhost}${rfs}`. In addition, '`rhost`' defaults to the local host name (`${host}`) and '`rfs`' defaults to the value of `${path}`, which is the full path of the requested file; '`/home/foo`' in the example above (see Section 3.3.3 [Selectors], page 26). `${autodir}` defaults to '`/a`' but may be changed with the `-a` command line option. Sun's automounter defaults to '`/tmp_mnt`'. Note that there is no '`/`' between the `${rhost}` and `${rfs}` since `${rfs}` begins with a '`/`'.

3.3.4.4 opts Option

The options to pass to the mount system call. A leading '`-`' is silently ignored. The mount options supported generally correspond to those used by **mount**(8) and are listed below. Some additional pseudo-options are interpreted by *Amd* and are also listed.

Unless specifically overridden, each of the system default mount options applies. Any options not recognized are ignored. If no options list is supplied the string '`rw,defaults`' is used and all the system default mount options apply. Options which are not applicable for a particular operating system are silently ignored. For example, only 4.4BSD is known to implement the `compress` and `spongy` options.

`acdirmax=`*n*
: Set the maximum directory attribute cache timeout to *n*.

`acdirmin=`*n*
: Set the minimum directory attribute cache timeout to *n*.

`acregmax=`*n*
: Set the maximum file attribute cache timeout to *n*.

`acregmin=`*n*
: Set the minimum file attribute cache timeout to *n*.

`actimeo=`*n*
: Set the overall attribute cache timeout to *n*.

`auto`
`ignore`
: Ignore this mount by **df**(1).

`cache`
: Allow data to be cached from a remote server for this mount.

`compress`
: Use NFS compression protocol.

`defperm`
: Ignore the permission mode bits, and default file permissions to 0555, UID 0, and GID 0. Useful for CD-ROMs formatted as ISO-9660.

`dev`
: Allow local special devices on this filesystem.

`dirmask=`*n*
: For PCFS mounts, specify the maximum file permissions for directories in the file system. See the '`mask`' option's description for more details. The mask value of *n* can be specified in decimal, octal, or hexadecimal.

`dumbtimr`
: Turn off the dynamic retransmit timeout estimator. This may be useful for UDP mounts that exhibit high retry rates, since it is possible that the dynamically estimated timeout interval is too short.

`extatt`
: Enable extended attributes in ISO-9660 file systems.

`fsid`
: Set ID of filesystem.

`gens`
: Enable generations in ISO-9660 file systems. Generations allow you to see all versions of a given file.

`group=`*n*
: For PCFS mounts, set the group of the files in the file system to *n* (which can either be a group name or a GID number). The default group is the group of the directory on which the file system is being mounted.

`grpid`
: Use BSD directory group-id semantics.

`int`
`intr`
: Allow keyboard interrupts on hard mounts.

`lock`	Use the NFS locking protocol (default)
`longname`	For PCFS mounts, force Win95 long names.
`mask=`*n*	For PCFS mounts, specify the maximum file permissions for files in the file system. For example, a mask of 755 specifies that, by default, the owner should have read, write, and execute permissions for files, but others should only have read and execute permissions. Only the nine low-order bits of mask are used. The default mask is taken from the directory on which the file system is being mounted. The mask value of *n* can be specified in decimal, octal, or hexadecimal.
`multi`	Perform multi-component lookup on files.
`maxgroups`	Set the maximum number of groups to allow for this mount.
`nfsv3`	Use NFS Version 3 for this mount.
`noac`	Turn off the attribute cache.
`noauto`	This option is used by the mount command in '`/etc/fstab`' or '`/etc/vfstab`' and means not to mount this file system when mount -a is used.
`nocache`	Do not allow data to be cached from a remote server for this mount.
`noconn`	Don't make a connection on datagram transports.
`nocto`	No close-to-open consistency.
`nodefperm`	Do not ignore the permission mode bits. Useful for CD-ROMS formatted as ISO-9660.
`nodev` `nodevs`	Don't allow local special devices on this filesystem.
`noexec`	Don't allow program execution.
`noint`	Do not allow keyboard interrupts for this mount
`nolock`	Do not use the NFS locking protocol
`nomnttab`	This option is used internally to tell Amd that a Solaris 8 system using mntfs is in use.
`norrip`	Turn off using of the Rock Ridge Interchange Protocol (RRIP) extensions to ISO-9660.
`nosub`	Disallow mounts beneath this mount.
`nosuid`	Don't allow set-uid or set-gid executables on this filesystem.
`noversion`	Strip the extension '`;#`' from the version string of files recorded on an ISO-9660 CD-ROM.
`nowin95`	For PCFS mounts, completely ignore Win95 entries.

`optionstr`
: Under Solaris 8, provide the kernel a string of options to parse and show as part of the special in-kernel mount file system.

`overlay`
: Overlay this mount on top of an existing mount, if any.

`pgthresh=`*n*
: Set the paging threshold to *n* kilobytes.

`port=`*n*
: Set the NFS port to *n*.

`posix`
: Turn on POSIX static pathconf for mounts.

`private`
: Use local locking instead of the NLM protocol, useful for IRIX 6 only.

`proplist`
: Support property lists (ACLs) for this mount, useful primarily for Tru64 UNIX.

`proto=`*s*
: Use transport protocol *s* for NFS (can be `"tcp"` or `"udp"`).

`quota`
: Enable quota checking on this mount.

`rdonly`
`ro`
: Mount this filesystem readonly.

`resvport`
: Use a reserved port (smaller than 1024) for remote NFS mounts. Most systems assume that, but some allow for mounts to occur on non-reserved ports. This causes problems when such a system tries to NFS mount one that requires reserved ports. It is recommended that this option always be on.

`retrans=`*n*
: The number of NFS retransmits made before a user error is generated by a '`soft`' mounted filesystem, and before a '`hard`' mounted filesystem reports '`NFS server` *yoyo* `not responding still trying`'.

`retry`
: Set the NFS retry counter.

`rrip`
: Uses the Rock Ridge Interchange Protocol (RRIP) extensions to ISO-9660.

`rsize=`*n*
: The NFS read packet size. You may need to set this if you are using NFS/UDP through a gateway or a slow link.

`rw`
: Allow reads and writes on this filesystem.

`shortname`
: For PCFS mounts, force old DOS short names only.

`soft`
: Give up after *retrans* retransmissions.

`spongy`
: Like '`soft`' for status requests, and '`hard`' for data transfers.

`suid`
: Allow set-uid programs on this mount.

`symttl`
: Turn off the symbolic link cache time-to-live.

`sync`
: Perform synchronous filesystem operations on this mount.

`tcp`
: Use TCP/IP instead of UDP/IP, ignored if the NFS implementation does not support TCP/IP mounts.

`timeo=`*n*
: The NFS timeout, in tenth-seconds, before a request is retransmitted.

`user=`*n*
: For PCFS mounts, set the owner of the files in the file system to *n* (which can either be a user name or a UID number). The default owner is the owner of the directory on which the file system is being mounted.

`vers=`*n*
: Use NFS protocol version number *n* (can be 2 or 3).

`wsize=`*n*
: The NFS write packet size. You may need to set this if you are using NFS/UDP through a gateway or a slow link.

The following options are implemented by *Amd*, rather than being passed to the kernel.

`nounmount`
: Configures the mount so that its time-to-live will never expire. This is the default for non-network based filesystem types (such as mounting local disks, floppies, and CD-ROMs). See also the related *unmount* option.

`ping=`*n*
: The interval, in seconds, between keep-alive pings. When four consecutive pings have failed the mount point is marked as hung. This interval defaults to 30 seconds; if the ping interval is set to zero, *Amd* will use the default 30-second interval. If the interval is set to -1 (or any other negative value), no pings are sent and the host is assumed to be always up, which can cause unmounts to hang See the *softlookup* option for a better alternative. Turning pings off can be useful in NFS-HA (High-Availability) sites where the NFS service rarely goes down. Setting the ping value to a large value can reduce the amount of NFS_NULL chatter on your network considerably, especially in large sites.

  Note that if you have multiple *Amd* entries using the same file server, and each entry sets a different value of N, then each time Amd mounts a new entry, the ping value will be re-evaluated (and updated, turned off, or turned back on as needed). Finally, note that NFS_NULL pings are sent for both UDP and TCP mounts, because even a hung TCP mount can cause user processes to hang.

`public`
: Use WebNFS multi-component lookup on the public file handle instead of the mount protocol to obtain NFS file handles, as documented in the WebNFS Client Specification, RFC 2054. This means that *Amd* will not attempt to contact the remote portmapper or remote mountd daemon, and will only connect to the well-known NFS port 2049 or the port specified with the *port* mount option, thus making it easier to use NFS through a firewall.

`retry=`*n*
: The number of times to retry the mount system call.

`softlookup`
: Configures *Amd*'s behavior with respect to already-mounted shares from NFS fileservers that are unreachable. If softlookup is specified, trying to access such a share will result in an error (EIO, which is changed from the ENOENT 6.0 used to return). If it is not specified, a regular symlink is provided and the access will probably hang in the NFS filesystem.

  The default behavior depends on whether the mount is 'soft' or 'hard'; softlookup can be used to change this default. This is changed from 6.0 which always behaved as if softlookup was specified.

`unmount`
: Configures the mount so that its time-to-live will indeed expire (and thus may be automatically unmounted). This is also the default for network-based filesystem

types (e.g., NFS). This option is useful for removable local media such as CD-ROMs, USB drives, etc. so they can expire when not in use, and get unmounted (such drives can get work out when they keep spinning). See also the related *nounmount* option.

`utimeout=`*n*

The interval, in seconds, that looked up and mounted map entries are cached. After that period of time, *Amd* will attempt to unmount the entries. If, however, the unmount fails (with EBUSY), then *Amd* will extend the mount's time-to-live by the *utimeout* value before the next unmount attempt is made. In fact the interval is extended before the unmount is attempted, to avoid thrashing. The default value is 120 seconds (two minutes) or as set by the `-w` command line option.

`xlatecookie`

Translate directory cookies between 32-long and 64-long lengths.

3.3.4.5 remopts Option

This option has the same use as `${opts}` but applies only when the remote host is on a non-local network. For example, when using NFS across a gateway it is often necessary to use smaller values for the data read and write sizes. This can simply be done by specifying the small values in *remopts*. When a non-local host is accessed, the smaller sizes will automatically be used.

Amd determines whether a host is local by examining the network interface configuration at startup. Any interface changes made after *Amd* has been started will not be noticed. The likely effect will be that a host may incorrectly be declared non-local.

Unless otherwise set, the value of `${remopts}` is the same as the value of `${opts}`.

3.3.4.6 sublink Option

The subdirectory within the mounted filesystem to which the reference should point. This can be used to prevent duplicate mounts in cases where multiple directories in the same mounted filesystem are used.

3.3.4.7 type Option

The filesystem type to be used. See Chapter 5 [Filesystem Types], page 45, for a full description of each type.

4 *Amd* Command Line Options

Many of *Amd*'s parameters can be set from the command line. The command line is also used to specify automount points and maps.

The general format of a command line is

```
amd [options] [{ directory map-name [-map-options] } ...]
```

For each directory and map-name given or specified in the '`amd.conf`' file, *Amd* establishes an automount point. The *map-options* may be any sequence of options or selectors—see Section 3.3 [Location Format], page 24. The *map-options* apply only to *Amd*'s mount point.

'`type:=toplvl;cache:=mapdefault;fs:=${map}`' is the default value for the map options. Default options for a map are read from a special entry in the map whose key is the string '`/defaults`'. When default options are given they are prepended to any options specified in the mount-map locations as explained in Section 3.3.1 [Map Defaults], page 25.

The *options* are any combination of those listed below.

Once the command line has been parsed, the automount points are mounted. The mount points are created if they do not already exist, in which case they will be removed when *Amd* exits. Finally, *Amd* disassociates itself from its controlling terminal and forks into the background.

Note: Even if *Amd* has been built with '`-DDEBUG`' (via `configure --enable-debug`), it will still background itself and disassociate itself from the controlling terminal. To use a debugger it is necessary to specify '`-D daemon`' on the command line. However, even with all of this, mounts and unmounts are performed in the background, and *Amd* will always fork before doing them. Therefore, debugging what happens closely during un/mounts is more challenging.

All of *Amd*'s command options (save `-F` and `-T`) can be specified in the '`amd.conf`' file. See Chapter 6 [Amd Configuration File], page 55. If *Amd* is invoked without any command line options, it will default to using the configuration file '`/etc/amd.conf`', if one exists.

4.1 `-a` *directory*

Specifies the default mount directory. This option changes the variable `${autodir}` which otherwise defaults to '`/a`'. For example, some sites prefer '`/amd`' or '`/n`'.

```
amd -a /amd ...
```

4.2 `-c` *cache-interval*

Selects the period, in seconds, for which a name is cached by *Amd*. If no reference is made to the volume in this period, *Amd* discards the volume name to filesystem mapping.

Once the last reference to a filesystem has been removed, *Amd* attempts to unmount the filesystem. If the unmount fails the interval is extended by a further period as specified by the '`-w`' command line option or by the '`utimeout`' mount option.

The default *cache-interval* is 300 seconds (five minutes).

4.3 `-d` *domain*

Specifies the host's domain. This sets the internal variable `${domain}` and affects the `${hostd}` variable.

If this option is not specified and the hostname already contains the local domain then that is used, otherwise the default value of `${domain}` is '`unknown.domain`'.

For example, if the local domain was '`doc.ic.ac.uk`', *Amd* could be started as follows:

```
amd -d doc.ic.ac.uk ...
```

4.4 `-k` *kernel-architecture*

Specifies the kernel architecture of the system. This is usually the output of '`uname -m`' (the "machine" value gotten from **uname**(2)). If the **uname**(2) system call is not available, the value of `${karch}` defaults to that of `${arch}`.

The only effect of this option is to set the variable `${karch}`.

This option would be used as follows:

```
amd -k `arch -k` ...
```

4.5 `-l` *log-option*

Selects the form of logging to be made. Several special *log-options* are recognized.

1. If *log-option* is the string '`syslog`', *Amd* will use the **syslog**(3) mechanism. If your system supports syslog facilities, then the default facility used is '`LOG_DAEMON`'.
2. When using syslog, if you wish to change the facility, append its name to the log option name, delimited by a single colon. For example, if *log-options* is the string '`syslog:local7`' then **Amd** will log messages via **syslog**(3) using the '`LOG_LOCAL7`' facility. If the facility name specified is not recognized, *Amd* will default to '`LOG_DAEMON`'. Note: while you can use any syslog facility available on your system, it is generally a bad idea to use those reserved for other services such as '`kern`', '`lpr`', '`cron`', etc.
3. If *log-option* is the string '`/dev/stderr`', *Amd* will use standard error, which is also the default target for log messages. To implement this, *Amd* simulates the effect of the '`/dev/fd`' driver.

Any other string is taken as a filename to use for logging. Log messages are appended to the file if it already exists, otherwise a new file is created. The file is opened once and then held open, rather than being re-opened for each message.

Normally, when long-running daemons hold an open file descriptor on a log file, it is impossible to "rotate" the log file and compress older logs on a daily basis. The daemon needs to be told to discard (via **close**(2)) its file handle, and re-open the log file. This is done using `amq -l` *log-option*. See Section 7.4.5 [Amq -l option], page 71.

If the '`syslog`' option is specified but the system does not support syslog or if the named file cannot be opened or created, *Amd* will use standard error. Error messages generated before *Amd* has finished parsing the command line are printed on standard error.

Since *Amd* tends to generate a lot of logging information (especially if debugging was turned on), and due to it being an important program running on the system, it is usually best to log to a separate disk file. In that case *Amd* would be started as follows:

```
amd -l /var/log/amd ...
```

4.6 `-n`

Normalizes the remote hostname before using it. Normalization is done by replacing the value of `${rhost}` with the (generally fully qualified) primary name returned by a hostname lookup.

This option should be used if several names are used to refer to a single host in a mount map.

4.7 `-o` *op-sys-ver*

Overrides the compiled-in version number of the operating system, with *op-sys-ver*. Useful when the built-in version is not desired for backward compatibility reasons. For example, if the built-in version is '`2.5.1`', you can override it to '`5.5.1`', and use older maps that were written with the latter in mind.

4.8 `-p`

Causes *Amd*'s process id to be printed on standard output. This can be redirected to a suitable file for use with kill:

```
amd -p > /var/run/amd.pid ...
```

This option only has an affect if *Amd* is running in daemon mode. If *Amd* is started with the `-D daemon` debug flag, this option is ignored.

4.9 `-r`

Tells *Amd* to restart existing mounts (see Section 5.24 [Inheritance Filesystem], page 54).

4.10 `-t` *timeout.retransmit*

Specifies the RPC *timeout* interval and the *retransmit* counter used by the kernel to communicate to *Amd*. These are used to set the '`timeo`' and '`retrans`' mount options, respectively. The default timeout is 0.8 seconds, and the default number of retransmissions is 11.

Amd relies on the kernel RPC retransmit mechanism to trigger mount retries. The values of these parameters change the overall retry interval. Too long an interval gives poor interactive response; too short an interval causes excessive retries.

4.11 `-v`

Print version information on standard error and then exit. The output is of the form:

```
Copyright (c) 1997-1999 Erez Zadok
Copyright (c) 1990 Jan-Simon Pendry
Copyright (c) 1990 Imperial College of Science, Technology & Medicine
Copyright (c) 1990 The Regents of the University of California.
am-utils version 6.0a15 (build 61).
Built by ezk@example.com on date Wed Oct 22 15:21:03 EDT 1997.
cpu=sparc (big-endian), arch=sun4, karch=sun4u.
full_os=solaris2.5.1, os=sos5, osver=5.5.1, vendor=sun.
Map support for: root, passwd, union, nisplus, nis, ndbm, file, error.
```

```
AMFS: nfs, link, nfsx, nfsl, host, linkx, program, union, inherit,
      ufs, lofs, hsfs, pcfs, auto, direct, toplvl, error.
FS: autofs, cachefs, cdfs, lofs, nfs, nfs3, pcfs, tfs, tmpfs, ufs.
Network 1: wire="mcl-lab-net.cs.columbia.edu" (netnumber=128.59.13).
Network 2: wire="14-net.cs.columbia.edu" (netnumber=128.59.14).
Network 3: wire="old-net.cs.columbia.edu" (netnumber=128.59.16).
```

The information includes the version number, number of times *Amd* was compiled on the local system, release date and name of the release. Following come the cpu type, byte ordering, and the architecture and kernel architecture as `${arch}` and `${karch}`, respectively. The next line lists the operating system full name, short name, version, and vendor. These four values correspond to the variables `${full_os}`, `${os}`, `${osver}`, and `${vendor}`, respectively. See Chapter 2 [Supported Platforms], page 17.

Then come a list of map types supported, filesystems internally supported by *Amd* (AMFS), and generic filesystems available (FS). Finally all known networks (if any) of this host are listed by name and number. They are available via the variables `${wire}` or `${network}`, and `${netnumber}` (see Section 3.3.3 [Selectors], page 26) or the '`in_network`' selector function (see Section 3.3.3.26 [in_network Selector Function], page 30).

4.12 `-w` *wait-timeout*

Selects the interval in seconds between unmount attempts after the initial time-to-live has expired.

This defaults to 120 seconds (two minutes).

4.13 `-x` *opts*

Specifies the type and verbosity of log messages. *opts* is a comma separated list selected from the following options:

`fatal`	Fatal errors (cannot be turned off)
`error`	Non-fatal errors (cannot be turned off)
`user`	Non-fatal user errors
`warn`	Recoverable errors
`warning`	Alias for `warn`
`info`	Information messages
`map`	Mount map usage
`stats`	Additional statistics
`all`	All of the above
`defaults`	An alias for "fatal,error,user,warning,info".

Initially a set of default logging flags is enabled. This is as if '`-x defaults`' or '`-x fatal,error,user,warning,info`' had been selected. The command line is parsed and logging is controlled by the `-x` option. The very first set of logging flags is saved and can not be subsequently disabled using *Amq*. This default set of options is useful for general production use.

