Automount Implementation Guide:

Best Practices, Tuning, and Troubleshooting Tips

Bikash R. Choudhury, Network Appliance, Inc.

May 2007 | TR-3591

Abstract

This paper mainly describes briefly how an automounter can be implemeted and configured for clients in different environments. This document provides better understanding about the internal mechanism that happens in the background, some best practice recommendations, and troubleshooting tips regarding automounter. Network Appliance, Inc. does not take responsibility to troubleshoot client-side automounter issues, but the intent of this paper is to communicate how to set up and use automounters optimally in various environments as needed.

Contents

Automount

To use `/etc/fstab`, regardless of how infrequently a user may access the NFS mounted file system, the system must dedicate resources to keep that mount in place. Perhaps many hundreds or thousands of clients will impact the startup time, if they are mounted sequentially. The automounter is very useful as a centralized administration tool more than as a solution to a performance problem. You can always use **bg** as a mount option in the vfstab/fstab or manually to mount in the background, but the user may log in before the mount is in place. In this case an automount solution is preferred. An automounter provides the ability to manage mount tables centrally, automatically mounting entries on demand and unmounting them after a predefined period of inactivity. In addition to the reduction in administrative overhead, an automounter provides a dramatic reduction in the resources needed to have a significant number of file systems available on demand from an arbitrary number of network servers. However, this only applies to situations where the overall number of file systems actively used is smaller than the potential number of file systems that can be mounted. The automount/unmount will happen periodically, and that is fairly expensive. Automounters use "maps" which define the file system to mount. Automounters can get their mounting information from centralized “maps,” which can be flat files, or even NIS maps or sections of an LDAP directory. In addition, it supports nested mapping, and supports wildcard variables. Automount operates during two stages: Startup time and mount time. When automount starts, it forks a daemon to serve the mountpoints specified in the maps. It does this by establishing the daemon as the NFS server for the specified mountpoints; in effect, it mounts the daemon at these mountpoints.

An automounter provides four key features:

- An automounter makes exported file systems on all servers listed in /etc/hosts available to your local host machine or /net/<ip-address> in automaster map.
- Wildcarding in the automounter makes it easy to mount remote directories to like-named mountpoints on the local host machine.
- The automounting features of the programs mount only the exported file systems that are used from the possibly huge list of servers.
- Having multiple servers improves the reliability of the servers and the network by removing the dependency on the network. This can be done from the command line, but automounter makes it more convenient.

Types of Automounters

There are two types of automounters in Linux® and Solaris: *AMD* and *autof***s**. AMD is the automount daemon, and is meant to work like the old Sun™ OS AMD. The biggest difference between AMD and autofs is where their code lives.

AMD is implemented in user space, meaning it's not part of the kernel. It works like a lot of other networked applications using RPC (Remote Procedure Call). This should make the application's inner workings easier to understand for those who use other RPC-enabled applications. The major drawback is that the RPC calls go to the kernel, where file system code lives. Every time it makes a file system call to the kernel, it incurs a context switch, where the duties involved in carrying out an operation pass from the user space to the kernel space. This is a somewhat costly proposition for something that could potentially be doing a hundred or more operations per second.

Autofs is a newer system that includes both a user-space daemon and code in the kernel that assists the daemon. This means that the kernel's file system code knows where the automount mountpoints are using “maps” on an otherwise normal underlying file system, and the automount program takes it from there. There are currently three versions of autofs: v3, v4, and v5. Behavior of autofsv3/v4 is similar to that of AMD and very old. Autofs5 is currently shipping with RHEL 5 [2.6.18+ kernel] and Solaris™ 10. Apart from the performance boost associated with being in kernel space, autofs also enables administrators to perform mounts for directories that are mounted under the mountpoints used directly by the users.

Usually, autofs is launched at boot time and is closed at shutdown (or reboot) time. However, the administrator is able to start or stop it "manually.”