The '`info`' messages include details of what is mounted and unmounted and when filesystems have timed out. If you want to have the default set of messages without the '`info`' messages then you simply need '`-x noinfo`'. The messages given by '`user`' relate to errors in the mount maps, so these are useful when new maps are installed. The following table lists the syslog priorities used for each of the message types.

`fatal`	'`LOG_CRIT`'
`error`	'`LOG_ERR`'
`user`	'`LOG_WARNING`'
`warning`	'`LOG_WARNING`'
`info`	'`LOG_INFO`'
`debug`	'`LOG_DEBUG`'
`map`	'`LOG_DEBUG`'
`stats`	'`LOG_INFO`'

The options can be prefixed by the string '`no`' to indicate that this option should be turned off. For example, to obtain all but '`info`' messages the option '`-x all,noinfo`' would be used.

If *Amd* was built with debugging enabled the `debug` option is automatically enabled regardless of the command line options.

4.14 `-y` *NIS-domain*

Selects an alternate NIS domain. This is useful for debugging and cross-domain shared mounting. If this flag is specified, *Amd* immediately attempts to bind to a server for this domain.

4.15 `-A` *architecture*

Specifies the OS architecture of the system. The only effect of this option is to set the variable `${arch}`.

This option would be used as follows:

```
amd -A i386 ...
```

4.16 `-C` *cluster-name*

Specifies the name of the cluster of which the local machine is a member. The only effect is to set the variable `${cluster}`. The *cluster-name* is will usually obtained by running another command which uses a database to map the local hostname into a cluster name. `${cluster}` can then be used as a selector to restrict mounting of replicated data. If this option is not given, `${cluster}` has the same value as `${domain}`. This would be used as follows:

```
amd -C `clustername` ...
```

4.17 `-D` *opts*

Controls the verbosity and coverage of the debugging trace; *opts* is a comma separated list of debugging options. The `-D` option is only available if *Amd* was compiled with '`-DDEBUG`', or configured with `configure --enable-debug`. The memory debugging facilities ('`mem`') are only available if *Amd* was compiled with '`-DDEBUG_MEM`' (in addition to '`-DDEBUG`'), or configured with `configure --enable-debug=mem`.

The most common options to use are '`-D trace`' and '`-D test`' (which turns on all the useful debug options). As usual, every option can be prefixed with '`no`' to turn it off.

`all`	all options (excluding hrtime and mtab)
`defaults`	"sensible" default options (all–excluding hrtime, mtab, and xdrtrace)
`test`	full debug options plus mtab,nodaemon,nofork,noamq
`amq`	register *Amd* with the RPC portmapper, for *Amq*
`daemon`	enter daemon mode
`fork`	fork child worker (hlfsd only)
`full`	program trace
`hrtime`	print high resolution time stamps (only if **syslog**(3) is not used)
`info`	info service specific debugging (hesiod, nis, etc.) In the case of hesiod maps, turns on the hesiod RES_DEBUG internal debugging option.
`mem`	trace memory allocations. Needs to be explicitly enabled at compile time with –enable-debug=mem.
`mtab`	use local mount-table file (defaults to '`/tmp/mtab`', see Section 6.5.6 [debug_mtab_file Parameter], page 59)
`readdir`	show readdir progress
`str`	debug string munging
`trace`	trace RPC protocol and NFS mount arguments
`xdrtrace`	trace XDR routines

You may also refer to the program source for a more detailed explanation of the available options.

4.18 `-F` *conf-file*

Specify an *Amd* configuration file *conf-file* to use. For a description of the format and syntax, see Chapter 6 [Amd Configuration File], page 55. This configuration file is used to specify any options in lieu of typing many of them on the command line. The '`amd.conf`' file includes directives for every command line option *Amd* has, and many more that are only available via the configuration file facility. The configuration file specified by this option is processed after all other options had been processed, regardless of the actual location of this option on the command line.

4.19 `-H`

Print a brief help and usage string.

4.20 `-O` *op-sys-name*

Overrides the compiled-in name of the operating system, with *op-sys-name*. Useful when the built-in name is not desired for backward compatibility reasons. For example, if the build in name is '`sunos5`', you can override it to the old name '`sos5`', and use older maps which were written with the latter in mind.

4.21 `-S`

Do *not* lock the running executable pages of *Amd* into memory. To improve *Amd*'s performance, systems that support the **plock**(3) or **mlockall**(2) call lock the *Amd* process into memory. This way there is less chance the operating system will schedule, page out, and swap the *Amd* process as needed. This tends to improve *Amd*'s performance, at the cost of reserving the memory used by the *Amd* process (making it unavailable for other processes). If this behavior is not desired, use the `-S` option.

4.22 `-T` *tag*

Specify a tag to use with '`amd.conf`'. All map entries tagged with *tag* will be processed. Map entries that are not tagged are always processed. Map entries that are tagged with a tag other than *tag* will not be processed.

5 Filesystem Types

To mount a volume, *Amd* must be told the type of filesystem to be used. Each filesystem type typically requires additional information such as the fileserver name for NFS.

From the point of view of *Amd*, a *filesystem* is anything that can resolve an incoming name lookup. An important feature is support for multiple filesystem types. Some of these filesystems are implemented in the local kernel and some on remote fileservers, whilst the others are implemented internally by *Amd*.

The two common filesystem types are UFS and NFS. Four other user accessible filesystems (‘`link`’, ‘`program`’, ‘`auto`’ and ‘`direct`’) are also implemented internally by *Amd* and these are described below. There are two additional filesystem types internal to *Amd* which are not directly accessible to the user (‘`inherit`’ and ‘`error`’). Their use is described since they may still have an effect visible to the user.

5.1 Network Filesystem (‘`nfs`’)

The *nfs* (‘`type:=nfs`’) filesystem type provides access to Sun’s NFS.

The following options must be specified:

`rhost` the remote fileserver. This must be an entry in the hosts database. IP addresses are not accepted. The default value is taken from the local host name (`${host}`) if no other value is specified.

`rfs` the remote filesystem. If no value is specified for this option, an internal default of `${path}` is used.

NFS mounts require a two stage process. First, the *file handle* of the remote file system must be obtained from the server. Then a mount system call must be done on the local system. *Amd* keeps a cache of file handles for remote file systems. The cache entries have a lifetime of a few minutes.

If a required file handle is not in the cache, *Amd* sends a request to the remote server to obtain it.

Historically, this documentation has maintained that *Amd* will try all the locations in parallel and use the first one which responds with a valid file handle. This has not been the case for quite some time, however. Instead, *Amd* will go through each location, one by one, and will only skip to the next one if the previous one either fails or times out.

An NFS entry might be:

```
jsp  host!=charm;type:=nfs;rhost:=charm;rfs:=/home/charm;sublink:=jsp
```

The mount system call and any unmount attempts are always done in a new task to avoid the possibility of blocking *Amd*.

5.2 Network Host Filesystem (‘`host`’)

The *host* (‘`type:=host`’) filesystem allows access to the entire export tree of an NFS server. The implementation is layered above the ‘`nfs`’ implementation so keep-alives work in the same way. The only option which needs to be specified is ‘`rhost`’ which is the name of the fileserver to mount.

The '`host`' filesystem type works by querying the mount daemon on the given fileserver to obtain its export list. *Amd* then obtains filehandles for each of the exported filesystems. Any errors at this stage cause that particular filesystem to be ignored. Finally each filesystem is mounted. Again, errors are logged but ignored. One common reason for mounts to fail is that the mount point does not exist. Although *Amd* attempts to automatically create the mount point, it may be on a remote filesystem to which *Amd* does not have write permission.

When an attempt to unmount a '`host`' filesystem mount fails, *Amd* remounts any filesystems which had successfully been unmounted. To do this *Amd* queries the mount daemon again and obtains a fresh copy of the export list. *Amd* then tries to mount any exported filesystems which are not currently mounted.

Sun's automounter provides a special '`-hosts`' map. To achieve the same effect with *Amd* requires two steps. First a mount map must be created as follows:

```
*       type:=host;rhost:=${key};fs:=${autodir}/${rhost}/root
```

and then start *Amd* with the following command

```
amd /net net.map
```

where '`net.map`' is the name of map described above. Note that the value of `${fs}` is overridden in the map. This is done to avoid a clash between the mount tree and any other filesystem already mounted from the same fileserver.

If different mount options are needed for different hosts then additional entries can be added to the map, for example

```
host2       opts:=ro,nosuid,soft
```

would soft mount '`host2`' read-only.

5.3 Network Filesystem Group ('`nfsx`')

The *nfsx* ('`type:=nfsx`') filesystem allows a group of filesystems to be mounted from a single NFS server. The implementation is layered above the '`nfs`' implementation so keep-alives work in the same way.

WARNING: '`nfsx`' is meant to be a "last resort" kind of solution. It is racy and poorly supported. The authors *highly* recommend that other solutions be considered before relying on it.

The options are the same as for the '`nfs`' filesystem with one difference for '`rfs`', as explained below.

The following options should be specified:

`rhost` the remote fileserver. The default value is taken from the local host name (`${host}`) if no other value is specified.

`rfs` is a list of filesystems to mount, and must be specified. The list is in the form of a comma separated strings.

For example:

```
pub type:=nfsx;rhost:=gould;\
    rfs:=/public,/,graphics,usenet;fs:=${autodir}/${rhost}/root
```

The first string defines the root of the tree, and is applied as a prefix to the remaining members of the list which define the individual filesystems. The first string is *not* used as a filesystem name. A serial operation is used to determine the local mount points to ensure a consistent layout of a tree of mounts.

Here, the *three* filesystems, '`/public`', '`/public/graphics`' and '`/public/usenet`', would be mounted.

A local mount point, `${fs}`, *must* be specified. The default local mount point will not work correctly in the general case. A suggestion is to use '`fs:=${autodir}/${rhost}/root`'.

5.4 Unix Filesystem ('`ufs`', '`xfs`', or '`efs`')

The *ufs* ('`type:=ufs`') filesystem type provides access to the system's standard disk filesystem—usually a derivative of the Berkeley Fast Filesystem.

The following option must be specified:

`dev`	the block special device to be mounted.

A UFS entry might be:

```
jsp   host==charm;type:=ufs;dev:=/dev/sd0d;sublink:=jsp
```

UFS is the default Unix disk-based file system, which Am-utils picks up during the autoconfiguration phase. Some systems have more than one type, such as IRIX, that comes with EFS (Extent File System) and XFS (Extended File System). In those cases, you may explicitly set the file system type, by using entries such:

```
ez1   type:=efs;dev:=/dev/xd0a
ez2   type:=xfs;dev:=/dev/sd3c
```

The UFS/XFS/EFS filesystems are never timed out by default, i.e. they will never be unmounted by *Amd*. If automatic unmounting is desired, the "unmount" option should be added to the mount options for the entry.

5.5 Caching Filesystem ('`cachefs`')

The *cachefs* ('`type:=cachefs`') filesystem caches files from one location onto another, presumably providing faster access. It is particularly useful to cache from a larger and remote (slower) NFS partition to a smaller and local (faster) UFS directory.

The following options must be specified:

`cachedir`	the directory where the cache is stored.
`rfs`	the path name to the "back file system" to be cached from.
`fs`	the "front file system" mount point to the cached files, where *Amd* will set a symbolic link pointing to.

A CacheFS entry for, say, the '`/import`' *Amd* mount point, might be:

```
copt  type:=cachefs;cachedir:=/cache;rfs:=/import/opt;fs:=/n/import/copt
```

Access to the pathname '`/import/copt`' will follow a symbolic link to '`/n/import/copt`'. The latter is the mount point for a caching file system, that caches from '`/import/opt`' to '`/cache`'.

The cachefs filesystem is never timed out by default, i.e. it will never be unmounted by *Amd*. If automatic unmounting is desired, the "unmount" option should be added to the mount options for the entry.

Caveats:

1. This file system is currently only implemented for Solaris 2.x!
2. Before being used for the first time, the cache directory *must* be initialized with '`cfsadmin -c` *`cachedir`*'. See the manual page for **cfsadmin**(1M) for more information.
3. The "back file system" mounted must be a complete file system, not a subdirectory thereof; otherwise you will get an error "Invalid Argument".
4. If *Amd* aborts abnormally, the state of the cache may be inconsistent, requiring running the command '`fsck -F cachefs` *`cachedir`*'. Otherwise you will get the error "No Space Left on Device".

5.6 CD-ROM Filesystem ('`cdfs`')

The *cdfs* ('`type:=cdfs`') filesystem mounts a CD-ROM with an ISO9660 format filesystem on it.

The following option must be specified:

`dev`   the block special device to be mounted.

Some operating systems will fail to mount read-only CDs unless the '`ro`' option is specified. A cdfs entry might be:

```
cdfs      os==sunos4;type:=cdfs;dev:=/dev/sr0 \
          os==sunos5;addopts:=ro;type:=cdfs;dev:=/dev/dsk/c0t6d0s2
```

5.7 Loopback Filesystem ('`lofs`')

The *lofs* ('`type:=lofs`') filesystem is also called the loopback filesystem. It mounts a local directory on another, thus providing mount-time binding to another location (unlike symbolic links).

The loopback filesystem is particularly useful within the context of a chroot-ed directory (via **chroot**(2)), to provide access to directories otherwise inaccessible.

The following option must be specified:

`rfs`   the pathname to be mounted on top of `${fs}`.

Usually, the FTP server runs in a chroot-ed environment, for security reasons. In this example, lofs is used to provide a subdirectory within a user's home directory, also available for public ftp.

```
lofs      type:=lofs;rfs:=/home/ezk/myftpdir;fs:=/usr/ftp/pub/ezk
```

5.8 Memory/RAM Filesystem ('`mfs`')

The *mfs* ('`type:=mfs`') filesystem is available in 4.4BSD, Linux, and other systems. It creates a filesystem in a portion of the system's memory, thus providing very fast file (volatile) access.

XXX: THIS FILESYSTEM IS NOT IMPLEMENTED YET!

5.9 Null Filesystem (‘`nullfs`’)

The *nullfs* (‘`type:=nullfs`’) filesystem is available from 4.4BSD, and is very similar to the loopback filesystem, *lofs*.

XXX: THIS FILESYSTEM IS NOT IMPLEMENTED YET!

5.10 Floppy Filesystem (‘`pcfs`’)

The *pcfs* (‘`type:=pcfs`’) filesystem mounts a floppy previously formatted for the MS-DOS format.

The following option must be specified:

`dev`	the block special device to be mounted.

A pcfs entry might be:

```
pcfs      os==sunos4;type:=pcfs;dev:=/dev/fd0 \
          os==sunos5;type:=pcfs;dev:=/dev/diskette
```

5.11 Translucent Filesystem (‘`tfs`’)

The *tfs* (‘`type:=tfs`’) filesystem is an older version of the 4.4BSD *unionfs*.

XXX: THIS FILESYSTEM IS NOT IMPLEMENTED YET!

5.12 Shared Memory+Swap Filesystem (‘`tmpfs`’)

The *tmpfs* (‘`type:=tmpfs`’) filesystem shares memory between a the swap device and the rest of the system. It is generally used to provide a fast access ‘`/tmp`’ directory, one that uses memory that is otherwise unused. This filesystem is available in SunOS 4.x and 5.x.

XXX: THIS FILESYSTEM IS NOT IMPLEMENTED YET!

5.13 User ID Mapping Filesystem (‘`umapfs`’)

The *umapfs* (‘`type:=umapfs`’) filesystem maps User IDs of file ownership, and is available from 4.4BSD.

XXX: THIS FILESYSTEM IS NOT IMPLEMENTED YET!

5.14 Program Filesystem (‘`program`’)

The *program* (‘`type:=program`’) filesystem type allows a program to be run whenever a mount or unmount is required. This allows easy addition of support for other filesystem types, such as MIT’s Remote Virtual Disk (RVD) which has a programmatic interface via the commands ‘`rvdmount`’ and ‘`rvdunmount`’.

Both of the following options must be specified:

`mount`	the program which will perform the mount.
`unmount`	
`umount`	the program which will perform the unmount. For convenience, you may use either ‘`unmount`’ or ‘`umount`’ but not both. If neither is defined, *Amd* will default to ‘`umount ${fs}`’ (the actual unmount program pathname will be automatically determined at the time GNU `configure` runs.)

The exit code from these two programs is interpreted as a Unix error code. As usual, exit code zero indicates success. To execute the program, *Amd* splits the string on whitespace to create an array of substrings. Single quotes '`''`' can be used to quote whitespace if that is required in an argument. There is no way to escape or change the single quote character.

To run e.g. the program '`rvdmount`' with a host name and filesystem as arguments, it would be specified by '`fs:=${autodir}${path};type:=program;mount:="/etc/rvdmount rvdmount fserver ${fs}";unmount:="/etc/rdvumount rvdumount ${fs}"`'.

The first element in the array is taken as the pathname of the program to execute. The other members of the array form the argument vector to be passed to the program, *including argument zero*. The array is exactly the same as the array passed to the execv() system call (man execv for details). The split string must have at least two elements. The programs are directly executed by *Amd*, not via a shell. Therefore, if a script is to be used as a mount/umount program, it *must* begin with a `#!` interpreter specification.

Often, this program mount type is used for Samba mounts, where you need a double slash in pathnames. However, *Amd* normalizes sequences of slashes into one slash. Therefore, you must use an escaped slash, preceded by an escaped backslash. So to get a double slash in the mount command, you need the eight character sequence '`\\\/\\\/`' in your map. For example:

'`mount="/sbin/mount mount -r -t smbfs -o-N,-Ihostname \\\/\\\/guest@venus/mp3"`'

If a filesystem type is to be heavily used, it may be worthwhile adding a new filesystem type into *Amd*, but for most uses the program filesystem should suffice.

When the program is run, standard input and standard error are inherited from the current values used by *Amd*. Standard output is a duplicate of standard error. The value specified with the `-l` command line option has no effect on standard error.

Amd guarantees that the mountpoint will be created before calling the mount program, and that it will be removed after the umount program returns success.

5.15 Symbolic Link Filesystem ('`link`')

Each filesystem type creates a symbolic link to point from the volume name to the physical mount point. The '`link`' filesystem does the same without any other side effects. This allows any part of the machines name space to be accessed via *Amd*.

One common use for the symlink filesystem is '`/homes`' which can be made to contain an entry for each user which points to their (auto-mounted) home directory. Although this may seem rather expensive, it provides a great deal of administrative flexibility.

The following option must be defined:

`fs` The value of *fs* option specifies the destination of the link, as modified by the *sublink* option. If *sublink* is non-null, it is appended to `${fs}/` and the resulting string is used as the target.

The '`link`' filesystem can be thought of as identical to the '`ufs`' filesystem but without actually mounting anything.

An example entry might be:

```
jsp   host==charm;type:=link;fs:=/home/charm;sublink:=jsp
```

which would return a symbolic link pointing to '`/home/charm/jsp`'.

5.16 Symbolic Link Filesystem II (‘linkx’)

The *linkx* (‘type:=linkx’) filesystem type is identical to ‘link’ with the exception that the target of the link must exist. Existence is checked with the **lstat**(2) system call.

The ‘linkx’ filesystem type is particularly useful for wildcard map entries. In this case, a list of possible targets can be given and *Amd* will choose the first one which exists on the local machine.

5.17 NFS-Link Filesystem (‘nfsl’)

The *nfsl* (‘type:=nfsl’) filesystem type is a combination of two others: ‘link’ and ‘nfs’. If the local host name is equal to the value of ${rhost} *and* the target pathname listed in ${fs} exists, ‘nfsl’ will behave exactly as ‘type:=link’, and refer to the target as a symbolic link. If the local host name is not equal to the value of ${rhost}, or if the target of the link does not exist, *Amd* will treat it as ‘type:=nfs’, and will mount a remote pathname for it.

The ‘nfsl’ filesystem type is particularly useful as a shorthand for the more cumbersome and yet one of the most popular *Amd* entries. For example, you can simplify all map entries that look like:

```
zing    -fs:=/n/shekel/u/zing \
        host!=shekel;type:=nfs;rhost:=shekel;rfs:=${fs} \
        host==shekel;type:=link
```

or

```
zing    -fs:=/n/shekel/u/zing \
        exists(${fs});type:=link \
        !exists(${fs});type:=nfs;rhost:=shekel;rfs:=${fs}
```

into a shorter form

```
zing    type:=nfsl;fs:=/n/shekel/u/zing;rhost:=shekel;rfs:=${fs}
```

Not just does it make the maps smaller and simpler, but it avoids possible mistakes that often happen when forgetting to set up the two entries (one for ‘type:=nfs’ and the other for ‘type:=link’) necessary to perform transparent mounts of existing or remote mounts.

5.18 Automount Filesystem (‘auto’)

The *auto* (‘type:=auto’) filesystem type creates a new automount point below an existing automount point. Top-level automount points appear as system mount points. An automount mount point can also appear as a sub-directory of an existing automount point. This allows some additional structure to be added, for example to mimic the mount tree of another machine.

The following options may be specified:

`cache` specifies whether the data in this mount-map should be cached. The default value is ‘none’, in which case no caching is done in order to conserve memory.

However, better performance and reliability can be obtained by caching some or all of a mount-map.

If the cache option specifies ‘all’, the entire map is enumerated when the mount point is created.

If the cache option specifies '`inc`', caching is done incrementally as and when data is required. Some map types do not support cache mode '`all`', in which case '`inc`' is used whenever '`all`' is requested.

Caching can be entirely disabled by using cache mode '`none`'.

If the cache option specifies '`regexp`' then the entire map will be enumerated and each key will be treated as an egrep-style regular expression. The order in which a cached map is searched does not correspond to the ordering in the source map so the regular expressions should be mutually exclusive to avoid confusion.

Each mount map type has a default cache type, usually '`inc`', which can be selected by specifying '`mapdefault`'.

The cache mode for a mount map can only be selected on the command line. Starting *Amd* with the command:

```
amd /homes hesiod.homes -cache:=inc
```

will cause '`/homes`' to be automounted using the *Hesiod* name server with local incremental caching of all successfully resolved names.

All cached data is forgotten whenever *Amd* receives a '`SIGHUP`' signal and, if cache '`all`' mode was selected, the cache will be reloaded. This can be used to inform *Amd* that a map has been updated. In addition, whenever a cache lookup fails and *Amd* needs to examine a map, the map's modify time is examined. If the cache is out of date with respect to the map then it is flushed as if a '`SIGHUP`' had been received.

An additional option ('`sync`') may be specified to force *Amd* to check the map's modify time whenever a cached entry is being used. For example, an incremental, synchronized cache would be created by the following command:

```
amd /homes hesiod.homes -cache:=inc,sync
```

`fs` specifies the name of the mount map to use for the new mount point.

Arguably this should have been specified with the `${rfs}` option but we are now stuck with it due to historical accident.

`pref` alters the name that is looked up in the mount map. If `${pref}`, the *prefix*, is non-null then it is prepended to the name requested by the kernel *before* the map is searched. The default prefix is the prefix of the parent map (if any) with name of the auto node appended to it. That means if you want no prefix you must say so in the map: '`pref:=null`'.

`opts` Normally, '`auto`' style maps are not browsable even if you turn on directory browsability (see Section 6.4.2 [browsable_dirs Parameter], page 56). To enable browsing entries in '`auto`' maps, specify '`opts:=browsable`' or '`opts:=fullybrowsable`' in the description of this map.

The server '`dylan.doc.ic.ac.uk`' has two user disks: '`/dev/dsk/2s0`' and '`/dev/dsk/5s0`'. These are accessed as '`/home/dylan/dk2`' and '`/home/dylan/dk5`' respectively. Since '`/home`' is already an automount point, this naming is achieved with the following map entries:

```
dylan        type:=auto;fs:=${map};pref:=${key}/
dylan/dk2    type:=ufs;dev:=/dev/dsk/2s0
dylan/dk5    type:=ufs;dev:=/dev/dsk/5s0
```

5.19 Direct Automount Filesystem (‘`direct`’)

The *direct* (‘`type:=direct`’) filesystem is almost identical to the automount filesystem. Instead of appearing to be a directory of mount points, it appears as a symbolic link to a mounted filesystem. The mount is done at the time the link is accessed. See Section 5.18 [Automount Filesystem], page 51, for a list of required options.

Direct automount points are created by specifying the ‘`direct`’ filesystem type on the command line:

```
amd ... /usr/man auto.direct -type:=direct
```

where ‘`auto.direct`’ would contain an entry such as:

```
usr/man     -type:=nfs;rfs:=/usr/man \
            rhost:=man-server1  rhost:=man-server2
```

In this example, ‘`man-server1`’ and ‘`man-server2`’ are file servers which export copies of the manual pages. Note that the key which is looked up is the name of the automount point without the leading ‘`/`’.

Note that the implementation of the traditional *direct* filesystem is essentially a hack (pretending that the root of an NFS filesystem is a symlink) and many modern operating systems get very unhappy about it. For example, Linux kernel 2.4+ completely disallows it, and Solaris 2.8 fails to unmount it when *Amd* shuts down. Therefore, the use of the traditional *direct* filesystem is strongly discouraged; it is only semi-supported, at best.

The autofs implementations that permit direct mounts are fully supported, however. That currently includes all versions of Solaris. Linux autofs does NOT support direct mounts at all.

5.20 Union Filesystem (‘`union`’)

The *union* (‘`type:=union`’) filesystem type allows the contents of several directories to be merged and made visible in a single directory. This can be used to overcome one of the major limitations of the Unix mount mechanism which only allows complete directories to be mounted.

For example, supposing ‘`/tmp`’ and ‘`/var/tmp`’ were to be merged into a new directory called ‘`/mtmp`’, with files in ‘`/var/tmp`’ taking precedence. The following command could be used to achieve this effect:

```
amd ... /mtmp union:/tmp:/var/tmp -type:=union
```

Currently, the unioned directories must *not* be automounted. That would cause a deadlock. This seriously limits the current usefulness of this filesystem type and the problem will be addressed in a future release of *Amd*.

Files created in the union directory are actually created in the last named directory. This is done by creating a wildcard entry which points to the correct directory. The wildcard entry is visible if the union directory is listed, so allowing you to see which directory has priority.

The files visible in the union directory are computed at the time *Amd* is started, and are not kept up-to-date with respect to the underlying directories. Similarly, if a link is removed, for example with the '`rm`' command, it will be lost forever.

5.21 Error Filesystem ('`error`')

The *error* ('`type:=error`') filesystem type is used internally as a catch-all in the case where none of the other filesystems was selected, or some other error occurred. Lookups and mounts always fail with "No such file or directory". All other operations trivially succeed.

The error filesystem is not directly accessible.

5.22 Top-level Filesystem ('`toplvl`')

The *toplvl* ('`type:=toplvl`') filesystems is derived from the '`auto`' filesystem and is used to mount the top-level automount nodes. Requests of this type are automatically generated from the command line arguments.

5.23 Root Filesystem ('`root`')

The *root* ('`type:=root`') filesystem type acts as an internal placeholder onto which *Amd* can pin '`toplvl`' mounts. Only one node of this type need ever exist and one is created automatically during startup. The effect of having more than one root node is undefined.

The root filesystem is not directly accessible.

5.24 Inheritance Filesystem ('`inherit`')

The *inheritance* ('`type:=inherit`') filesystem is not directly accessible. Instead, internal mount nodes of this type are automatically generated when *Amd* is started with the `-r` option. At this time the system mount table is scanned to locate any filesystems which are already mounted. If any reference to these filesystems is made through *Amd* then instead of attempting to mount it, *Amd* simulates the mount and *inherits* the filesystem. This allows a new version of *Amd* to be installed on a live system simply by killing the old daemon with '`SIGTERM`' and starting the new one.

This filesystem type is not generally visible externally, but it is possible that the output from '`amq -m`' may list '`inherit`' as the filesystem type. This happens when an inherit operation cannot be completed for some reason, usually because a fileserver is down.

6 Amd Configuration File

The '`amd.conf`' file is the configuration file for *Amd*, as part of the am-utils suite. This file contains runtime configuration information for the *Amd* automounter program.

6.1 File Format

The '`amd.conf`' file consists of sections and parameters. A section begins with the name of the section in square brackets '`[]`' and continues until the next section begins or the end of the file is reached. Sections contain parameters of the form '`name = value`'.

The file is line-based — that is, each newline-terminated line represents either a comment, a section name or a parameter. No line-continuation syntax is available.

Section names, parameter names and their values are case sensitive.

Only the first equals sign in a parameter is significant. Whitespace before or after the first equals sign is discarded. Leading, trailing and internal whitespace in section and parameter names is irrelevant. Leading and trailing whitespace in a parameter value is discarded. Internal whitespace within a parameter value is not allowed, unless the whole parameter value is quoted with double quotes as in '`name = "some value"`'.

Any line beginning with a pound sign '`#`' is ignored, as are lines containing only whitespace.

The values following the equals sign in parameters are all either a string (no quotes needed if string does not include spaces) or a boolean, which may be given as '`yes`'/'`no`'. Case is significant in all values. Some items such as cache timeouts are numeric.

6.2 The Global Section

The global section must be specified as '`[global]`'. Parameters in this section either apply to *Amd* as a whole, or to all other regular map sections which follow. There should be only one global section defined in one configuration file.

It is highly recommended that this section be specified first in the configuration file. If it is not, then regular map sections which precede it will not use global values defined later.

6.3 Regular Map Sections

Parameters in regular (non-global) sections apply to a single map entry. For example, if the map section '`[/homes]`' is defined, then all parameters following it will be applied to the '`/homes`' *Amd*-managed mount point.

6.4 Common Parameters

These parameters can be specified either in the global or a map-specific section. Entries specified in a map-specific section override the default value or one defined in the global section. If such a common parameter is specified only in the global section, it is applicable to all regular map sections that follow.

6.4.1 `autofs_use_lofs` Parameter

(type=string, default=‘`yes`’). When set to ‘`yes`’, *Amd*’s autofs code will use lofs-type (loopback) mounts for `type:=link` mounts, as well as several other cases that require local references. This has the advantage that *Amd* does not use a secondary mount point and users do not see external pathnames (the infamous `/bin/pwd` problem, where it reports a different path than the user chdir’ed into). One of the disadvantages of using this option is that the autofs code is relatively new and the in-place mounts have not been throughly tested.

If this option is set to ‘`no`’, then *Amd*’s autofs code will use symlinks instead of lofs-type mounts for local references. This has the advantage of using simpler (more stable) code, but at the expense of negating one of autofs’s big advantages: the hiding of *Amd*’s internal paths. Note that symlinks are not supported in all autofs implementations, especially those derived from Solaris Autofs v1. Also, on Solaris 2.6 and newer, autofs symlinks are not cached, resulting in repeated up-call requests to *Amd*.

6.4.2 `browsable_dirs` Parameter

(type=string, default=‘`no`’). If ‘`yes`’, then *Amd*’s top-level mount points will be browsable to **readdir**(3) calls. This means you could run for example **ls**(1) and see what keys are available to mount in that directory. Not all entries are made visible to **readdir**(3): the ‘`/defaults`’ entry, wildcard entries, and those with a ‘`/`’ in them are not included. If you specify ‘`full`’ to this option, all but the ‘`/defaults`’ entry will be visible. Note that if you run a command which will attempt to **stat**(2) the entries, such as often done by ‘`ls -l`’ or ‘`ls -F`’, *Amd* will attempt to mount *every* entry in that map. This is often called a “mount storm”.

Note that mount storms are mostly avoided by using autofs mounts (‘`mount_type = autofs`’).

6.4.3 `map_defaults` Parameter

(type=string, default to empty). This option sets a string to be used as the map’s `/defaults` entry, overriding any `/defaults` specified in the map. This allows local users to override a given map’s defaults without modifying maps globally (which is impossible in sites where the maps are managed by a different administrative group).

6.4.4 `map_options` Parameter

(type=string, default no options). This option is the same as specifying map options on the command line to *Amd*, such as ‘`cache:=all`’.

6.4.5 `map_type` Parameter

(type=string, default search all map types). If specified, *Amd* will initialize the map only for the type given. This is useful to avoid the default map search type used by *Amd* which takes longer and can have undesired side-effects such as initializing NIS even if not used. Possible values are

‘`file`’	plain files
‘`hesiod`’	Hesiod name service from MIT

'`ldap`'	Lightweight Directory Access Protocol
'`ndbm`'	(New) dbm style hash files
'`nis`'	Network Information Services (version 2)
'`nisplus`'	Network Information Services Plus (version 3)
'`passwd`'	local password files
'`union`'	union maps

6.4.6 `mount_type` Parameter

(type=string, default='`nfs`'). All *Amd* mount types default to NFS. That is, *Amd* is an NFS server on the map mount points, for the local host it is running on. If '`autofs`' is specified, *Amd* will be an autofs server for those mount points.

6.4.7 `search_path` Parameter

(type=string, default no search path). This provides a (colon-delimited) search path for file maps. Using a search path, sites can allow for local map customizations and overrides, and can distributed maps in several locations as needed.

6.4.8 `selectors_in_defaults` Parameter

(type=boolean, default='`no`'). If '`yes`', then the '`/defaults`' entry of maps will search for and process any selectors before setting defaults for all other keys in that map. Useful when you want to set different options for a complete map based on some parameters. For example, you may want to better the NFS performance over slow slip-based networks as follows:

```
/defaults \
    wire==slip-net;opts:=intr,rsize=1024,wsize=1024 \
    wire!=slip-net;opts:=intr,rsize=8192,wsize=8192
```

Deprecated form: selectors_on_default.

6.4.9 `sun_map_syntax` Parameter

(type=boolean, default='`no`'). If '`yes`', then *Amd* will parse the map according to the Sun Automount syntax.

6.5 Global Parameters

The following parameters are applicable to the '`[global]`' section only.

6.5.1 `arch` Parameter

(type=string, default to compiled in value). Same as the `-A` option to *Amd*. Allows you to override the value of the *arch Amd* variable.

6.5.2 `auto_attrcache` Parameter

(type=numeric, default=0). Specify in seconds (or units of 0.1 seconds, depending on the OS), what is the (kernel-side) NFS attribute cache timeout for *Amd*'s own automount points. A value of 0 is supposed to turn off attribute caching, meaning that *Amd* will be consulted via a kernel-RPC each time someone stat()'s the mount point (which could be abused as a denial-of-service attack).

WARNING: *Amd* depends on being able to turn off the NFS attribute cache of the client OS. If it cannot be turned off, then users may get ESTALE errors or symlinks that point to the wrong places. This is more likely under heavy use of *Amd*, for example if your system is experiencing frequent map changes or frequent mounts/unmounts. Therefore, under normal circumstances, this parameter should remain set to 0, to ensure that the attribute cache is indeed off.

Unfortunately, some kernels (e.g., certain BSDs) don't have a way to turn off the NFS attribute cache. Setting this parameter to 0 is supposed to turn off attribute caching entirely, but unfortunately it does not; instead, the attribute cache is set to some internal hard-coded default (usually anywhere from 5-30 seconds). If you suspect that your OS doesn't have a reliable way of turning off the attribute cache, then it is better to set this parameter to the smallest possible non-zero value (set '`auto_attrcache=1`' in your `amd.conf`). This will not eliminate the problem, but reduce the risk window somewhat. The best solutions are (1) to use *Amd* in Autofs mode, if it's supported in your OS, and (2) talk to your OS vendor to support a true '`noac`' flag. See the README.attrcache (`http://www.am-utils.org/docs/am-utils/attrcache.txt`) document for more details.

If you are able to turn off the attribute cache on your OS, alas, *Amd*'s performance may degrade (when not using Autofs) because every traversal of an automounter-controlled pathname will result in a lookup request from the kernel to *Amd*. Under heavy loads, for example when using recursive tools like '`find`', '`rdist`', or '`rsync`', this performance degradation can be noticeable. There are two possible solutions that some administrators have chosen to improve performance:

1. First, you can turn off unmounting using the '`nounmount`' mount option. This will ensure that no *Amd* symlink could ever change, thereby the kernel's attribute cache and *Amd* will always be in sync. However, this method will cause the number of mounts to keep growing, even if some are no longer in use; this has the disadvantage that your system could be more susceptible to hangs if even one of those accumulating mounts hangs due to a downed server.
2. Second, you can turn on attribute caching carefully by setting a small automounter attribute cache value (say, one second), and a relatively large dismount interval (say, one hour). (See Section 6.5.8 [dismount_interval Parameter], page 59.) For example, you can set this in your `amd.conf`:

   ```
   [global]
   auto_attrcache = 1
   dismount_interval = 3600
   ```

   This has the benefit of using the kernel's attribute cache and thus improving performance. The disadvantage with this option is that the window of vulnerability is not eliminated entirely: it is only made smaller.

6.5.3 `auto_dir` Parameter

(type=string, default='`/a`'). Same as the `-a` option to *Amd*. This sets the private directory where *Amd* will create sub-directories for its real mount points.

6.5.4 `cache_duration` Parameter

(type=numeric, default=300). Same as the `-c` option to *Amd*. Sets the duration in seconds that looked-up or mounted map entries remain in the cache.

6.5.5 `cluster` Parameter

(type=string, default no cluster). Same as the `-C` option to *Amd*. Specifies the alternate HP-UX cluster to use.

6.5.6 `debug_mtab_file` Parameter

(type=string, default="/tmp/mtab"). Path to mtab file that is used by *Amd* to store a list of mounted file systems during debug-mtab mode. This option only applies to systems that store mtab information on disk.

6.5.7 `debug_options` Parameter

(type=string, default no debug options). Same as the `-D` option to *Amd*. Specify any debugging options for *Amd*. Works only if am-utils was configured for debugging using the `--enable-debug` option. The additional '`mem`' option can be turned on via `--enable-debug=mem`. Otherwise debugging options are ignored. Options are comma delimited, and can be preceded by the string '`no`' to negate their meaning. You can get the list of supported debugging and logging options by running `amd -H`. Possible values those listed for the -D option. See Section 4.17 [-D Option], page 42.

6.5.8 `dismount_interval` Parameter

(type=numeric, default=120). Same as the `-w` option to *Amd*. Specify in seconds, the time between attempts to dismount file systems that have exceeded their cached times.

6.5.9 `domain_strip` Parameter

(type=boolean, default='`yes`'). If '`yes`', then the domain name part referred to by `${rhost}` is stripped off. This is useful to keep logs and smaller. If '`no`', then the domain name part is left changed. This is useful when using multiple domains with the same maps (as you may have hosts whose domain-stripped name is identical).

6.5.10 `exec_map_timeout` Parameter

(type=numeric, default=10). The timeout in seconds that *Amd* will wait for an executable map program before an answer is returned from that program (or script). This value should be set to as small as possible while still allowing normal replies to be returned before the timer expires, because during the time that the executable map program is queried, *Amd* is essentially waiting and is thus not responding to any other queries. See Section 3.1.9 [Executable maps], page 23.

6.5.11 `forced_unmounts` Parameter

(type=boolean, default='`no`'). Sometimes, mount points are hung due to unrecoverable conditions, such as when NFS servers migrate, change their IP address, are down permanently, or due to hardware failures, and more. In this case, attempting to unmount an existing mount point, or even just to **stat**(2) it, results in one of three fatal errors: EIO, ESTALE, or EBUSY. At that point, *Amd* can do little to recover that hung point (in fact, the OS cannot automatically recover either). For that reason, some OSs support special kinds of forced unmounts, which must be used very carefully: they will force an unmount immediately (or lazily on Linux), which could result in application data loss. However, that may be the only way to recover the entire host (without rebooting). Once a hung mount point is forced out, *Amd* can then re-mount a replacement one (if available), bringing a mostly-hung system back to operation and avoiding a potentially costly reboot.

If the '`forced_unmounts`' option is set to '`yes`', and the client OS supports forced or lazy unmounts, then *Amd* will attempt to use them if it gets any of the three serious error conditions listed above. Note that *Amd* will force the unmount of mount points that returned EBUSY only for '`type:=toplvl`' mounts (see Section 5.22 [Top-level Filesystem], page 54): that is, *Amd*'s own mount points. This is useful to recover from a previously hung *Amd*, and to ensure that an existing *Amd* can shutdown cleanly even if some processes are keeping its mount points busy (i.e., when a user's shell process uses `cd` to set its CWD to *Amd*'s own mount point).