Autofs can have four different options:

1. *start*: as its name suggests, it starts the process. When starting, autofs will parse the nsswitch.conf file [automount: files NIS]. If the entry contains ‘files’ first, it will search for a /etc/auto.master file for "maps" specifying mountpoints. Then it installs automount for every mountpoint. In Solaris it mounts an autofs file system on the mountpoint which serves as the trigger point to contact automountd. Next, autofs looks for NIS maps as listed in the /etc/nsswitch.conf file (/etc/rc.d/init.d/autofs start).
2. *stop*: stops autofs and all automounts (/etc/rc.d/init.d/autofs stop).
3. *status*: displays the present configuration and all running automounts (/etc/rc.d/init.d/autofs status).
4. *reload*: rereads the auto.master map and kills the no more appearing automounts and starts those for new mountpoints. Note that changes made in the maps are taken into account at next startup. On the other hand, the changes made in auto.master imply an autofs restart (/etc/rc.d/init.d/autofs reload).

In short, autofs is nothing else but a script consulting auto.master before starting the automounts associated to each described mountpoint.

Automount/AMD

AMD works from an initial mountpoint (the one found by autofs to start automount) and from a new map describing the features of this initial mountpoint. AMD gets its main configuration information from a file called /etc/amd.conf. This file contains global settings for the daemon, and, unlike autofs, it delegates the configuration of different local mountpoints to separate map files. The map associated to the automount will hold all required information for mounting the file system automatically. This automatic mount is done as soon as someone tries to access something in the directory tree starting from the mountpoint. AMD does a few other things behind the scenes to keep operations healthy. First of all, it sends out RPC requests at regular intervals to all the servers it knows to see if they are alive. If one isn't, AMD will not try to mount it. This checking also allows AMD to offer access to replicated file systems; that is, you can set up multiple redundant servers, and if one goes down, AMD will try to mount another one only if the file system is not currently mounted. Note that it can’t unmount/remount from a dead server to a new server.

Next, the file systems are automatically "unmounted" after an inactive period (the default timeout is 5 minutes). Setting the timeout value that determines unmount after an inactive period to 15 minutes is a best practice recommendation. However, this value may vary for different environments as a very high timeout value will not update the changes to the maps quickly, and with a very low value the unmounting will happen frequently to remove the cache entries.

Both automount and autofs allow the administrator to configure all file systems a machine can access the same way the administrator would be using the mount command. The user can then access these systems in a fully transparent manner, without worrying about how the kernel will answer the request and without needing ‘root’ access to mount the file systems.

The pair (automount and autofs) can be seen as the client/server model used in networking. A server is running, waiting for a request. When a request comes to the server, it spawns a new thread of execution answering the request while the other one is waiting for new requests. Here, *autofs* is the client triggering the mount request to automount and *automount* the one of the "duplicated server.” The requests are runtime entries, and the information required to resolve the request is held in configuration files.

Automount Maps and Configuration

On RPM-based systems like Red Hat or SuSE installation, if autofs has not been installed by default then it can be installed from the distribution CDs. Installing the RPM packages will get you to this point easily enough, but here's the part you might not be sure about if you haven't done this before.

Automount determines which mountpoints to monitor and which file systems to mount from a special set of configuration files called maps. Maps can reside on the local machine or be managed via NIS. By default, all local automount maps are located in the */etc* directory and typically have filenames prefixed with *auto*.

The automount configuration primarily consists of a master map called the auto.master located in the /etc directory. The auto.master contains information regarding each automount-managed mountpoint listed in it.

Master: autofs does utilize the *name service switch* to locate the source of maps, and therefore it must be specified as a [maptype] along with the [mapname] as mentioned in the format below. This file contains the source of all further configurations where automount-owned file systems are to be mounted and any other map files that are to be read. The automount consults the master map only at startup time. A modification to the master map will take effect only the next time you restart automount.
Format: `mountpoint [maptype:]mapname [mountoptions]`

'`mountpoint`' could be a full path to the directory used as a mountpoint. The directory path is created if it does not exist. '+' indicates if the map is an *included* map. '/' indicates if this is a *direct* map. The string ``/-'' tells automount that the `mapname` field value is the name of a direct map that automount is to read for file systems to mount.

'`mapname`' is the map file that has to be read. This field can also have a '`-hosts`' if the map is a "/net" map or '`-null`' if we want to override a specific map entry. If the map file begins with a '/', then it is interpreted as a local file; otherwise the name service switch mechanism is used to locate the source of the problem. A [maptype] can be either a file, program,yp,nisplus, or ldap.