If this option is set to '`no`' (the default), then *Amd* will not attempt this special recovery procedure.

6.5.12 `full_os` Parameter

(type=string, default to compiled in value). The full name of the operating system, along with its version. Allows you to override the compiled-in full name and version of the operating system. Useful when the compiled-in name is not desired. For example, the full operating system name on linux comes up as '`linux`', but you can override it to '`linux-2.2.5`'.

6.5.13 `fully_qualified_hosts` Parameter

(type=string, default='`no`'). If '`yes`', *Amd* will perform RPC authentication using fully-qualified host names. This is necessary for some systems, and especially when performing cross-domain mounting. For this function to work, the *Amd* variable '`${hostd}`' is used, requiring that '`${domain}`' not be null.

6.5.14 `hesiod_base` Parameter

(type=string, default='`automount`'). Specify the base name for hesiod maps.

6.5.15 `karch` Parameter

(type=string, default to karch of the system). Same as the `-k` option to *Amd*. Allows you to override the kernel-architecture of your system. Useful for example on Sun (Sparc) machines, where you can build one *Amd* binary, and run it on multiple machines, yet you want each one to get the correct *karch* variable set (for example, sun4c, sun4m, sun4u, etc.) Note that if not specified, *Amd* will use **uname**(2) to figure out the kernel architecture of the machine.

6.5.16 `ldap_base` Parameter

(type=string, default not set). Specify the base name for LDAP. This often includes LDAP-specific values such as country and organization.

6.5.17 `ldap_cache_maxmem` Parameter

(type=numeric, default=131072). Specify the maximum memory *Amd* should use to cache LDAP entries.

6.5.18 `ldap_cache_seconds` Parameter

(type=numeric, default=0). Specify the number of seconds to keep entries in the cache.

6.5.19 `ldap_hostports` Parameter

(type=string, default not set). Specify the LDAP host and port values.

6.5.20 `ldap_proto_version` Parameter

(type=numeric, default=2). Specify the LDAP protocol version to use. With a value of 3 will use LDAPv3 protocol.

6.5.21 `local_domain` Parameter

(type=string, default no sub-domain). Same as the `-d` option to *Amd*. Specify the local domain name. If this option is not given the domain name is determined from the hostname, by removing the first component of the fully-qualified host name.

6.5.22 `localhost_address` Parameter

(type=string, default to localhost or 127.0.0.1). Specify the name or IP address for *Amd* to use when connecting the sockets for the local NFS server and the RPC server. This defaults to 127.0.0.1 or whatever the host reports as its local address. This parameter is useful on hosts with multiple addresses where you want to force *Amd* to connect to a specific address.

6.5.23 `log_file` Parameter

(type=string, default=‘`stderr`’). Same as the `-l` option to *Amd*. Specify a file name to log *Amd* events to. If the string ‘`/dev/stderr`’ is specified, *Amd* will send its events to the standard error file descriptor.

If the string ‘`syslog`’ is given, *Amd* will record its events with the system logger **syslogd**(8). If your system supports syslog facilities, then the default facility used is ‘`LOG_DAEMON`’.

When using syslog, if you wish to change the facility, append its name to the option name, delimited by a single colon. For example, if it is the string ‘`syslog:local7`’ then *Amd* will log messages via **syslog**(3) using the ‘`LOG_LOCAL7`’ facility. If the facility name specified is not recognized, *Amd* will default to ‘`LOG_DAEMON`’. Note: while you can use any syslog facility available on your system, it is generally a bad idea to use those reserved for other services such as ‘`kern`’, ‘`lpr`’, ‘`cron`’, etc.

6.5.24 `log_options` Parameter

(type=string, default="defaults"). Same as the `-x` option to *Amd*. Specify any logging options for *Amd*. Options are comma delimited, and can be preceded by the string ‘`no`’ to negate their meaning. The ‘`debug`’ logging option is only available if am-utils was configured with `--enable-debug`. You can get the list of supported debugging options by running `amd -H`. Possible values are:

'`all`'	all messages
'`defaults`'	an alias for "fatal,error,user,warning,info"
'`debug`'	debug messages
'`error`'	non-fatal system errors (cannot be turned off)
'`fatal`'	fatal errors (cannot be turned off)
'`info`'	information
'`map`'	map errors
'`stats`'	additional statistical information
'`user`'	non-fatal user errors
'`warn`'	warnings
'`warning`'	warnings

6.5.25 `map_reload_interval` Parameter

(type=numeric, default=3600). The number of seconds that *Amd* will wait before it checks to see if any maps have changed at their source (NIS servers, LDAP servers, files, etc.). *Amd* will reload only those maps that have changed.

6.5.26 `nfs_allow_any_interface` Parameter

(type=string, default='`no`'). Normally *Amd* accepts local NFS packets only from 127.0.0.1. If this parameter is set to '`yes`', then *amd* will accept local NFS packets from any local interface; this is useful on hosts that may have multiple interfaces where the system is forced to send all outgoing packets (even those bound to the same host) via an address other than 127.0.0.1.

6.5.27 `nfs_allow_insecure_port` Parameter

(type=string, default='`no`'). Normally *Amd* will refuse requests coming from unprivileged ports (i.e., ports >= 1024 on Unix systems), so that only privileged users and the kernel can send NFS requests to it. However, some kernels (certain versions of Darwin, MacOS X, and Linux) have bugs that cause them to use unprivileged ports in certain situations, which causes *Amd* to stop dead in its tracks. This parameter allows *Amd* to operate normally even on such systems, at the expense of a slight decrease in the security of its operations. If you see messages like "ignoring request from foo:1234, port not reserved" in your *Amd* log, try enabling this parameter and give it another go.

6.5.28 `nfs_proto` Parameter

(type=string, default to trying version tcp then udp). By default, *Amd* tries `tcp` and then `udp`. This option forces the overall NFS protocol used to TCP or UDP. It overrides what is in the *Amd* maps, and is useful when *Amd* is compiled with TCP support in NFSv2/NFSv3 that may not be stable. With this option you can turn off the complete usage of TCP for NFS dynamically (without having to recompile *Amd*), and use UDP only, until such time as TCP support is desired again.

6.5.29 `nfs_retransmit_counter` Parameter

(type=numeric, default=11). Same as the *retransmit* part of the `-t` *timeout.retransmit* option to *Amd*. Specifies the number of NFS retransmissions that the kernel will use to communicate with *Amd* using either UDP or TCP mounts. See Section 4.10 [-t Option], page 39.

6.5.30 `nfs_retransmit_counter_udp` Parameter

(type=numeric, default=11). Same as the *nfs_retransmit_counter* parameter, but applied globally only to UDP mounts. See Section 6.5.29 [nfs_retransmit_counter Parameter], page 63.

6.5.31 `nfs_retransmit_counter_tcp` Parameter

(type=numeric, default=11). Same as the *nfs_retransmit_counter* parameter, but applied globally only to TCP mounts. See Section 6.5.29 [nfs_retransmit_counter Parameter], page 63.

6.5.32 `nfs_retransmit_counter_toplvl` Parameter

(type=numeric, default=11). Same as the *nfs_retransmit_counter* parameter, applied only for *Amd*'s top-level UDP mounts. On some systems it is useful to set this differently than the OS default, so as to better tune *Amd*'s responsiveness under heavy scheduler loads. See Section 6.5.29 [nfs_retransmit_counter Parameter], page 63.

6.5.33 `nfs_retry_interval` Parameter

(type=numeric, default=8). Same as the *timeout* part of the `-t` *timeout.retransmit* option to *Amd*. Specifies the NFS timeout interval, in *tenths* of seconds, between NFS/RPC retries (for UDP or TCP). This is the value that the kernel will use to communicate with *Amd*. See Section 4.10 [-t Option], page 39.

Amd relies on the kernel RPC retransmit mechanism to trigger mount retries. The values of the *nfs_retransmit_counter* and the *nfs_retry_interval* parameters change the overall retry interval. Too long an interval gives poor interactive response; too short an interval causes excessive retries.

6.5.34 `nfs_retry_interval_udp` Parameter

(type=numeric, default=8). Same as the *nfs_retry_interval* parameter, but applied globally only to UDP mounts. See Section 6.5.33 [nfs_retry_interval Parameter], page 63.

6.5.35 `nfs_retry_interval_tcp` Parameter

(type=numeric, default=8). Same as the *nfs_retry_interval* parameter, but applied globally only to TCP mounts. See Section 6.5.33 [nfs_retry_interval Parameter], page 63.

6.5.36 `nfs_retry_interval_toplvl` Parameter

(type=numeric, default=8). Same as the *nfs_retry_interval* parameter, applied only for *Amd*'s top-level UDP mounts. On some systems it is useful to set this differently than the OS default, so as to better tune *Amd*'s responsiveness under heavy scheduler loads. See Section 6.5.33 [nfs_retry_interval Parameter], page 63.

6.5.37 `nfs_vers` Parameter

(type=numeric, default to trying version 3 then 2). By default, *Amd* tries version 3 and then version 2. This option forces the overall NFS protocol used to version 3 or 2. It overrides what is in the *Amd* maps, and is useful when *Amd* is compiled with NFSv3 support that may not be stable. With this option you can turn off the complete usage of NFSv3 dynamically (without having to recompile *Amd*), and use NFSv2 only, until such time as NFSv3 support is desired again.

6.5.38 `nis_domain` Parameter

(type=string, default to local NIS domain name). Same as the `-y` option to *Amd*. Specify an alternative NIS domain from which to fetch the NIS maps. The default is the system domain name. This option is ignored if NIS support is not available.

6.5.39 `normalize_hostnames` Parameter

(type=boolean, default='`no`'). Same as the `-n` option to *Amd*. If '`yes`', then the name referred to by `${rhost}` is normalized relative to the host database before being used. The effect is to translate aliases into "official" names.

6.5.40 `normalize_slashes` Parameter

(type=boolean, default='`yes`'). If '`yes`' then amd will condense all multiple `/` (slash) characters into one and remove all trailing slashes. If '`no`', then amd will not touch strings that may contain repeated or trailing slashes. The latter is sometimes useful with SMB mounts, which often require multiple slash characters in pathnames.

6.5.41 `os` Parameter

(type=string, default to compiled in value). Same as the `-O` option to *Amd*. Allows you to override the compiled-in name of the operating system. Useful when the built-in name is not desired for backward compatibility reasons. For example, if the built-in name is '`sunos5`', you can override it to '`sos5`', and use older maps which were written with the latter in mind.

6.5.42 `osver` Parameter

(type=string, default to compiled in value). Same as the `-o` option to *Amd*. Allows you to override the compiled-in version number of the operating system. Useful when the built-in version is not desired for backward compatibility reasons. For example, if the build in version is '`2.5.1`', you can override it to '`5.5.1`', and use older maps that were written with the latter in mind.

6.5.43 `pid_file` Parameter

(type=string, default='`/dev/stdout`'). Specify a file to store the process ID of the running daemon into. If not specified, *Amd* will print its process id onto the standard output. Useful for killing *Amd* after it had run. Note that the PID of a running *Amd* can also be retrieved via *Amq* (see Section 7.4.7 [Amq -p option], page 72).

This file is used only if the '`print_pid`' option is on (see Section 6.5.47 [print_pid Parameter], page 65).

6.5.44 `plock` Parameter

(type=boolean, default='`yes`'). Same as the `-S` option to *Amd*. If '`yes`', lock the running executable pages of *Amd* into memory. To improve *Amd*'s performance, systems that support the **plock**(3) or **mlockall**(2) call can lock the *Amd* process into memory. This way there is less chance the operating system will schedule, page out, and swap the *Amd* process as needed. This improves *Amd*'s performance, at the cost of reserving the memory used by the *Amd* process (making it unavailable for other processes).

6.5.45 `portmap_program` Parameter

(type=numeric, default=300019). Specify an alternate Port-mapper RPC program number, other than the official number. This is useful when running multiple *Amd* processes. For example, you can run another *Amd* in "test" mode, without affecting the primary *Amd* process in any way. For safety reasons, the alternate program numbers that can be specified must be in the range 300019-300029, inclusive. *Amq* has an option `-P` which can be used to specify an alternate program number of an *Amd* to contact. In this way, amq can fully control any number of *Amd* processes running on the same host.

6.5.46 `preferred_amq_port` Parameter

(type=numeric, default=0). Specify an alternate Port-mapper RPC port number for *Amd*'s *Amq* service. This is used for both UDP and TCP. Setting this value to 0 (or not defining it) will cause *Amd* to select an arbitrary port number. Setting the *Amq* RPC service port to a specific number is useful in firewalled or NAT'ed environments, where you need to know which port *Amd* will listen on.

6.5.47 `print_pid` Parameter

(type=boolean, default='`no`'). Same as the `-p` option to *Amd*. If '`yes`', *Amd* will print its process ID upon starting.

6.5.48 `print_version` Parameter

(type=boolean, default='`no`'). Same as the `-v` option to *Amd*, but the version prints and *Amd* continues to run. If '`yes`', *Amd* will print its version information string, which includes some configuration and compilation values.

6.5.49 `restart_mounts` Parameter

(type=boolean, default='`no`'). Same as the `-r` option to *Amd*. If '`yes`' *Amd* will scan the mount table to determine which file systems are currently mounted. Whenever one of these would have been auto-mounted, *Amd* inherits it.

6.5.50 `show_statfs_entries` Parameter

(type=boolean), default='`no`'). If '`yes`', then all maps which are browsable will also show the number of entries (keys) they have when **df**(1) runs. (This is accomplished by returning non-zero values to the **statfs**(2) system call).

6.5.51 `truncate_log` Parameter

(type=boolean), default=‘`no`’). If ‘`yes`’, then *Amd* will truncate the log file (if it’s a regular file) on startup. This could be useful when conducting extensive testing on *Amd* maps (or *Amd* itself) and you don’t want to see log data from a previous run in the same file.

6.5.52 `unmount_on_exit` Parameter

(type=boolean, default=‘`no`’). If ‘`yes`’, then *Amd* will attempt to unmount all file systems which it knows about. Normally it leaves all (esp. NFS) mounted file systems intact. Note that *Amd* does not know about file systems mounted before it starts up, unless the ‘`restart_mounts`’ option is used (see Section 6.5.49 [restart_mounts Parameter], page 65).

6.5.53 `use_tcpwrappers` Parameter

(type=boolean), default=‘`yes`’). If ‘`yes`’, then amd will use the tcpwrappers (tcpd/librwap) library (if available) to control access to *Amd* via the `/etc/hosts.allow` and `/etc/hosts.deny` files. *Amd* will verify that the host running *Amq* is authorized to connect. The `amd` service name must used in the `/etc/hosts.allow` and `/etc/hosts.deny` files. For example, to allow only localhost to connect to *Amd*, add this line to `/etc/hosts.allow`:

```
amd: localhost
```

and this line to `/etc/hosts.deny`:

```
amd: ALL
```

Consult the man pages for **hosts_access**(5) for more information on using the tcpwrappers access-control library.

Note that in particular, you should not configure your `hosts.allow` file to spawn a command for *Amd*: that will cause *Amd* to not be able to `waitpid` on the child process ID of any background un/mount that *Amd* issued, resulting in a confused *Amd* that does not know what happened to those background un/mount requests.

6.5.54 `vendor` Parameter

(type=string, default to compiled in value). The name of the vendor of the operating system. Overrides the compiled-in vendor name. Useful when the compiled-in name is not desired. For example, most Intel based systems set the vendor name to ‘`unknown`’, but you can set it to ‘`redhat`’.

6.6 Regular Map Parameters

The following parameters are applicable only to regular map sections.

6.6.1 map_name Parameter

(type=string, must be specified). Name of the map where the keys are located.

6.6.2 tag Parameter

(type=string, default no tag). Each map entry in the configuration file can be tagged. If no tag is specified, that map section will always be processed by *Amd*. If it is specified, then *Amd* will process the map if the `-T` option was given to *Amd*, and the value given to that command-line option matches that in the map section.

6.7 amd.conf Examples

The following is the actual `amd.conf` file I used at the Computer Science Department of Columbia University.

```
# GLOBAL OPTIONS SECTION
[ global ]
normalize_hostnames =    no
print_pid =              no
#pid_file =              /var/run/amd.pid
restart_mounts =         yes
#unmount_on_exit =       yes
auto_dir =               /n
log_file =               /var/log/amd
log_options =            all
#debug_options =         defaults
plock =                  no
selectors_in_defaults =  yes
# config.guess picks up "sunos5" and I don't want to edit my maps yet
os =                     sos5
# if you print_version after setting up "os", it will show it.
print_version =          no
map_type =               file
search_path =            /etc/amdmaps:/usr/lib/amd:/usr/local/AMD/lib
browsable_dirs =         yes
fully_qualified_hosts =  no

# DEFINE AN AMD MOUNT POINT
[ /u ]
map_name =               amd.u

[ /proj ]
map_name =               amd.proj

[ /src ]
map_name =               amd.src

[ /misc ]
map_name =               amd.misc

[ /import ]
map_name =               amd.import

[ /tftpboot/.amd ]
tag =                    tftpboot
map_name =               amd.tftpboot
```

7 Run-time Administration

7.1 Starting *Amd*

Amd is best started from '`/etc/rc.local`' on BSD systems, or from the appropriate start-level script in '`/etc/init.d`' on System V systems.

```
if [ -f /usr/local/sbin/ctl-amd ]; then
    /usr/local/sbin/ctl-amd start; (echo -n ' amd') > /dev/console
fi
```

The shell script, '`ctl-amd`' is used to start, stop, or restart *Amd*. It is a relatively generic script. All options you want to set should not be made in this script, but rather updated in the '`amd.conf`' file. See Chapter 6 [Amd Configuration File], page 55.

If you do not wish to use an *Amd* configuration file, you may start *Amd* manually. For example, getting the map entries via NIS:

```
amd -r -l /var/log/amd `ypcat -k auto.master`
```

7.2 Stopping *Amd*

Amd stops in response to two signals.

'`SIGTERM`' causes the top-level automount points to be unmounted and then *Amd* to exit. Any automounted filesystems are left mounted. They can be recovered by restarting *Amd* with the `-r` command line option.

'`SIGINT`' causes *Amd* to attempt to unmount any filesystems which it has automounted, in addition to the actions of '`SIGTERM`'. This signal is primarily used for debugging.

Actions taken for other signals are undefined.

The easiest and safest way to stop *Amd*, without having to find its process ID by hand, is to use the '`ctl-amd`' script, as with:

```
ctl-amd stop
```

7.3 Restarting *Amd*

Before *Amd* can be started, it is vital to ensure that no other *Amd* processes are managing any of the mount points, and that the previous process(es) have terminated cleanly. When a terminating signal is set to *Amd*, the automounter does *not* terminate right then. Rather, it starts by unmounting all of its managed mount mounts in the background, and then terminates. It usually takes a few seconds for this process to happen, but it can take an arbitrarily longer time. If two or more *Amd* processes attempt to manage the same mount point, it usually will result in a system lockup.

The easiest and safest way to restart *Amd*, without having to find its process ID by hand, sending it the '`SIGTERM`' signal, waiting for *Amd* to die cleanly, and verifying so, is to use the '`ctl-amd`' script, as with:

```
ctl-amd restart
```

The script will locate the process ID of *Amd*, kill it, and wait for it to die cleanly before starting a new instance of the automounter. '`ctl-amd`' will wait for a total of 30 seconds for *Amd* to die, and will check once every 5 seconds if it had.

7.4 Controlling *Amd*

It is sometimes desirable or necessary to exercise external control over some of *Amd*'s internal state. To support this requirement, *Amd* implements an RPC interface which is used by the *Amq* program. A variety of information is available.

Amq generally applies an operation, specified by a single letter option, to a list of mount points. The default operation is to obtain statistics about each mount point. This is similar to the output shown above but includes information about the number and type of accesses to each mount point.

7.4.1 *Amq* default information

With no arguments, *Amq* obtains a brief list of all existing mounts created by *Amd*. This is different from the list displayed by **df**(1) since the latter only includes system mount points.