'`[mountoptions]`' are a list of options that are passed to the mount command (separated by commas) to control the mounting of the entries mentioned in mapname, unless the entries in mapname list other options. Mount options listed in a direct or indirect map override options listed here.

Example:

Sample auto.master:

```
/eng_tools     file: /etc/auto_eng        -ro
/net           -hosts
/misc          file: /etc/auto_misc       --timeout=900
/-             file: /etc/auto.direct     -ro
/u             yp:automap                 --timeout=900
+auto_master
```

The first entry in above example is a pointer to an indirect map located at /etc/auto_eng; the remote file systems listed in this map will be mounted under /eng_tools/ in the indirect map. All file systems listed in /etc/auto_eng will be mounted read-only unless this option is overridden in the indirect map. automount creates /eng_tools if it does not exist. If it exists and is not empty, the existing contents are hidden by the automounted file system.

In the second entry; with -hosts you cannot choose individual mountpoints to mount each file system; all mounting is to a single mountpoint. The -hosts map is a built-in map supplied by automount. This map allows a client to access directories that are exported from any host in its host's database. The location of the host's database that your system uses is determined by the services running on your system (DNS, NIS, local) and how those services are specified in the /etc/nsswitch.conf file. By default, this mountpoint is /net. When the automountd starts, it builds the mountpoint named /net and listens for requests that cross this mountpoint.

When a user executes **cd /net/jelly**, the automountd detects that the mountpoint /net has been accessed. It consults its maps and finds that the -hosts built-in map is specified for the /net mountpoint. It then executes the library routine 'gethostbyname jelly'. In case the name server is not running, it looks for an entry for jelly in the /etc/hosts file on the local host. Then the

gethostbyname returns information on how to reach the server jelly. If gethostbyname cannot acquire this information, the **cd** command fails.

The third entry accesses files in /misc. automount will read the file /etc/auto.misc to find the mount options and the key associated to the file system. The 'timeout' option says that the mounted file systems can try to unmount themselves 900 seconds after use. A /etc/auto.misc may have the following contents:

```
payroll       -fstype=nfs   moon:/vol/vol09/payroll

sales         -fstype=ext3  sun:/vol/vol07/sales

  *                         mars:/vol/vol02/home/&
```

'auto.misc' can also contain a '*' that indicates a wild card entry. This entry is consulted if the specified key does not exist in the map. The automount queries mars for any username it does not find in the indirect map. We can use nfs and ext3 file system in the maps while mounting.

The fourth entry is a pointer to a direct map located at */etc/auto.direct*; the remote file systems listed in this direct map will be mounted read-only unless this option is overridden in the direct map. The mountpoint is listed in the direct map. '/-' is a special token reserved to indicate that the map is a direct mount and is not associated with any specific top-level directory.

The fifth entry, '/u', is slightly different where it specifies 'yp: automap' in the map field. We also have the keyword 'file:' in front of the other map field entries, even though 'file' is the default because in RH 7.3, autofs spits out a lot of errors until the keywords are mentioned in the map field. The 'yp' keyword tells the automounter to look up the map that it is using. It checks with the /etc/nsswitch.conf [automount: files NIS] to indicate that NIS is used.

The sixth entry begins with a (+) followed by a mapname, auto_master, which causes this map to be included from its source as if it were present in the master map [auto.master]. The contents of auto_master map may look like:

```
       /site  /etc/auto.site
```

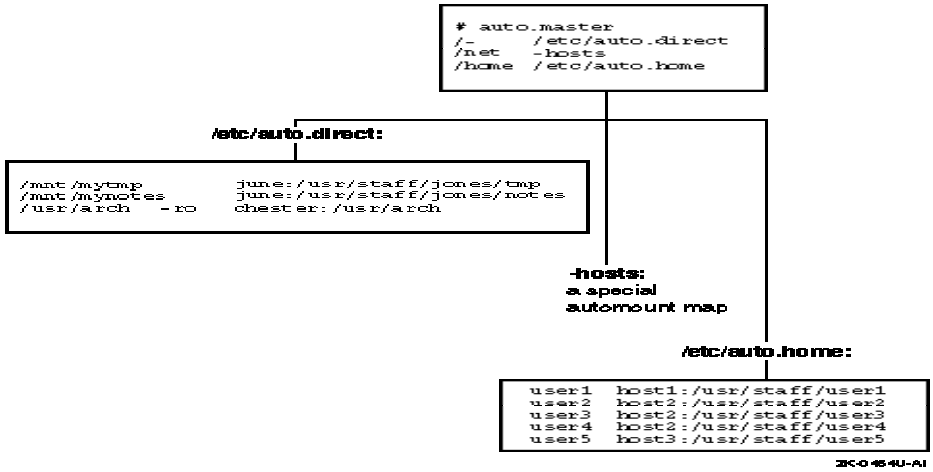
As a result of which the auto.master will look this:

```
/eng_tools       file: /etc/auto.eng          -ro
/net             -hosts                       -
/misc            file: /etc/auto_misc         --timeout=900
/-               file: /etc/auto.direct  -ro
/u               yp:automap              --timeout=900
/site            file: /etc/auto.site
```

The timeout on mountpoints is set to 15 minutes and can be changed using a command line option when the service is started.

Upon startup, autofs sources master maps from all locations specified in /etc/nsswitch.conf on the automount line. A new variable has been introduced in autofs4 which allows the user to determine whether all master maps are sourced, or only the first master map which is found. This variable is ONE_AUTO_MASTER, and should be set to '1' in /etc/ sysconfig/autofs to enable this new behavior. The default setting is '0', which preserves backward compatibility.

The following diagram illustrates an auto.master map that points to the /etc/auto.direct direct map, the built-in -hosts map, and the /etc/auto.home indirect map. Each map to which the auto.master map points is expanded to show its sample contents. Note that all of the information contained in the master map can be specified on the command line. The master map, however, simplifies organization and administration of automount.



There are mainly three types of automount maps: Direct, Indirect and Multimount.

Direct: These maps contain a list of full directory paths and the location from which the file system is to be mounted. A direct map entry can also contain pointers to other maps that automount should read (that is, nested maps). Each entry in a direct map is an automount mountpoint, and automount only mounts its daemon at these mountpoints at startup. Therefore, adding or deleting an entry in a direct map will take effect only the next time you restart automount. Existing entries, however, can be modified (for example, changing mount options or server names, but not names of mountpoints) while automount is running, and these will take effect when the changed entry is next mounted, because automount consults the direct maps whenever a mount has to be done.

For instance, you modify the file /etc/auto.direct so that the directory /usr/src is now mounted from a different server. The new entry takes effect immediately (if /usr/src is not mounted at this time) when you try to access it. If it is mounted now, you can wait until the default unmounting takes place before you access it. You can unmount before it expires with the umount command; if you do, you must notify automount that the mount table has changed by stopping and restarting the automount before accessing the directory. You can now mount the directory from the new server. However, if you wanted to delete the direct map entry containing /usr/src, you would have to restart automount for the deletion to take effect. Another disadvantage of direct maps is when automount reads a direct map; it creates an entry for each automounted directory in the internal mount table, /etc/mnttab. This can cause the mount table to become very large.

The advantages of using a direct map are:

- A user can see the contents of a direct-mounted directory with the ls command. If the contents are not mounted, ls will mount them.
- Direct-mounted automounted directories can share the same parent directory with local or standard-mounted files and directories.

```
Format:        key          [mountoptions]             location
```

‘key’ is the full pathname of the mountpoint.

‘[mountoptions]’ are a comma-separated list of options that apply to a particular mount. They could also be map options such as ‘-fstype=ext3’.

‘location’ specifies the file system to be mounted on the ‘key’. This conforms either to a single file system, replicated server entry, or multimount entry.

Note: *The same format applies for all the three kind of maps discussed here along with the following sections.*

A replicated server entry supports multiple read-only copies of the same data. The entries can come from different paths on different servers. The server selection has to have the closest network proximities and lowest weight.

```
/mnt/prods/demo    ro  svlf01,svlf10, svlf07(1),svlf09(2) \
                       rtpf09:/vol/vol05/prod/demo
/mnt/app/demo      ro  svlf01:/vol/vol06/app/demo  \
                       svlf02:/vol/vol02/app/demo
```

A [netmask] field is available in the /etc/nsswitch.conf file, or a /etc/ netmasks file can be manually created for the replicated servers' functionality to work properly. In the above example, if NetApp Storage systems svlf01 and svlf10 are unreachable or down, then the clients try to connect to svlf07 (1) and svlf09 (2) in sequence, depending on the weights assigned.