The output from this option includes the following information:

- the automount point,
- the filesystem type,
- the mount map or mount information,
- the internal, or system mount point.

For example:

```
/             root   "root"                      sky:(pid75)
/homes        toplvl /usr/local/etc/amd.homes   /homes
/home         toplvl /usr/local/etc/amd.home    /home
/homes/jsp    nfs    charm:/home/charm          /a/charm/home/charm/jsp
/homes/phjk   nfs    toytown:/home/toytown      /a/toytown/home/toytown/ai/phjk
```

If an argument is given then statistics for that volume name will be output. For example:

```
What         Uid   Getattr Lookup RdDir   RdLnk   Statfs Mounted@
/homes       0     1196    512    22      0       30     90/09/14 12:32:55
/homes/jsp   0     0       0      0       1180    0      90/10/13 12:56:58
```

`What`	the volume name.
`Uid`	ignored.
`Getattr`	the count of NFS *getattr* requests on this node. This should only be non-zero for directory nodes.
`Lookup`	the count of NFS *lookup* requests on this node. This should only be non-zero for directory nodes.
`RdDir`	the count of NFS *readdir* requests on this node. This should only be non-zero for directory nodes.
`RdLnk`	the count of NFS *readlink* requests on this node. This should be zero for directory nodes.

`Statfs`	the count of NFS *statfs* requests on this node. This should only be non-zero for top-level automount points.
`Mounted@`	the date and time the volume name was first referenced.

7.4.2 *Amq* `-f` option

The `-f` option causes *Amd* to flush the internal mount map cache. This is useful for example in Hesiod maps since *Amd* will not automatically notice when they have been updated. The map cache can also be synchronized with the map source by using the '`sync`' option (see Section 5.18 [Automount Filesystem], page 51).

7.4.3 *Amq* `-h` option

By default the local host is used. In an HP-UX cluster the root server is used since that is the only place in the cluster where *Amd* will be running. To query *Amd* on another host the `-h` option should be used.

7.4.4 *Amq* `-H` option

Print a brief help and usage string.

7.4.5 *Amq* `-l` option

Tell *Amd* to use *log_file* as the log file name. For security reasons, this *must* be the same log file which *Amd* used when started. This option is therefore only useful to refresh *Amd*'s open file handle on the log file, so that it can be rotated and compressed via daily cron jobs.

7.4.6 *Amq* `-m` option

The `-m` option displays similar information about mounted filesystems, rather than automount points. The output includes the following information:

- the mount information,
- the mount point,
- the filesystem type,
- the number of references to this filesystem,
- the server hostname,
- the state of the file server,
- any error which has occurred.

For example:

```
"root"           truth:(pid602)     root   1 localhost is up
hesiod.home      /home              toplvl 1 localhost is up
hesiod.vol       /vol               toplvl 1 localhost is up
hesiod.homes     /homes             toplvl 1 localhost is up
amy:/home/amy    /a/amy/home/amy    nfs    5 amy is up
swan:/home/swan  /a/swan/home/swan  nfs    0 swan is up (Permission denied)█
ex:/home/ex      /a/ex/home/ex      nfs    0 ex is down
```

When the reference count is zero the filesystem is not mounted but the mount point and server information is still being maintained by *Amd*.

7.4.7 *Amq* `-p` option

Return the process ID of the remote or locally running *Amd*. Useful when you need to send a signal to the local *Amd* process, and would rather not have to search through the process table. This option is used in the '`ctl-amd`' script.

7.4.8 *Amq* `-P` option

Contact an alternate running *Amd* that had registered itself on a different RPC *program_number* and apply all other operations to that instance of the automounter. This is useful when you run multiple copies of *Amd*, and need to manage each one separately. If not specified, *Amq* will use the default program number for *Amd*, 300019. For security reasons, the only alternate program numbers *Amd* can use range from 300019 to 300029, inclusive.

For example, to kill an alternate running *Amd*:

```
kill `amq -p -P 300020`
```

7.4.9 *Amq* `-s` option

The `-s` option displays global statistics. If any other options are specified or any filesystems named then this option is ignored. For example:

```
requests  stale     mount     mount     unmount
deferred  fhandles  ok        failed    failed
1054      1         487       290       7017
```

'`Deferred requests`'
: are those for which an immediate reply could not be constructed. For example, this would happen if a background mount was required.

'`Stale filehandles`'
: counts the number of times the kernel passes a stale filehandle to *Amd*. Large numbers indicate problems.

'`Mount ok`'
: counts the number of automounts which were successful.

'`Mount failed`'
: counts the number of automounts which failed.

'`Unmount failed`'
: counts the number of times a filesystem could not be unmounted. Very large numbers here indicate that the time between unmount attempts should be increased.

7.4.10 *Amq* `-T` option

The `-T` option causes the *Amq* to contact *Amd* using the TCP transport only (connection oriented). Normally, *Amq* will use TCP first, and if that failed, will try UDP.

7.4.11 *Amq* `-U` option

The `-U` option causes the *Amq* to contact *Amd* using the UDP transport only (connectionless). Normally, *Amq* will use TCP first, and if that failed, will try UDP.

7.4.12 *Amq* `-u` option

The `-u` option causes the time-to-live interval of the named mount points to be expired, thus causing an unmount attempt. This is the only safe way to unmount an automounted filesystem. It is not possible to unmount a filesystem which has been mounted with the '`nounmount`' flag.

7.4.13 *Amq* `-v` option

The `-v` option displays the version of *Amd* in a similar way to *Amd*'s `-v` option.

7.4.14 *Amq* `-w` option

The `-w` option translates a full pathname as returned by **getpwd**(3) into a short *Amd* pathname that goes through its mount points. This option requires that *Amd* is running.

7.4.15 Other *Amq* options

Two other operations are implemented. These modify the state of *Amd* as a whole, rather than any particular filesystem. The `-x` and `-D` options have exactly the same effect as *Amd*'s corresponding command line options.

When *Amd* receives the `-x` flag, it disallows turning off the '`fatal`' or '`error`' flags. Both are on by default. They are mandatory so that *Amd* could report important errors, including errors relating to turning flags on/off.

8 FSinfo

XXX: this chapter should be reviewed by someone knowledgeable with fsinfo.

8.1 *FSinfo* overview

FSinfo is a filesystem management tool. It has been designed to work with *Amd* to help system administrators keep track of the ever increasing filesystem namespace under their control.

The purpose of *FSinfo* is to generate all the important standard filesystem data files from a single set of input data. Starting with a single data source guarantees that all the generated files are self-consistent. One of the possible output data formats is a set of *Amd* maps which can be used among the set of hosts described in the input data.

FSinfo implements a declarative language. This language is specifically designed for describing filesystem namespace and physical layouts. The basic declaration defines a mounted filesystem including its device name, mount point, and all the volumes and access permissions. *FSinfo* reads this information and builds an internal map of the entire network of hosts. Using this map, many different data formats can be produced including '`/etc/fstab`', '`/etc/exports`', *Amd* mount maps and '`/etc/bootparams`'.

8.2 Using *FSinfo*

The basic strategy when using *FSinfo* is to gather all the information about all disks on all machines into one set of declarations. For each machine being managed, the following data is required:

- Hostname
- List of all filesystems and, optionally, their mount points.
- Names of volumes stored on each filesystem.
- NFS export information for each volume.
- The list of static filesystem mounts.

The following information can also be entered into the same configuration files so that all data can be kept in one place.

- List of network interfaces
- IP address of each interface
- Hardware address of each interface
- Dumpset to which each filesystem belongs
- and more ...

To generate *Amd* mount maps, the automount tree must also be defined (see Section 8.8 [FSinfo automount definitions], page 82). This will have been designed at the time the volume names were allocated. Some volume names will not be automounted, so *FSinfo* needs an explicit list of which volumes should be automounted.

Hostnames are required at several places in the *FSinfo* language. It is important to stick to either fully qualified names or unqualified names. Using a mixture of the two will inevitably result in confusion.

Sometimes volumes need to be referenced which are not defined in the set of hosts being managed with *FSinfo*. The required action is to add a dummy set of definitions for the host and volume names required. Since the files generated for those particular hosts will not be used on them, the exact values used is not critical.

8.3 *FSinfo* grammar

FSinfo has a relatively simple grammar. Distinct syntactic constructs exist for each of the different types of data, though they share a common flavor. Several conventions are used in the grammar fragments below.

The notation, *list(*`xxx`*)*, indicates a list of zero or more `xxx`'s. The notation, *opt(*`xxx`*)*, indicates zero or one `xxx`. Items in double quotes, *eg* `"host"`, represent input tokens. Items in angle brackets, *eg* <*hostname*>, represent strings in the input. Strings need not be in double quotes, except to differentiate them from reserved words. Quoted strings may include the usual set of C "\" escape sequences with one exception: a backslash-newline-whitespace sequence is squashed into a single space character. To defeat this feature, put a further backslash at the start of the second line.

At the outermost level of the grammar, the input consists of a sequence of host and automount declarations. These declarations are all parsed before they are analyzed. This means they can appear in any order and cyclic host references are possible.

```
fsinfo      : list(fsinfo_attr) ;

fsinfo_attr : host | automount ;
```

8.4 *FSinfo* host definitions

A host declaration consists of three parts: a set of machine attribute data, a list of filesystems physically attached to the machine, and a list of additional statically mounted filesystems.

```
host        : "host" host_data list(filesystem) list(mount) ;
```

Each host must be declared in this way exactly once. Such things as the hardware address, the architecture and operating system types and the cluster name are all specified within the *host data*.

All the disks the machine has should then be described in the *list of filesystems*. When describing disks, you can specify what *volname* the disk/partition should have and all such entries are built up into a dictionary which can then be used for building the automounter maps.

The *list of mounts* specifies all the filesystems that should be statically mounted on the machine.

8.5 *FSinfo* host attributes

The host data, *host_data*, always includes the *hostname*. In addition, several other host attributes can be given.

```
host_data   : <hostname>
            | "{" list(host_attrs) "}" <hostname>
            ;
```

```
host_attrs  : host_attr "=" <string>
            | netif
            ;

host_attr   : "config"
            | "arch"
            | "os"
            | "cluster"
            ;
```

The *hostname* is, typically, the fully qualified hostname of the machine.

Examples:

```
host dylan.doc.ic.ac.uk

host {
    os = hpux
    arch = hp300
} dougal.doc.ic.ac.uk
```

The options that can be given as host attributes are shown below.

8.5.1 netif Option

This defines the set of network interfaces configured on the machine. The interface attributes collected by *FSinfo* are the IP address, subnet mask and hardware address. Multiple interfaces may be defined for hosts with several interfaces by an entry for each interface. The values given are sanity checked, but are currently unused for anything else.

```
netif       : "netif" <string> "{" list(netif_attrs) "}" ;

netif_attrs : netif_attr "=" <string> ;

netif_attr  : "inaddr" | "netmask" | "hwaddr" ;
```

Examples:

```
netif ie0 {
    inaddr  = 129.31.81.37
    netmask = 0xfffffe00
    hwaddr  = "08:00:20:01:a6:a5"
}

netif ec0 { }
```

8.5.2 config Option

This option allows you to specify configuration variables for the startup scripts (‘`rc`’ scripts). A simple string should immediately follow the keyword.

Example:

```
config "NFS_SERVER=true"
config "ZEPHYR=true"
```

This option is currently unsupported.

8.5.3 arch Option

This defines the architecture of the machine. For example:

```
arch = hp300
```

This is intended to be of use when building architecture specific mountmaps, however, the option is currently unsupported.

8.5.4 os Option

This defines the operating system type of the host. For example:

```
os = hpux
```

This information is used when creating the '`fstab`' files, for example in choosing which format to use for the '`fstab`' entries within the file.

8.5.5 cluster Option

This is used for specifying in which cluster the machine belongs. For example:

```
cluster = "theory"
```

The cluster is intended to be used when generating the automount maps, although it is currently unsupported.

8.6 *FSinfo* filesystems

The list of physically attached filesystems follows the machine attributes. These should define all the filesystems available from this machine, whether exported or not. In addition to the device name, filesystems have several attributes, such as filesystem type, mount options, and '`fsck`' pass number which are needed to generate '`fstab`' entries.

```
filesystem  : "fs" <device> "{" list(fs_data) "}" ;

fs_data     : fs_data_attr "=" <string>
            | mount
            ;

fs_data_attr
            : "fstype" | "opts" | "passno"
            | "freq" | "dumpset" | "log"
            ;
```

Here, <*device*> is the device name of the disk (for example, '`/dev/dsk/2s0`'). The device name is used for building the mount maps and for the '`fstab`' file. The attributes that can be specified are shown in the following section.

The *FSinfo* configuration file for `dylan.doc.ic.ac.uk` is listed below.

```
host dylan.doc.ic.ac.uk

fs /dev/dsk/0s0 {
        fstype = swap
}
```

```
     fs /dev/dsk/0s0 {
             fstype = hfs
             opts = rw,noquota,grpid
             passno = 0;
             freq = 1;
             mount / { }
     }

     fs /dev/dsk/1s0 {
             fstype = hfs
             opts = defaults
             passno = 1;
             freq = 1;
             mount /usr {
                     local {
                             exportfs "dougal eden dylan zebedee brian"
                             volname /nfs/hp300/local
                     }
             }
     }

     fs /dev/dsk/2s0 {
             fstype = hfs
             opts = defaults
             passno = 1;
             freq = 1;
             mount default {
                     exportfs "toytown_clients hangers_on"
                     volname /home/dylan/dk2
             }
     }

     fs /dev/dsk/3s0 {
             fstype = hfs
             opts = defaults
             passno = 1;
             freq = 1;
             mount default {
                     exportfs "toytown_clients hangers_on"
                     volname /home/dylan/dk3
             }
     }

     fs /dev/dsk/5s0 {
             fstype = hfs
             opts = defaults
```

```
                passno = 1;
                freq = 1;
                mount default {
                        exportfs "toytown_clients hangers_on"
                        volname /home/dylan/dk5
                }
        }
```

8.6.1 fstype Option

This specifies the type of filesystem being declared and will be placed into the '`fstab`' file as is. The value of this option will be handed to `mount` as the filesystem type—it should have such values as `4.2`, `nfs` or `swap`. The value is not examined for correctness.

There is one special case. If the filesystem type is specified as '`export`' then the filesystem information will not be added to the host's '`fstab`' information, but it will still be visible on the network. This is useful for defining hosts which contain referenced volumes but which are not under full control of *FSinfo*.

Example:

```
fstype = swap
```

8.6.2 opts Option

This defines any options that should be given to **mount**(8) in the '`fstab`' file. For example:

```
opts = rw,nosuid,grpid
```

8.6.3 passno Option

This defines the **fsck**(8) pass number in which to check the filesystem. This value will be placed into the '`fstab`' file.

Example:

```
passno = 1
```

8.6.4 freq Option

This defines the interval (in days) between dumps. The value is placed as is into the '`fstab`' file.

Example:

```
freq = 3
```

8.6.5 mount Option

This defines the mountpoint at which to place the filesystem. If the mountpoint of the filesystem is specified as `default`, then the filesystem will be mounted in the automounter's tree under its volume name and the mount will automatically be inherited by the automounter.

Following the mountpoint, namespace information for the filesystem may be described. The options that can be given here are `exportfs`, `volname` and `sel`.

The format is:

```
mount        : "mount" vol_tree ;

vol_tree     : list(vol_tree_attr) ;

vol_tree_attr
             :  <string> "{" list(vol_tree_info) vol_tree "}" ;

vol_tree_info
             : "exportfs" <export-data>
             | "volname" <volname>
             | "sel" <selector-list>
             ;
```

Example:

```
mount default {
    exportfs "dylan dougal florence zebedee"
    volname /vol/andrew
}
```

In the above example, the filesystem currently being declared will have an entry placed into the '`exports`' file allowing the filesystem to be exported to the machines `dylan`, `dougal`, `florence` and `zebedee`. The volume name by which the filesystem will be referred to remotely, is '`/vol/andrew`'. By declaring the mountpoint to be `default`, the filesystem will be mounted on the local machine in the automounter tree, where *Amd* will automatically inherit the mount as '`/vol/andrew`'.

'`exportfs`'
: a string defining which machines the filesystem may be exported to. This is copied, as is, into the '`exports`' file—no sanity checking is performed on this string.

'`volname`'
: a string which declares the remote name by which to reference the filesystem. The string is entered into a dictionary and allows you to refer to this filesystem in other places by this volume name.

'`sel`'
: a string which is placed into the automounter maps as a selector for the filesystem.

8.6.6 dumpset Option

This provides support for Imperial College's local file backup tools and is not documented further here.

8.6.7 log Option

Specifies the log device for the current filesystem. This is ignored if not required by the particular filesystem type.

8.7 *FSinfo* static mounts

Each host may also have a number of statically mounted filesystems. For example, the host may be a diskless workstation in which case it will have no `fs` declarations. In this case the `mount` declaration is used to determine from where its filesystems will be mounted. In addition to being added to the '`fstab`' file, this information can also be used to generate a suitable '`bootparams`' file.

```
mount       : "mount" <volname> list(localinfo) ;

localinfo   : localinfo_attr <string> ;

localinfo_attr
            : "as"
            | "from"
            | "fstype"
            | "opts"
            ;
```

The filesystem specified to be mounted will be searched for in the dictionary of volume names built when scanning the list of hosts' definitions.

The attributes have the following semantics:

'`from` *`machine`*'
: mount the filesystem from the machine with the hostname of *machine*.

'`as` *`mountpoint`*'
: mount the filesystem locally as the name given, in case this is different from the advertised volume name of the filesystem.

'`opts` *`options`*'
: native **mount**(8) options.

'`fstype` *`type`*'
: type of filesystem to be mounted.

An example:

```
mount /export/exec/hp300/local as /usr/local
```

If the mountpoint specified is either '`/`' or '`swap`', the machine will be considered to be booting off the net and this will be noted for use in generating a '`bootparams`' file for the host which owns the filesystems.

8.8 Defining an *Amd* Mount Map in *FSinfo*

The maps used by *Amd* can be constructed from *FSinfo* by defining all the automount trees. *FSinfo* takes all the definitions found and builds one map for each top level tree.

The automount tree is usually defined last. A single automount configuration will usually apply to an entire management domain. One `automount` declaration is needed for each *Amd* automount point. *FSinfo* determines whether the automount point is *direct* (see Section 5.19 [Direct Automount Filesystem], page 53) or *indirect* (see Section 5.22 [Top-level Filesystem], page 54). Direct automount points are distinguished by the fact that there is no underlying *automount_tree*.

```
automount   : "automount" opt(auto_opts) automount_tree ;
```

```
auto_opts   : "opts" <mount-options> ;

automount_tree
            : list(automount_attr)
            ;

automount_attr
            : <string> "=" <volname>
            | <string> "->" <symlink>
            | <string> "{" automount_tree "}"
            ;
```

If <*mount-options*> is given, then it is the string to be placed in the maps for *Amd* for the `opts` option.

A *map* is typically a tree of filesystems, for example '`home`' normally contains a tree of filesystems representing other machines in the network.

A map can either be given as a name representing an already defined volume name, or it can be a tree. A tree is represented by placing braces after the name. For example, to define a tree '`/vol`', the following map would be defined:

```
automount /vol { }
```

Within a tree, the only items that can appear are more maps. For example:

```
automount /vol {
    andrew { }
    X11 { }
}
```

In this case, *FSinfo* will look for volumes named '`/vol/andrew`' and '`/vol/X11`' and a map entry will be generated for each. If the volumes are defined more than once, then *FSinfo* will generate a series of alternate entries for them in the maps.