The automount implementations provide multiple servers to the mount command. NFS takes care of switching the servers when one doesn't respond.

The following is a sample direct automount map:

```
/mnt/demo            -ro,hard   emeaflr01:/vol/vol02/demo
/mnt/demo/users      -ro,hard   rtpflr09:/vol/vol10/home
```

To illustrate a direct mount, let's assume that automount was configured to mount the remote file system emeaflr01:/vol/vol02/demo in the EMEA region onto a local directory, /mnt/demo, at Sunnyvale. When a user issues a command to access a directory or file under /mnt/demo, this should cause the automount to mount it automatically. These are the steps automount takes to complete the mount:

- `mkdir /tmp_mnt/demo`

If the /tmp_mnt and /tmp_mnt/demo directories do not exist at the time, automount creates them as well.

- `mount -f NFS emeaflr01:/vol/vol02/demo /mnt/demo`

automount performs a conventional NFS mount.

As a result, emeaflr01:/vol/vol02/demo appears to the user to be directly NFS-mounted on the local /mnt/demo directory. The mounting of the remote /mnt/demo directory hides (makes unavailable) any subdirectories under the preexisting /mnt/demo.

Notes: 1) Do not automount a remote directory on a local directory that is a symbolic link. If you are using NIS to manage your AutoFS maps, you need to be careful of situations where the NIS server is also a server from which file systems are automounted. If the mountpoint specified by the NIS map is the same as the actual source directory on the server, AutoFS may attempt to mount the source directory over itself when a user accesses the mountpoint on the NIS server. This may cause the directory to become unavailable.
2) You cannot use the asterisk (*) wildcard in a direct map.
3) A limited direct mount map was supported in autofs4.1.

Indirect: These maps describe the mountpoints, or directory hierarchy, under the directory specified in the auto.master. An indirect map does not contain the full pathname of the mountpoint. An indirect map is always dependent on the master map that has the arguments for automount command, or through nested maps that contain a direct or indirect map. There can be more than one indirect map. Entries can be modified, deleted, or added to indirect maps, and the change will take effect the next time the map is used, which is the next time a mount occurs. To maximize performance and minimize kernel memory usage, no

nodes are created as a result of *ls -l* request. Nodes for the given entries will only be created if a process makes a subsequent lookup of entries listed by ***ls –l***.

Note: *An indirect map hides any local, standard-mounted, or direct-mounted files or directories underneath the mountpoint for the map.*

The main advantages of using indirect maps are:

- Because entries can be added to and deleted from indirect maps without restarting automount and because they do not clutter the mount table like direct maps do, indirect maps are preferable and should be used whenever possible.
- When automount reads an indirect map, it creates only one entry for the entire map in the internal mount table, /etc/mnttab. Additional entries are created when directories are actually mounted. The mount table does not occupy unnecessary space because only mounted directories appear in it.
- Newer versions of AutoFS allow a user to view the available mountpoints for indirect maps, without having to actually mount each file system when browsability is enabled by default.

The format is same as the 'Direct Map'.

/eng_apps file: /etc/auto.eng -ro

Indirect maps are usually called /etc/auto.name, where name helps you in identifying what is configured in the map. In the above example 'eng' is represented as the 'name'. If you plan to use NIS to manage your AutoFS maps, and if your file system does not support filenames longer than 14 characters, restrict your indirect map names to 10 characters or fewer. In the older versions of autofs, the direct map auto.eng had a dot (.) in the name. But after addition of NIS+ support in aufofs5, this was changed to auto_eng (underscore instead of dot) as NIS+ is hierarchical in nature and the underscore is used to separate the levels.

The local_subdirectory specified in the indirect map is the deepest subdirectory in the local directory path name. For example, if you are mounting a remote directory on /nfs/biz/apps, the local_subdirectory specified in the indirect map is apps.

The local_parent_directory specified in the master map is all but the deepest subdirectory in the local directory path name. For example, if you are mounting a remote directory on /nfs/biz/apps, the local_parent_directory specified in the master map is /nfs/biz.