Instead of a tree, either a link (*name* `->` *destination*) or a reference can be specified (*name* `=` *destination*). A link creates a symbolic link to the string specified, without further processing the entry. A reference will examine the destination filesystem and optimize the reference. For example, to create an entry for `njw` in the '`/homes`' map, either of the two forms can be used:

```
automount /homes {
    njw -> /home/dylan/njw
}
```

or

```
automount /homes {
    njw = /home/dylan/njw
}
```

In the first example, when '`/homes/njw`' is referenced from *Amd*, a link will be created leading to '`/home/dylan/njw`' and the automounter will be referenced a second time to resolve this filename. The map entry would be:

```
njw type:=link;fs:=/home/dylan/njw
```

In the second example, the destination directory is analyzed and found to be in the filesystem '`/home/dylan`' which has previously been defined in the maps. Hence the map entry will look like:

```
njw rhost:=dylan;rfs:=/home/dylan;sublink:=njw
```

Creating only one symbolic link, and one access to *Amd*.

8.9 *FSinfo* Command Line Options

FSinfo is started from the command line by using the command:

```
fsinfo [options] files ...
```

The input to *FSinfo* is a single set of definitions of machines and automount maps. If multiple files are given on the command-line, then the files are concatenated together to form the input source. The files are passed individually through the C pre-processor before being parsed.

Several options define a prefix for the name of an output file. If the prefix is not specified no output of that type is produced. The suffix used will correspond either to the hostname to which a file belongs, or to the type of output if only one file is produced. Dumpsets and the '`bootparams`' file are in the latter class. To put the output into a subdirectory simply put a '`/`' at the end of the prefix, making sure that the directory has already been made before running *Fsinfo*.

8.9.1 `-a` *autodir*

Specifies the directory name in which to place the automounter's mountpoints. This defaults to '`/a`'. Some sites have the autodir set to be '`/amd`', and this would be achieved by:

```
fsinfo -a /amd ...
```

8.9.2 `-b` *bootparams*

This specifies the prefix for the '`bootparams`' filename. If it is not given, then the file will not be generated. The '`bootparams`' file will be constructed for the destination machine and will be placed into a file named '`bootparams`' and prefixed by this string. The file generated contains a list of entries describing each diskless client that can boot from the destination machine.

As an example, to create a '`bootparams`' file in the directory '`generic`', the following would be used:

```
fsinfo -b generic/ ...
```

8.9.3 `-d` *dumpsets*

This specifies the prefix for the '`dumpsets`' file. If it is not specified, then the file will not be generated. The file will be for the destination machine and will be placed into a filename '`dumpsets`', prefixed by this string. The '`dumpsets`' file is for use by Imperial College's local backup system.

For example, to create a '`dumpsets`' file in the directory '`generic`', then you would use the following:

```
fsinfo -d generic/ ...
```

8.9.4 `-e` *exportfs*

Defines the prefix for the '`exports`' files. If it is not given, then the file will not be generated. For each machine defined in the configuration files as having disks, an '`exports`' file is constructed and given a filename determined by the name of the machine, prefixed with this string. If a machine is defined as diskless, then no '`exports`' file will be created for it. The files contain entries for directories on the machine that may be exported to clients.

Example: To create the '`exports`' files for each diskfull machine and place them into the directory '`exports`':

```
fsinfo -e exports/ ...
```

8.9.5 `-f` *fstab*

This defines the prefix for the '`fstab`' files. The files will only be created if this prefix is defined. For each machine defined in the configuration files, a '`fstab`' file is created with the filename determined by prefixing this string with the name of the machine. These files contain entries for filesystems and partitions to mount at boot time.

Example, to create the files in the directory '`fstabs`':

```
fsinfo -f fstabs/ ...
```

8.9.6 `-h` *hostname*

Defines the hostname of the destination machine to process for. If this is not specified, it defaults to the local machine name, as returned by **gethostname**(2).

Example:

```
fsinfo -h dylan.doc.ic.ac.uk ...
```

8.9.7 `-m` *mount-maps*

Defines the prefix for the automounter files. The maps will only be produced if this prefix is defined. The mount maps suitable for the network defined by the configuration files will be placed into files with names calculated by prefixing this string to the name of each map.

For example, to create the automounter maps and place them in the directory '`automaps`':

```
fsinfo -m automaps/ ...
```

8.9.8 `-q`

Selects quiet mode. *FSinfo* suppress the "running commentary" and only outputs any error messages which are generated.

8.9.9 `-v`

Selects verbose mode. When this is activated, the program will display more messages, and display all the information discovered when performing the semantic analysis phase. Each verbose message is output to '`stdout`' on a line starting with a '`#`' character.

8.9.10 `-D` *name[=defn]*

Defines a symbol *name* for the preprocessor when reading the configuration files. Equivalent to `#define` directive.

8.9.11 `-I` *directory*

This option is passed into the preprocessor for the configuration files. It specifies directories in which to find include files

8.9.12 `-U` *name*

Removes any initial definition of the symbol *name*. Inverse of the `-D` option.

8.10 Errors produced by *FSinfo*

The following table documents the errors and warnings which *FSinfo* may produce.

`" expected`
: Occurs if an unescaped newline is found in a quoted string.

`ambiguous mount:` *`volume`* `is a replicated filesystem`
: If several filesystems are declared as having the same volume name, they will be considered replicated filesystems. To mount a replicated filesystem statically, a specific host will need to be named, to say which particular copy to try and mount, else this error will result.

`can't open` *`filename`* `for writing`
: Occurs if any errors are encountered when opening an output file.

`cannot determine localname since volname` *`volume`* `is not uniquely defined`
: If a volume is replicated and an attempt is made to mount the filesystem statically without specifying a local mountpoint, *FSinfo* cannot calculate a mountpoint, as the desired pathname would be ambiguous.

`device` `has duplicate exportfs data`
: Produced if the '`exportfs`' option is used multiple times within the same branch of a filesystem definition. For example, if you attempt to set the '`exportfs`' data at different levels of the mountpoint directory tree.

`dump frequency for` *`host`*`:`*`device`* `is non-zero`
: Occurs if *device* has its '`fstype`' declared to be '`swap`' or '`export`' and the '`dump`' option is set to a value greater than zero. Swap devices should not be dumped.

`duplicate host` *`hostname`*`!`
: If a host has more than one definition.

`end of file within comment`
: A comment was unterminated before the end of one of the configuration files.

`filename``: cannot open for reading`
: If a file specified on the command line as containing configuration data could not be opened.

`filesystem` `has a volname but no exportfs data`
: Occurs when a volume name is declared for a file system, but the string specifying what machines the filesystem can be exported to is missing.

`fs field "`*`field-name`*`" already set`
: Occurs when multiple definitions are given for one of the attributes of a host's filesystem.

`host field "`*`field-name`*`" already set`
: If duplicate definitions are given for any of the fields with a host definition.

`host``:`*`device`*` has more than one mount point`
: Occurs if the mount option for a host's filesystem specifies multiple trees at which to place the mountpoint.

`host``:`*`device`*` has no mount point`
: Occurs if the '`mount`' option is not specified for a host's filesystem.

`host``:`*`device`*` needs field "`*`field-name`*`"`
: Occurs when a filesystem is missing a required field. *field-name* could be one of '`fstype`', '`opts`', '`passno`' or '`mount`'.

`host``:mount field specified for swap partition`
: Occurs if a mountpoint is given for a filesystem whose type is declared to be '`swap`'.

`malformed IP dotted quad: `*`address`*
: If the Internet address of an interface is incorrectly specified. An Internet address definition is handled to **inet_addr**(3N) to see if it can cope. If not, then this message will be displayed.

`malformed netmask: `*`netmask`*
: If the netmask cannot be decoded as though it were a hexadecimal number, then this message will be displayed. It will typically be caused by incorrect characters in the *netmask* value.

`mount field "`*`field-name`*`" already set`
: Occurs when a static mount has multiple definitions of the same field.

`mount tree field "`*`field-name`*`" already set`
: Occurs when the *field-name* is defined more than once during the definition of a filesystems mountpoint.

`netif field `*`field-name`*` already set`
: Occurs if you attempt to define an attribute of an interface more than once.

`network booting requires both root and swap areas`
: Occurs if a machine has mount declarations for either the root partition or the swap area, but not both. You cannot define a machine to only partially boot via the network.

`no disk mounts on `*`hostname`*
: If there are no static mounts, nor local disk mounts specified for a machine, this message will be displayed.

`no volname given for `*`host`*`:`*`device`*
: Occurs when a filesystem is defined to be mounted on '`default`', but no volume name is given for the file system, then the mountpoint cannot be determined.

`not allowed '/' in a directory name`
: Occurs when a pathname with multiple directory elements is specified as the name for an automounter tree. A tree should only have one name at each level.

`pass number for` *`host`*`:`*`device`* `is non-zero`
: Occurs if *device* has its '`fstype`' declared to be '`swap`' or '`export`' and the **fsck**(8) pass number is set. Swap devices should not be fsck'd. See Section 8.6.1 [FSinfo fstype Option], page 80.

`sub-directory` *`directory`* `of` *`directory-tree`* `starts with '/'`
: Within the filesystem specification for a host, if an element *directory* of the mountpoint begins with a '`/`' and it is not the start of the tree.

`sub-directory of` *`directory-tree`* `is named "default"`
: '`default`' is a keyword used to specify if a mountpoint should be automatically calculated by *FSinfo*. If you attempt to specify a directory name as this, it will use the filename of '`default`' but will produce this warning.

`unknown \ sequence`
: Occurs if an unknown escape sequence is found inside a string. Within a string, you can give the standard C escape sequences for strings, such as newlines and tab characters.

`unknown directory attribute`
: If an unknown keyword is found while reading the definition of a host's filesystem mount option.

`unknown filesystem attribute`
: Occurs if an unrecognized keyword is used when defining a host's filesystems.

`unknown host attribute`
: Occurs if an unrecognized keyword is used when defining a host.

`unknown mount attribute`
: Occurs if an unrecognized keyword is found while parsing the list of static mounts.

`unknown volname` *`volume`* `automounted` *`[`* `on` *`name ]`*
: Occurs if *volume* is used in a definition of an automount map but the volume name has not been declared during the host filesystem definitions.

`volname` *`volume`* `is unknown`
: Occurs if an attempt is made to mount or reference a volume name which has not been declared during the host filesystem definitions.

`volname` *`volume`* `not exported from` *`machine`*
: Occurs if you attempt to mount the volume *volume* from a machine which has not declared itself to have such a filesystem available.

9 Hlfsd

Hlfsd is a daemon which implements a filesystem containing a symbolic link to subdirectory within a user's home directory, depending on the user which accessed that link. It was primarily designed to redirect incoming mail to users' home directories, so that it can be read from anywhere. It was designed and implemented by Erez Zadok (`http://www.cs.sunysb.edu/~ezk`) and Alexander Dupuy (`dupuy AT cs.columbia.edu`), at the Computer Science Department (`http://www.cs.columbia.edu/`) of Columbia University (`http://www.columbia.edu/`). A paper (`http://www.fsl.cs.sunysb.edu/docs/hlfsd/hlfsd.html`) on *Hlfsd* was presented at the Usenix LISA VII conference in 1993.

Hlfsd operates by mounting itself as an NFS server for the directory containing *linkname*, which defaults to '`/hlfs/home`'. Lookups within that directory are handled by *Hlfsd*, which uses the password map to determine how to resolve the lookup. The directory will be created if it doesn't already exist. The symbolic link will be to the accessing user's home directory, with *subdir* appended to it. If not specified, *subdir* defaults to '`.hlfsdir`'. This directory will also be created if it does not already exist.

A '`SIGTERM`' sent to *Hlfsd* will cause it to shutdown. A '`SIGHUP`' will flush the internal caches, and reload the password map. It will also close and reopen the log file, to enable the original log file to be removed or rotated. A '`SIGUSR1`' will cause it to dump its internal table of user IDs and home directories to the file '`/tmp/hlfsddump`'.

9.1 Introduction to Hlfsd

Electronic mail has become one of the major applications for many computer networks, and use of this service is expected to increase over time, as networks proliferate and become faster. Providing a convenient environment for users to read, compose, and send electronic mail has become a requirement for systems administrators (SAs).

Widely used methods for handling mail usually require users to be logged into a designated "home" machine, where their mailbox files reside. Only on that one machine can they read newly arrived mail. Since users have to be logged into that system to read their mail, they often find it convenient to run all of their other processes on that system as well, including memory and CPU-intensive jobs. For example, in our department, we have allocated and configured several multi-processor servers to handle such demanding CPU/memory applications, but these were underutilized, in large part due to the inconvenience of not being able to read mail on those machines. (No home directories were located on these designated CPU-servers, since we did not want NFS service for users' home directories to have to compete with CPU-intensive jobs. At the same time, we discouraged users from running demanding applications on their home machines.)

Many different solutions have been proposed to allow users to read their mail on any host. However, all of these solutions fail in one or more of several ways:

- they introduce new single points of failure
- they require using different mail transfer agents (MTAs) or user agents (UAs)
- they do not solve the problem for all cases, i.e. the solution is only partially successful for a particular environment.

We have designed a simple filesystem, called the *Home-Link File System*, to provide the ability to deliver mail to users' home directories, without modification to mail-related applications. We have endeavored to make it as stable as possible. Of great importance to us was to make sure the HLFS daemon, '`hlfsd`' , would not hang under any circumstances, and would take the next-best action when faced with problems. Compared to alternative methods, *Hlfsd* is a stable, more general solution, and easier to install/use. In fact, in some ways, we have even managed to improve the reliability and security of mail service.

Our server implements a small filesystem containing a symbolic link to a subdirectory of the invoking user's home directory, and named symbolic links to users' mailbox files.

The *Hlfsd* server finds out the *uid* of the process that is accessing its mount point, and resolves the pathname component '`home`' as a symbolic link to a subdirectory within the home directory given by the *uid*'s entry in the password file. If the *gid* of the process that attempts to access a mailbox file is a special one (called HLFS_GID), then the server maps the name of the *next* pathname component directly to the user's mailbox. This is necessary so that access to a mailbox file by users other than the owner can succeed. The server has safety features in case of failures such as hung filesystems or home directory filesystems that are inaccessible or full.

On most of our machines, mail gets delivered to the directory '`/var/spool/mail`'. Many programs, including UAs, depend on that path. *Hlfsd* creates a directory '`/mail`', and mounts itself on top of that directory. *Hlfsd* implements the path name component called '`home`', pointing to a subdirectory of the user's home directory. We have made '`/var/spool/mail`' a symbolic link to '`/mail/home`', so that accessing '`/var/spool/mail`' actually causes access to a subdirectory within a user's home directory.

The following table shows an example of how resolving the pathname '`/var/mail/`*`NAME`*' to '`/users/ezk/.mailspool/`*`NAME`*' proceeds.

Resolving Component	**Pathname left to resolve**	**Value if symbolic link**
`/`	`var/mail/`*NAME*	
`var/`	`mail/`*NAME*	
`mail@`	`/mail/home/`*NAME*	`mail@ -> /mail/home`
`/`	`mail/home/`*NAME*	
`mail/`	`home/`*NAME*	
`home@`	*NAME*	`home@ -> /users/ezk/.mailspool`
`/`	`users/ezk/.mailspool/`*NAME*	
`users/`	`ezk/.mailspool/`*NAME*	

`ezk/`	`.mailspool/`*NAME*
`.mailspool/`	*NAME*
NAME	

9.2 Background to Mail Delivery

This section provides an in-depth discussion of why available methods for delivering mail to home directories are not as good as the one used by *Hlfsd*.

9.2.1 Single-Host Mail Spool Directory

The most common method for mail delivery is for mail to be appended to a mailbox file in a standard spool directory on the designated "mail home" machine of the user. The greatest advantage of this method is that it is the default method most vendors provide with their systems, thus very little (if any) configuration is required on the SA's part. All they need to set up are mail aliases directing mail to the host on which the user's mailbox file is assigned. (Otherwise, mail is delivered locally, and users find mailboxes on many machines.)

As users become more sophisticated, and aided by windowing systems, they find themselves logging in on multiple hosts at once, performing several tasks concurrently. They ask to be able to read their mail on any host on the network, not just the one designated as their "mail home".

9.2.2 Centralized Mail Spool Directory

A popular method for providing mail readability from any host is to have all mail delivered to a mail spool directory on a designated "mail-server" which is exported via NFS to all of the hosts on the network. Configuring such a system is relatively easy. On most systems, the bulk of the work is a one-time addition to one or two configuration files in '`/etc`'. The file-server's spool directory is then hard-mounted across every machine on the local network. In small environments with only a handful of hosts this can be an acceptable solution. In our department, with a couple of hundred active hosts and thousands of mail messages processed daily, this was deemed completely unacceptable, as it introduced several types of problems:

Scalability and Performance

As more and more machines get added to the network, more mail traffic has to go over NFS to and from the mail-server. Users like to run mail-watchers, and read their mail often. The stress on the shared infrastructure increases with every user and host added; loads on the mail server would most certainly be high since all mail delivery goes through that one machine.[1] This leads to lower reliability and performance. To reduce the number of concurrent connections between clients and the server host, some SAs have resorted to automounting the mail-spool directory. But this solution only makes things worse: since users

[1] Delivery via NFS-mounted filesystems may require usage of '`rpc.lockd`' and '`rpc.statd`' to provide distributed file-locking, both of which are widely regarded as unstable and unreliable. Furthermore, this will degrade performance, as local processes as well as remote '`nfsd`' processes are kept busy.

often run mail watchers, and many popular applications such as '`trn`', '`emacs`', '`csh`' or '`ksh`' check periodically for new mail, the automounted directory would be effectively permanently mounted. If it gets unmounted automatically by the automounter program, it is most likely to get mounted shortly afterwards, consuming more I/O resources by the constant cycle of mount and umount calls.

Reliability

The mail-server host and its network connectivity must be very reliable. Worse, since the spool directory has to be hard-mounted,[2] many processes which access the spool directory (various shells, '`login`', '`emacs`', etc.) would be hung as long as connectivity to the mail-server is severed. To improve reliability, SAs may choose to backup the mail-server's spool partition several times a day. This may make things worse since reading or delivering mail while backups are in progress may cause backups to be inconsistent; more backups consume more backup-media resources, and increase the load on the mail-server host.

9.2.3 Distributed Mail Spool Service

Despite the existence of a few systems that support delivery to users' home directories, mail delivery to home directories hasn't caught on. We believe the main reason is that there are too many programs that "know" where mailbox files reside. Besides the obvious (the delivery program '`/bin/mail`' and mail readers like '`/usr/ucb/Mail`', '`mush`', '`mm`', etc.), other programs that know mailbox location are login, from, almost every shell, '`xbiff`', '`xmailbox`', and even some programs not directly related to mail, such as '`emacs`' and '`trn`'. Although some of these programs can be configured to look in different directories with the use of environment variables and other resources, many of them cannot. The overall porting work is significant.

Other methods that have yet to catch on require the use of a special mail-reading server, such as IMAP or POP. The main disadvantage of these systems is that UAs need to be modified to use these services — a long and involved task. That is why they are not popular at this time.

Several other ideas have been proposed and even used in various environments. None of them is robust. They are mostly very specialized, inflexible, and do not extend to the general case. Some of the ideas are plain bad, potentially leading to lost or corrupt mail:

automounters

Using an automounter such as *Amd* to provide a set of symbolic links from the normal spool directory to user home directories is not sufficient. UAs rename, unlink, and recreate the mailbox as a regular file, therefore it must be a real file, not a symbolic link. Furthermore, it must reside in a real directory which is writable by the UAs and MTAs. This method may also require populating '`/var/spool/mail`' with symbolic links and making sure they are updated. Making *Amd* manage that directory directly fails, since many various lock files need to be managed as well. Also, *Amd* does not provide all of the NFS op-

[2] No SA in their right minds would soft-mount read/write partitions — the chances for data loss are too great.

erations which are required to write mail such as write, create, remove, and unlink.

`$MAIL`

Setting this variable to an automounted directory pointing to the user's mail spool host only solves the problem for those programs which know and use `$MAIL`. Many programs don't, therefore this solution is partial and of limited flexibility. Also, it requires the SAs or the users to set it themselves — an added level of inconvenience and possible failures.

`/bin/mail`

Using a different mail delivery agent could be the solution. One such example is '`hdmail`'. However, '`hdmail`' still requires modifying all UAs, the MTA's configuration, installing new daemons, and changing login scripts. This makes the system less upgradable or compatible with others, and adds one more complicated system for SAs to deal with. It is not a complete solution because it still requires each user have their `$MAIL` variable setup correctly, and that every program use this variable.

9.2.4 Why Deliver Into the Home Directory?

There are several major reasons why SAs might want to deliver mail directly into the users' home directories:

Location

Many mail readers need to move mail from the spool directory to the user's home directory. It speeds up this operation if the two are on the same filesystem. If for some reason the user's home directory is inaccessible, it isn't that useful to be able to read mail, since there is no place to move it to. In some cases, trying to move mail to a non-existent or hung filesystem may result in mail loss.

Distribution

Having all mail spool directories spread among the many more filesystems minimizes the chances that complete environments will grind to a halt when a single server is down. It does increase the chance that there will be someone who is not able to read their mail when a machine is down, but that is usually preferred to having no one be able to read their mail because a centralized mail server is down. The problem of losing some mail due to the (presumably) higher chances that a user's machine is down is minimized in HLFS.

Security

Delivering mail to users' home directories has another advantage — enhanced security and privacy. Since a shared system mail spool directory has to be world-readable and searchable, any user can see whether other users have mail, when they last received new mail, or when they last read their mail. Programs such as '`finger`' display this information, which some consider an infringement of privacy. While it is possible to disable this feature of '`finger`' so that remote users cannot see a mailbox file's status, this doesn't prevent local users from getting the information. Furthermore, there are more programs which make use of this information. In shared environments, disabling such programs has to be done on a system-wide basis, but with mail delivered to users' home directories, users less concerned with privacy who do want to let others know when they last received or read mail can easily do so using file protection bits.

In summary, delivering mail to home directories provides users the functionality sought, and also avoids most of the problems just discussed.

9.3 Using Hlfsd

9.3.1 Controlling Hlfsd

Much the same way *Amd* is controlled by '`ctl-amd`', so does *Hlfsd* get controlled by the '`ctl-hlfsd`' script:

`ctl-hlfsd start`
: Start a new *Hlfsd*.

`ctl-hlfsd stop`
: Stop a running *Hlfsd*.

`ctl-hlfsd restart`
: Stop a running *Hlfsd*, wait for 10 seconds, and then start a new one. It is hoped that within 10 seconds, the previously running *Hlfsd* terminate properly; otherwise, starting a second one could cause system lockup.

For example, on our systems, we start *Hlfsd* within '`ctl-hlfsd`' as follows on Solaris 2 systems:

```
hlfsd -a /var/alt_mail -x all -l /var/log/hlfsd /mail/home .mailspool
```

The directory '`/var/alt_mail`' is a directory in the root partition where alternate mail will be delivered into, when it cannot be delivered into the user's home directory.

Normal mail gets delivered into '`/var/mail`', but on our systems, that is a symbolic link to '`/mail/home`'. '`/mail`' is managed by *Hlfsd*, which creates a dynamic symlink named '`home`', pointing to the subdirectory '`.mailspool`' *within* the accessing user's home directory. This results in mail which normally should go to '`/var/mail/$USER`', to go to '`$HOME/.mailspool/$USER`'.

Hlfsd does not create the '`/var/mail`' symlink. This needs to be created (manually) once on each host, by the system administrators, as follows:

```
mv /var/mail /var/alt_mail
ln -s /mail/home /var/mail
```

Hlfsd also responds to the following signals:

A '`SIGHUP`' signal sent to *Hlfsd* will force it to reload the password map immediately.

A '`SIGUSR1`' signal sent to *Hlfsd* will cause it to dump its internal password map to the file '`/usr/tmp/hlfsd.dump.XXXXXX`', where '`XXXXXX`' will be replaced by a random string generated by **mktemp**(3) or (the more secure) **mkstemp**(3).

9.3.2 Hlfsd Options

`-a` *`alt_dir`*
: Alternate directory. The name of the directory to which the symbolic link returned by *Hlfsd* will point, if it cannot access the home directory of the user.

This defaults to '`/var/hlfs`'. This directory will be created if it doesn't exist. It is expected that either users will read these files, or the system administrators will run a script to resend this "lost mail" to its owner.

`-c` *`cache-interval`*
: Caching interval. *Hlfsd* will cache the validity of home directories for this interval, in seconds. Entries which have been verified within the last *cache-interval* seconds will not be verified again, since the operation could be expensive, and the entries are most likely still valid. After the interval has expired, *Hlfsd* will re-verify the validity of the user's home directory, and reset the cache time-counter. The default value for *cache-interval* is 300 seconds (5 minutes).

`-f` Force fast startup. This option tells *Hlfsd* to skip startup-time consistency checks such as existence of mount directory, alternate spool directory, symlink to be hidden under the mount directory, their permissions and validity.

`-g` *`group`* Set the special group HLFS_GID to *group*. Programs such as '`/usr/ucb/from`' or '`/usr/sbin/in.comsat`', which access the mailboxes of other users, must be setgid '`HLFS_GID`' to work properly. The default group is '`hlfs`'. If no group is provided, and there is no group '`hlfs`', this feature is disabled.

`-h` Help. Print a brief help message, and exit.

`-i` *`reload-interval`*
: Map-reloading interval. Each *reload-interval* seconds, *Hlfsd* will reload the password map. *Hlfsd* needs the password map for the UIDs and home directory pathnames. *Hlfsd* schedules a '`SIGALRM`' to reload the password maps. A '`SIGHUP`' sent to *Hlfsd* will force it to reload the maps immediately. The default value for *reload-interval* is 900 seconds (15 minutes.)

`-l` *`logfile`*
: Specify a log file to which *Hlfsd* will record events. If *logfile* is the string '`syslog`' then the log messages will be sent to the system log daemon by **syslog**(3), using the '`LOG_DAEMON`' facility. This is also the default.

`-n` No verify. *Hlfsd* will not verify the validity of the symbolic link it will be returning, or that the user's home directory contains sufficient disk-space for spooling. This can speed up *Hlfsd* at the cost of possibly returning symbolic links to home directories which are not currently accessible or are full. By default, *Hlfsd* validates the symbolic-link in the background. The `-n` option overrides the meaning of the `-c` option, since no caching is necessary.

`-o` *`mount-options`*
: Mount options which *Hlfsd* will use to mount itself on top of *dirname*. By default, *mount-options* is set to '`ro`'. If the system supports symbolic-link caching, default options are set to '`ro,nocache`'.

`-p` Print PID. Outputs the process-id of *Hlfsd* to standard output where it can be saved into a file.

`-v` Version. Displays version information to standard error.

`-x` *`log-options`*
: Specify run-time logging options. The options are a comma separated list chosen from: '`fatal`', '`error`', '`user`', '`warn`', '`info`', '`map`', '`stats`', '`all`'.

`-C`
: Force *Hlfsd* to run on systems that cannot turn off the NFS attribute-cache. Use of this option on those systems is discouraged, as it may result in loss or misdelivery of mail. The option is ignored on systems that can turn off the attribute-cache.

`-D` *`log-options`*
: Select from a variety of debugging options. Prefixing an option with the string '`no`' reverses the effect of that option. Options are cumulative. The most useful option is '`all`'. Since this option is only used for debugging other options are not documented here. A fuller description is available in the program source.

`-P` *`password-file`*
: Read the user-name, user-id, and home directory information from the file *password-file*. Normally, *Hlfsd* will use **getpwent**(3) to read the password database. This option allows you to override the default database, and is useful if you want to map users' mail files to a directory other than their home directory. Only the username, uid, and home-directory fields of the file *password-file* are read and checked. All other fields are ignored. The file *password-file* must otherwise be compliant with Unix Version 7 colon-delimited format **passwd**(4).

9.3.3 Hlfsd Files

The following files are used by *Hlfsd*:

'`/hlfs`'
: directory under which *Hlfsd* mounts itself and manages the symbolic link '`home`'.

'`.hlfsdir`'
: default sub-directory in the user's home directory, to which the '`home`' symbolic link returned by *Hlfsd* points.

'`/var/hlfs`'
: directory to which '`home`' symbolic link returned by *Hlfsd* points if it is unable to verify the that user's home directory is accessible.

'`/usr/tmp/hlfsd.dump.XXXXXX`'
: file to which *Hlfsd* will dump its internal password map when it receives the '`SIGUSR1`' signal. '`XXXXXX`' will be replaced by a random string generated by **mktemp**(3) or (the more secure) **mkstemp**(3).

For discussion on other files used by *Hlfsd*, see See Section 10.12 [lostaltmail], page 99, and Section 10.13 [lostaltmail.conf-sample], page 99.

10 Assorted Tools

The following are additional utilities and scripts included with am-utils, and get installed.

10.1 am-eject

A shell script unmounts a floppy or CD-ROM that is automounted, and then attempts to eject the removable device.

10.2 amd.conf-sample

A sample *Amd* configuration file. See Chapter 6 [Amd Configuration File], page 55.

10.3 amd2ldif

A script to convert *Amd* maps to LDAP input files. Use it as follows:

```
amd2ldif mapname base < amd.mapfile > mapfile.ldif
```

10.4 amd2sun

A script to convert *Amd* maps to Sun Automounter maps. Use it as follows

```
amd2sun < amd.mapfile > auto_mapfile
```

10.5 automount2amd

A script to convert old Sun Automounter maps to *Amd* maps.

Say you have the Sun automount file *auto.foo*, with these two lines:

```
home                 earth:/home
moon  -ro,intr       server:/proj/images
```

Running

```
automount2amd auto.foo > amd.foo
```

will produce the *Amd* map *amd.foo* with this content:

```
# generated by automount2amd on Sat Aug 14 17:59:32 US/Eastern 1999

/defaults \\
  type:=nfs;opts:=rw,grpid,nosuid,utimeout=600

home \
  host==earth;type:=link;fs:=/home \\
  rhost:=earth;rfs:=/home

moon \
  -addopts:=ro,intr \\
  host==server;type:=link;fs:=/proj/images \\
  rhost:=server;rfs:=/proj/images
```

This perl script will use the following */default* entry

```
type:=nfs;opts:=rw,grpid,nosuid,utimeout=600
```

If you wish to override that, define the **$DEFAULTS** environment variable, or modify the script.

If you wish to generate Amd maps using the *hostd* (see Section 3.3.3.8 [hostd Selector Variable], page 27) *Amd* map syntax, then define the environment variable **$DOMAIN** or modify the script.

Note that automount2amd does not understand the syntax in newer Sun Automount maps, those used with autofs.

10.6 ctl-amd

A script to start, stop, or restart *Amd*. Use it as follows:

`ctl-amd start`
: Start a new *Amd* process.

`ctl-amd stop`
: Stop the running *Amd*.

`ctl-amd restart`
: Stop the running *Amd* (if any), safely wait for it to terminate, and then start a new process — only if the previous one died cleanly.

See Chapter 7 [Run-time Administration], page 69, for more details.

10.7 ctl-hlfsd

A script for controlling *Hlfsd*, much the same way '`ctl-amd`' controls *Amd*. Use it as follows:

`ctl-hlfsd start`
: Start a new *Hlfsd* process.

`ctl-hlfsd stop`
: Stop the running *Hlfsd*.

`ctl-hlfsd restart`
: Stop the running *Hlfsd* (if any), wait for 10 seconds for it to terminate, and then start a new process — only if the previous one died cleanly.

See Chapter 9 [Hlfsd], page 89, for more details.

10.8 expn

A script to expand email addresses into their full name. It is generally useful when using with the '`lostaltmail`' script, but is a useful tools otherwise.

```
$ expn -v ezk@example.com
ezk@example.com ->
        ezk@shekel.example.com
ezk@shekel.example.com ->
        Erez Zadok <"| /usr/local/mh/lib/slocal -user ezk || exit 75>
        Erez Zadok <\ezk>
        Erez Zadok </u/zing/ezk/.mailspool/backup>
```

10.9 fix-amd-map

Am-utils changed some of the syntax and default values of some variables. For example, the default value for '`${os}`' for Solaris 2.x (aka SunOS 5.x) systems used to be '`sos5`', it is now more automatically generated from '`config.guess`' and its value is '`sunos5`'.

This script converts older *Amd* maps to new ones. Use it as follows:

```
fix-amd-map < old.map > new.map
```

10.10 fixmount

'`fixmount`' is a variant of **showmount**(8) that can delete bogus mount entries in remote **mountd**(8) daemons. This is useful to cleanup otherwise ever-accumulating "junk". Use it for example:

```
fixmount -r host
```

See the online manual page for '`fixmount`' for more details of its usage.

10.11 fixrmtab

A script to invalidate '`/etc/rmtab`' entries for hosts named. Also restart mountd for changes to take effect. Use it for example:

```
fixrmtab host1 host2 ...
```

10.12 lostaltmail

A script used with *Hlfsd* to resend any "lost" mail. *Hlfsd* redirects mail which cannot be written into the user's home directory to an alternate directory. This is useful to continue delivering mail, even if the user's file system was unavailable, full, or over quota. But, the mail which gets delivered to the alternate directory needs to be resent to its respective users. This is what the '`lostaltmail`' script does.

Use it as follows:

```
lostaltmail
```

This script needs a configuration file '`lostaltmail.conf`' set up with the right parameters to properly work. See Chapter 9 [Hlfsd], page 89, for more details.

10.13 lostaltmail.conf-sample

This is a text file with configuration parameters needed for the '`lostaltmail`' script. The script includes comments explaining each of the configuration variables. See it for more information. Also see Chapter 9 [Hlfsd], page 89 for general information.

10.14 mk-amd-map

This program converts a normal *Amd* map file into an ndbm database with the same prefix as the named file. Use it as follows:

```
mk-amd-map mapname
```

10.15 pawd

Pawd is used to print the current working directory, adjusted to reflect proper paths that can be reused to go through the automounter for the shortest possible path. In particular, the path printed back does not include any of *Amd*'s local mount points. Using them is unsafe, because *Amd* may unmount managed file systems from the mount points, and thus including them in paths may not always find the files within.

Without any arguments, *Pawd* will print the automounter adjusted current working directory. With any number of arguments, it will print the adjusted path of each one of the arguments.

10.16 redhat-ctl-amd

This script is similar to *ctl-amd* (see Section 10.6 [ctl-amd], page 98) but is intended for Red Hat Linux systems. You can safely copy *redhat-ctl-amd* onto '`/etc/rc.d/init.d/amd`'. The script supplied by *Am-utils* is usually better than the one provided by Red Hat, because the Red Hat script does not correctly kill *Amd* processes: it is too quick to kill the wrong processes, leaving stale or hung mount points behind.

10.17 wait4amd

A script to wait for *Amd* to start on a particular host before performing an arbitrary command. The command is executed repeatedly, with 1 second intervals in between. You may interrupt the script using '`^C`' (or whatever keyboard sequence your terminal's '`intr`' function is bound to).

Examples:

`wait4amd saturn amq -p -h saturn`
: When *Amd* is up on host '`saturn`', get the process ID of that running *Amd*.

`wait4amd pluto rlogin pluto`
: Remote login to host '`pluto`' when *Amd* is up on that host. It is generally necessary to wait for *Amd* to properly start and initialize on a remote host before logging in to it, because otherwise user home directories may not be accessible across the network.

`wait4amd pluto`
: A short-hand version of the previous command, since the most useful reason for this script is to login to a remote host. I use it very often when testing out new versions of *Amd*, and need to reboot hung hosts.

10.18 wait4amd2die

This script is used internally by '`ctl-amd`' when used to restart *Amd*. It waits for *Amd* to terminate. If it detected that *Amd* terminated cleanly, this script will return an exist status of zero. Otherwise, it will return a non-zero exit status.

The script tests for *Amd*'s existence once every 5 seconds, six times, for a total of 30 seconds. It will return a zero exist status as soon as it detects that *Amd* dies.

10.19 wire-test

A simple program to test if some of the most basic networking functions in am-util's library '`libamu`' work. It also tests the combination of NFS protocol and version number that are supported from the current host, to a remote one.

For example, in this test a machine which only supports NFS Version 2 is contacting a remote host that can support the same version, but using both UDP and TCP. If no host name is specified, '`wire-test`' will try '`localhost`'.

```
$ wire-test moisil
Network name is "mcl-lab-net.cs.columbia.edu"
Network number is "128.59.13"
Network name is "old-net.cs.columbia.edu"
Network number is "128.59.16"
My IP address is 0x7f000001.
NFS Version and protocol tests to host "moisil"...
        testing vers=2, proto="udp" -> found version 2.
        testing vers=3, proto="udp" -> failed!
        testing vers=2, proto="tcp" -> found version 2.
        testing vers=3, proto="tcp" -> failed!
```

11 Examples

11.1 User Filesystems

With more than one fileserver, the directories most frequently cross-mounted are those containing user home directories. A common convention used at Imperial College is to mount the user disks under `/home/`*machine*.

Typically, the '`/etc/fstab`' file contained a long list of entries such as:

```
machine:/home/machine /home/machine nfs ...
```

for each fileserver on the network.

There are numerous problems with this system. The mount list can become quite large and some of the machines may be down when a system is booted. When a new fileserver is installed, '`/etc/fstab`' must be updated on every machine, the mount directory created and the filesystem mounted.

In many environments most people use the same few workstations, but it is convenient to go to a colleague's machine and access your own files. When a server goes down, it can cause a process on a client machine to hang. By minimizing the mounted filesystems to only include those actively being used, there is less chance that a filesystem will be mounted when a server goes down.

The following is a short extract from a map taken from a research fileserver at Imperial College.

Note the entry for '`localhost`' which is used for users such as the operator ('`opr`') who have a home directory on most machine as '`/home/localhost/opr`'.

```
/defaults       opts:=rw,intr,grpid,nosuid
charm           host!=${key};type:=nfs;rhost:=${key};rfs:=/home/${key} \
                host==${key};type:=ufs;dev:=/dev/xd0g
#
...

#
localhost       type:=link;fs:=${host}
...
#
# dylan has two user disks so have a
# top directory in which to mount them.
#
dylan           type:=auto;fs:=${map};pref:=${key}/
#
dylan/dk2       host!=dylan;type:=nfs;rhost:=dylan;rfs:=/home/${key} \
                host==dylan;type:=ufs;dev:=/dev/dsk/2s0
#
dylan/dk5       host!=dylan;type:=nfs;rhost:=dylan;rfs:=/home/${key} \
                host==dylan;type:=ufs;dev:=/dev/dsk/5s0
...
#
```

```
toytown         host!=${key};type:=nfs;rhost:=${key};rfs:=/home/${key} \
                host==${key};type:=ufs;dev:=/dev/xy1g
...
#
zebedee         host!=${key};type:=nfs;rhost:=${key};rfs:=/home/${key} \
                host==${key};type:=ufs;dev:=/dev/dsk/1s0
#
# Just for access...
#
gould           type:=auto;fs:=${map};pref:=${key}/
gould/staff     host!=gould;type:=nfs;rhost:=gould;rfs:=/home/${key}
#
gummo           host!=${key};type:=nfs;rhost:=${key};rfs:=/home/${key}
...
```

This map is shared by most of the machines listed so on those systems any of the user disks is accessible via a consistent name. *Amd* is started with the following command

```
amd /home amd.home
```

Note that when mounting a remote filesystem, the *automounted* mount point is referenced, so that the filesystem will be mounted if it is not yet (at the time the remote 'mountd' obtains the file handle).

11.2 Home Directories

One convention for home directories is to locate them in '/homes' so user 'jsp''s home directory is '/homes/jsp'. With more than a single fileserver it is convenient to spread user files across several machines. All that is required is a mount-map which converts login names to an automounted directory.

Such a map might be started by the command:

```
amd /homes amd.homes
```

where the map 'amd.homes' contained the entries:

```
/defaults   type:=link   # All the entries are of type:=link
jsp         fs:=/home/charm/jsp
njw         fs:=/home/dylan/dk5/njw
...
phjk        fs:=/home/toytown/ai/phjk
sjv         fs:=/home/ganymede/sjv
```

Whenever a login name is accessed in '/homes' a symbolic link appears pointing to the real location of that user's home directory. In this example, '/homes/jsp' would appear to be a symbolic link pointing to '/home/charm/jsp'. Of course, '/home' would also be an automount point.

This system causes an extra level of symbolic links to be used. Although that turns out to be relatively inexpensive, an alternative is to directly mount the required filesystems in the '/homes' map. The required map is simple, but long, and its creation is best automated. The entry for 'jsp' could be:

```
jsp   -sublink:=${key};rfs:=/home/charm \
                     host==charm;type:=ufs;dev:=/dev/xd0g \
                     host!=charm;type:=nfs;rhost:=charm
```

This map can become quite big if it contains a large number of entries. By combining two other features of *Amd* it can be greatly simplified.

First the UFS partitions should be mounted under the control of '`/etc/fstab`', taking care that they are mounted in the same place that *Amd* would have automounted them. In most cases this would be something like '`/a/host/home/host`' and '`/etc/fstab`' on host '`charm`' would have a line:

```
/dev/xy0g /a/charm/home/charm 4.2 rw,nosuid,grpid 1 5
```

The map can then be changed to:

```
/defaults    type:=nfs;sublink:=${key};opts:=rw,intr,nosuid,grpid
jsp          rhost:=charm;rfs:=/home/charm
njw          rhost:=dylan;rfs:=/home/dylan/dk5
...
phjk         rhost:=toytown;rfs:=/home/toytown;sublink:=ai/${key}
sjv          rhost:=ganymede;rfs:=/home/ganymede
```

This map operates as usual on a remote machine (*ie* `${host}` not equal to `${rhost}`). On the machine where the filesystem is stored (*ie* `${host}` equal to `${rhost}`), *Amd* will construct a local filesystem mount point which corresponds to the name of the locally mounted UFS partition. If *Amd* is started with the `-r` option then instead of attempting an NFS mount, *Amd* will simply inherit the UFS mount (see Section 5.24 [Inheritance Filesystem], page 54). If `-r` is not used then a loopback NFS mount will be made. This type of mount is known to cause a deadlock on many systems.

11.3 Architecture Sharing

Often a filesystem will be shared by machines of different architectures. Separate trees can be maintained for the executable images for each architecture, but it may be more convenient to have a shared tree, with distinct subdirectories.

A shared tree might have the following structure on the fileserver (called '`fserver`' in the example):

```
local/tex
local/tex/fonts
local/tex/lib
local/tex/bin
local/tex/bin/sun3
local/tex/bin/sun4
local/tex/bin/hp9000
...
```

In this example, the subdirectories of '`local/tex/bin`' should be hidden when accessed via the automount point (conventionally '`/vol`'). A mount-map for '`/vol`' to achieve this would look like:

```
/defaults   sublink:=${/key};rhost:=fserver;type:=link
tex         type:=auto;fs:=${map};pref:=${key}/
tex/fonts   host!=fserver;type:=nfs;rfs:=/vol/tex \
            host==fserver;fs:=/usr/local/tex
tex/lib         host!=fserver;type:=nfs;rfs:=/vol/tex \
            host==fserver;fs:=/usr/local/tex
tex/bin         -sublink:=${/key}/${arch} \
            host!=fserver;type:=nfs;rfs:=/vol/tex \
            host:=fserver;fs:=/usr/local/tex
```

When '`/vol/tex/bin`' is referenced, the current machine architecture is automatically appended to the path by the `${sublink}` variable. This means that users can have '`/vol/tex/bin`' in their '`PATH`' without concern for architecture dependencies.

11.4 Wildcard Names & Replicated Servers

By using the wildcard facility, *Amd* can *overlay* an existing directory with additional entries. The system files are usually mounted under '`/usr`'. If instead, *Amd* is mounted on '`/usr`', additional names can be overlayed to augment or replace names in the "master" '`/usr`'. A map to do this would have the form:

```
local  type:=auto;fs:=local-map
share  type:=auto;fs:=share-map
*      -type:=nfs;rfs:=/export/exec/${arch};sublink:="${key}" \
        rhost:=fserv1  rhost:=fserv2  rhost:=fserv3
```

Note that the assignment to `${sublink}` is surrounded by double quotes to prevent the incoming key from causing the map to be misinterpreted. This map has the effect of directing any access to '`/usr/local`' or '`/usr/share`' to another automount point.

In this example, it is assumed that the '`/usr`' files are replicated on three fileservers: '`fserv1`', '`fserv2`' and '`fserv3`'. For any references other than to '`local`' and '`share`' one of the servers is used and a symbolic link to `${autodir}/${rhost}/export/exec/${arch}/`*`whatever`* is returned once an appropriate filesystem has been mounted.

11.5 '`rwho`' servers

The '`/usr/spool/rwho`' directory is a good candidate for automounting. For efficiency reasons it is best to capture the rwho data on a small number of machines and then mount that information onto a large number of clients. The data written into the rwho files is byte order dependent so only servers with the correct byte ordering can be used by a client:

```
/defaults         type:=nfs
usr/spool/rwho    -byte==little;rfs:=/usr/spool/rwho \
                      rhost:=vaxA  rhost:=vaxB \
                  || -rfs:=/usr/spool/rwho \
                      rhost:=sun4  rhost:=hp300
```

11.6 '`/vol`'

'`/vol`' is used as a catch-all for volumes which do not have other conventional names.

Below is part of the '`/vol`' map for the domain '`doc.ic.ac.uk`'. The '`r+d`' tree is used for new or experimental software that needs to be available everywhere without installing it on all the fileservers. Users wishing to try out the new software then simply include '`/vol/r+d/{bin,ucb}`' in their path.

The main tree resides on one host '`gould.doc.ic.ac.uk`', which has different '`bin`', '`etc`', '`lib`' and '`ucb`' sub-directories for each machine architecture. For example, '`/vol/r+d/bin`' for a Sun-4 would be stored in the sub-directory '`bin/sun4`' of the filesystem '`/usr/r+d`'. When it was accessed a symbolic link pointing to '`/a/gould/usr/r+d/bin/sun4`' would be returned.

```
/defaults     type:=nfs;opts:=rw,grpid,nosuid,intr,soft
wp            -opts:=rw,grpid,nosuid;rhost:=charm \
              host==charm;type:=link;fs:=/usr/local/wp \
              host!=charm;type:=nfs;rfs:=/vol/wp
...
#
src           -opts:=rw,grpid,nosuid;rhost:=charm \
              host==charm;type:=link;fs:=/usr/src \
              host!=charm;type:=nfs;rfs:=/vol/src
#
r+d           type:=auto;fs:=${map};pref:=r+d/
# per architecture bin,etc,lib&ucb...
r+d/bin       rhost:=gould.doc.ic.ac.uk;rfs:=/usr/r+d;sublink:=${/key}/${arch}
r+d/etc       rhost:=gould.doc.ic.ac.uk;rfs:=/usr/r+d;sublink:=${/key}/${arch}
r+d/include   rhost:=gould.doc.ic.ac.uk;rfs:=/usr/r+d;sublink:=${/key}
r+d/lib       rhost:=gould.doc.ic.ac.uk;rfs:=/usr/r+d;sublink:=${/key}/${arch}
r+d/man       rhost:=gould.doc.ic.ac.uk;rfs:=/usr/r+d;sublink:=${/key}
r+d/src       rhost:=gould.doc.ic.ac.uk;rfs:=/usr/r+d;sublink:=${/key}
r+d/ucb       rhost:=gould.doc.ic.ac.uk;rfs:=/usr/r+d;sublink:=${/key}/${arch}
# hades pictures
pictures      -opts:=rw,grpid,nosuid;rhost:=thpfs \
              host==thpfs;type:=link;fs:=/nbsd/pictures \
              host!=thpfs;type:=nfs;rfs:=/nbsd;sublink:=pictures
# hades tools
hades         -opts:=rw,grpid,nosuid;rhost:=thpfs \
              host==thpfs;type:=link;fs:=/nbsd/hades \
              host!=thpfs;type:=nfs;rfs:=/nbsd;sublink:=hades
# bsd tools for hp.
bsd           -opts:=rw,grpid,nosuid;arch==hp9000;rhost:=thpfs \
              host==thpfs;type:=link;fs:=/nbsd/bsd \
              host!=thpfs;type:=nfs;rfs:=/nbsd;sublink:=bsd
```

11.7 '`/defaults`' with selectors

It is sometimes useful to have different defaults for a given map. To achieve this, the '`/defaults`' entry must be able to process normal selectors. This feature is turned on by setting '`selectors_in_defaults = yes`' in the '`amd.conf`' file. See Section 6.4.8 [selectors_in_defaults Parameter], page 57.

In this example, I set different default NFS mount options for hosts which are running over a slower network link. By setting a smaller size for the NFS read and write buffer sizes, you can greatly improve remote file service performance.

```
/defaults \
  wire==slip-net;opts:=rw,intr,rsize=1024,wsize=1024,timeo=20,retrans=10 \
  wire!=slip-net;opts:=rw,intr
```

11.8 '/tftpboot' in a chroot-ed environment

In this complex example, we attempt to run an *Amd* process *inside* a chroot-ed environment. '`tftpd`' (Trivial FTP) is used to trivially retrieve files used to boot X-Terminals, Network Printers, Network routers, diskless workstations, and other such devices. For security reasons, '`tftpd`' (and also '`ftpd`') processes are run using the **chroot**(2) system call. This provides an environment for these processes, where access to any files outside the directory where the chroot-ed process runs is denied.

For example, if you start '`tftpd`' on your system with

```
chroot /tftpboot /usr/sbin/tftpd
```

then the '`tftpd`' process will not be able to access any files outside '`/tftpboot`'. This ensures that no one can retrieve files such as '`/etc/passwd`' and run password crackers on it.

Since the TFTP service works by broadcast, it is necessary to have at least one TFTP server running on each subnet. If you have lots of files that you need to make available for '`tftp`', and many subnets, it could take significant amounts of disk space on each host serving them.

A solution we implemented at Columbia University was to have every host run '`tftpd`', but have those servers retrieve the boot files from two replicated servers. Those replicated servers have special partitions dedicated to the many network boot files.

We start *Amd* as follows:

```
amd /tftpboot/.amd amd.tftpboot
```

That is, *Amd* is serving the directory '`/tftpboot/.amd`'. The '`tftp`' server runs inside '`/tftpboot`' and is chroot-ed in that directory too. The '`amd.tftpboot`' map looks like:

```
#
# Amd /tftpboot directory -> host map
#

/defaults  opts:=nosuid,ro,intr,soft;fs:=/tftpboot/import;type:=nfs

tp         host==lol;rfs:=/n/lol/import/tftpboot;type:=lofs \
           host==ober;rfs:=/n/ober/misc/win/tftpboot;type:=lofs \
           rhost:=ober;rfs:=/n/ober/misc/win/tftpboot \
           rhost:=lol;rfs:=/n/lol/import/tftpboot
```

To help understand this example, I list a few of the file entries that are created inside '`/tftpboot`':

```
$ ls -la /tftpboot
dr-xr-xr-x   2 root   512 Aug 30 23:11 .amd
```

```
    drwxrwsr-x  12 root   512 Aug 30 08:00 import
    lrwxrwxrwx   1 root    33 Feb 27  1997 adminpr.cfg -> ./.amd/tp/hplj/adminpr.cfg
    lrwxrwxrwx   1 root    22 Dec  5  1996 tekxp -> ./.amd/tp/xterms/tekxp
    lrwxrwxrwx   1 root     1 Dec  5  1996 tftpboot -> .
```

Here is an explanation of each of the entries listed above:

`.amd` This is the *Amd* mount point. Note that you do not need to run a separate *Amd* process for the TFTP service. The **chroot**(2) system call only protects against file access, but the same process can still serve files and directories inside and outside the chroot-ed environment, because *Amd* itself was not run in chroot-ed mode.

`import` This is the mount point where *Amd* will mount the directories containing the boot files. The map is designed so that remote directories will be NFS mounted (even if they are already mounted elsewhere), and local directories are loopback mounted (since they are not accessible outside the chroot-ed '`/tftpboot`' directory).

`adminpr.cfg`
`tekxp` Two manually created symbolic links to directories *inside* the *Amd*-managed directory. The crossing of the component '`tp`' will cause *Amd* to automount one of the remote replicas. Once crossed, access to files inside proceeds as usual. The '`adminpr.cfg`' is a configuration file for an HP Laser-Jet 4si printer, and the '`tekxp`' is a directory for Tektronix X-Terminal boot files.

`tftpboot` This innocent looking symlink is important. Usually, when devices boot via the TFTP service, they perform the '`get file`' command to retrieve *file*. However, some devices assume that '`tftpd`' does not run in a chroot-ed environment, but rather "unprotected", and thus use a full pathname for files to retrieve, as in '`get /tftpboot/file`'. This symlink effectively strips out the leading '`/tftpboot/`'.

12 Internals

Note that there are more error and logging messages possible than are listed here. Most of them are self-explanatory. Refer to the program sources for more details on the rest.

12.1 Log Messages

In the following sections a brief explanation is given of some of the log messages made by *Amd*. Where the message is in '`typewriter`' font, it corresponds exactly to the message produced by *Amd*. Words in *italic* are replaced by an appropriate string. Variables, `${var}`, indicate that the value of the appropriate variable is output.

Log messages are either sent directly to a file, or logged via the **syslog**(3) mechanism. See Section 6.5.23 [log_file Parameter], page 61. In either case, entries in the file are of the form:

```
date-string  hostname amd[pid]  message
```

12.1.1 Fatal errors

Amd attempts to deal with unusual events. Whenever it is not possible to deal with such an error, *Amd* will log an appropriate message and, if it cannot possibly continue, will either exit or abort. These messages are selected by '`-x fatal`' on the command line. When **syslog**(3) is being used, they are logged with level '`LOG_FATAL`'. Even if *Amd* continues to operate it is likely to remain in a precarious state and should be restarted at the earliest opportunity.

`Attempting to inherit not-a-filesystem`
: The prototype mount point created during a filesystem restart did not contain a reference to the restarted filesystem. This error "should never happen".

`Can't bind to domain "`*`NIS-domain`*`"`
: A specific NIS domain was requested on the command line, but no server for that domain is available on the local net.

`Can't determine IP address of this host (`*`hostname`*`)`
: When *Amd* starts it determines its own IP address. If this lookup fails then *Amd* cannot continue. The hostname it looks up is that obtained returned by **gethostname**(2) system call.

`Can't find root file handle for `*`automount point`*
: *Amd* creates its own file handles for the automount points. When it mounts itself as a server, it must pass these file handles to the local kernel. If the filehandle is not obtainable the mount point is ignored. This error "should never happen".

`Must be root to mount filesystems (euid = `*`euid`*`)`
: To prevent embarrassment, *Amd* makes sure it has appropriate system privileges. This amounts to having an euid of 0. The check is made after argument processing complete to give non-root users a chance to access the `-v` option.

`No work to do - quitting`
: No automount points were given on the command line and so there is no work to do.

`Out of memory`
: While attempting to malloc some memory, the memory space available to *Amd* was exhausted. This is an unrecoverable error.

`Out of memory in realloc`
: While attempting to realloc some memory, the memory space available to *Amd* was exhausted. This is an unrecoverable error.

`cannot create rpc/udp service`
: Either the NFS or AMQ endpoint could not be created.

`gethostname: *description*`
: The **gethostname**(2) system call failed during startup.

`host name is not set`
: The **gethostname**(2) system call returned a zero length host name. This can happen if *Amd* is started in single user mode just after booting the system.

`ifs_match called!`
: An internal error occurred while restarting a pre-mounted filesystem. This error "should never happen".

`mount_afs: *description*`
: An error occurred while *Amd* was mounting itself.

`run_rpc failed`
: Somehow the main NFS server loop failed. This error "should never happen".

`unable to free rpc arguments in amqprog_1`
: The incoming arguments to the AMQ server could not be free'ed.

`unable to free rpc arguments in nfs_program_1`
: The incoming arguments to the NFS server could not be free'ed.

`unable to register (AMQ_PROGRAM, AMQ_VERSION, udp)`
: The AMQ server could not be registered with the local portmapper or the internal RPC dispatcher.

`unable to register (NFS_PROGRAM, NFS_VERSION, 0)`
: The NFS server could not be registered with the internal RPC dispatcher.

XXX: This section needs to be updated

12.1.2 Info messages

Amd generates information messages to record state changes. These messages are selected by '`-x info`' on the command line. When **syslog**(3) is being used, they are logged with level '`LOG_INFO`'.

The messages listed below can be generated and are in a format suitable for simple statistical analysis. *mount-info* is the string that is displayed by *Amq* in its mount information column and placed in the system mount table.

`"${*path*}" forcibly timed out`
: An automount point has been timed out by the *Amq* command.

`"${`*`path`*`}" has timed out`
: No access to the automount point has been made within the timeout period.

`Filehandle denied for "${`*`rhost`*`}:${`*`rfs`*`}"`
: The mount daemon refused to return a file handle for the requested filesystem.

`Filehandle error for "${`*`rhost`*`}:${`*`rfs`*`}": `*`description`*
: The mount daemon gave some other error for the requested filesystem.

`Finishing with status `*`exit-status`*
: *Amd* is about to exit with the given exit status.

`Re-synchronizing cache for map ${`*`map`*`}`
: The named map has been modified and the internal cache is being re-synchronized.

`file server ${`*`rhost`*`} is down - timeout of "${`*`path`*`}" ignored`
: An automount point has timed out, but the corresponding file server is known to be down. This message is only produced once for each mount point for which the server is down.

`file server ${`*`rhost`*`} type nfs is down`
: An NFS file server that was previously up is now down.

`file server ${`*`rhost`*`} type nfs is up`
: An NFS file server that was previously down is now up.

`file server ${`*`rhost`*`} type nfs starts down`
: A new NFS file server has been referenced and is known to be down.

`file server ${`*`rhost`*`} type nfs starts up`
: A new NFS file server has been referenced and is known to be up.

`mount of "${`*`path`*`}" on ${`*`fs`*`} timed out`
: Attempts to mount a filesystem for the given automount point have failed to complete within 30 seconds.

`mount-info`` mounted fstype ${`*`type`*`} on ${`*`fs`*`}`
: A new file system has been mounted.

`mount-info`` restarted fstype ${`*`type`*`} on ${`*`fs`*`}`
: *Amd* is using a pre-mounted filesystem to satisfy a mount request.

`mount-info`` unmounted fstype ${`*`type`*`} from ${`*`fs`*`}`
: A file system has been unmounted.

`mount-info`` unmounted fstype ${`*`type`*`} from ${`*`fs`*`} link ${`*`fs`*`}/${`*`sublink`*`}`
: A file system of which only a sub-directory was in use has been unmounted.

`restarting `*`mount-info`*` on ${`*`fs`*`}`
: A pre-mounted file system has been noted.

XXX: This section needs to be updated

Acknowledgments & Trademarks

Many thanks to the Am-Utils Users mailing list through the months developing am-utils. These members have contributed to the discussions, ideas, code and documentation, and subjected their systems to alpha quality code. Special thanks go to those authors (`http://www.am-utils.org/docs/am-utils/AUTHORS.txt`) who have submitted patches, and especially to the maintainers:

- Erez Zadok (`http://www.cs.sunysb.edu/~ezk`)
- Ion Badulescu (`ionut AT badula.org`)
- Rainer Orth (`ro AT techfak.uni-bielefeld.de`)
- Nick Williams (`nick.williams AT morganstanley.com`)

Thanks to the Formal Methods Group at Imperial College for suffering patiently while *Amd* was being developed on their machines.

Thanks to the many people who have helped with the development of *Amd*, especially Piete Brooks at the Cambridge University Computing Lab for many hours of testing, experimentation and discussion.

Thanks to the older Amd Workers (`amd-workers AT majordomo.glue.umd.edu`) mailing list (now defunct) members for many suggestions and bug reports to *Amd*.

- **DEC**, **VAX** and **Ultrix** are registered trademarks of Digital Equipment Corporation.
- **AIX** and **IBM** are registered trademarks of International Business Machines Corporation.
- **Sun**, **NFS** and **SunOS** are registered trademarks of Sun Microsystems, Inc.
- **UNIX** is a registered trademark in the USA and other countries, exclusively licensed through X/Open Company, Ltd.
- All other registered trademarks are owned by their respective owners.

Index

Table of Contents