The local_parent_directory and local_subdirectory should not exist; AutoFS creates them when it mounts the remote directory. If the local_parent_directory or local_subdirectory contains files or directories, they are hidden beneath the remote directory when it is mounted.

Note: The local_subdirectory and local_parent_directory must not be symbolic links.

If you are using NIS to manage your AutoFS maps, make sure the local mountpoint is different from the exported directory on the server. If they are the same, the server may attempt to mount its exported directory over itself, and the directory will become unavailable.

The following example contains sample lines from an AutoFS indirect map on the NFS client, sage. The pound sign (#) indicates a comment. AutoFS ignores everything from the pound sign to the end of the line. The long line entries may be broken by quoting the new line character with a backslash (\).
/etc/auto.eng file

```
# local mountpoint        mount options       remote server:directory

       draw                -nosuid                    thyme:/vol/vol05/biz/apps

       write               -nosuid                    basil:/vol/vol06/write
```

The following example contains sample lines from the AutoFS master map on the NFS client, sage. The master map also includes an entry for the direct map, /etc/auto_direct.

```
# /etc/auto_master file
# local mountpoint        map name            mount options

       /-               file:/etc/auto.direct

       /u/eng           file:/etc/auto.eng
```

Ghosting of map directories allows you to see the directories in the autofs map without mounting them. When they are accessed, such as when a directory listing is requested, the map entry is mounted so that it is seen. In Solaris '–browse' is set by default.

The Linux automounter supports ghosting of indirect map entries; this is tantamount to the 'browse' option for Solaris. Each indirect map entry will appear in the automounted directory, but will only be mounted upon access. At daemon startup time, the automounter creates directory entries for each of the keys in the indirect map. If a wildcard entry is present, directories matching it will only be ghosted after they are explicitly accessed. Thus, if you have a map which contains only a wildcard key, then your automounted directory will initially be empty. However, the automounter will remember any entries which are mounted.

Ghosting is *not* enabled by default. To enable it, edit file /etc/sysconfig/autofs and append the "-g" option as follows:

```
DAEMONOPTIONS="--timeout=900 -g"
```

These changes will be effective after restarting autofs service:

```
# service autofs restart
```

After the ghosting option is set, with respect to the above example:

Directory listing of /u/eng will contain:

```
/draw
/write
```

A best practice recommendation is to enable the browsing or ghosting of maps, as it is inexpensive and the information is more readily available if it is hierarchically organized. For example, a large corporation may be divided into divisions, and each division may be subdivided into organizations. Members of a team are easily identifiable when the division and organization to which they belong are known.

Multimount: These entries allow the user to specify a directory hierarchy that will be mounted. Mount options can be specified per mountpoint. This feature allows the automount daemon to seek multiple lookup methods in succession. A lookup could query NIS and file maps.

```
project      -rw                                    \
   /                 /svlflr01:/vol/vol0            \
   /bin      -ro     /svlflr01:/vol/vol02/bin       \
   /user             /svlflr01:/vol/vol03/home      \
   /binary   -ro     /svlfl03:/vol/vol04/binary     \
   /scratch          /svlfl03:/export/scratch
```

This example demonstrates how to tag multiple servers into a single directory structure. One point to note here is that the userdir directory contains both an NFS-mounted file system and mountpoints beneath it. When any directory in this hierarchy is accessed, the automount daemon

mounts every entry in the directory hierarchy. The expiry of a multi mount entry also happens atomically.

The multi-mount mechanism implements -hosts. '-hosts' is a special map that treats 'key' as a <servername> and performs a 'showmount –e' on the key or the <servername> and sorts the output. The program map auto.net generates multimount entries on the fly, and the daemon mounts them when /net/<servername> is accessed. The <servername> is used as the key. [auto.net] is part of autofs and will not show the contents of the program.

Autofs4 in Linux has some shortcomings related to multimounts. The multimount maps mount and unmount as a single unit that causes all the subdirectories that are not even required to mount. This puts a lot of pressure on the reserved port space. /net is implemented as a multimount map. Autofs5 alleviates this issue with the lazy mount implementation.

autofs: The Inside Story

The automounter mainly works in two parts: a user-space daemon, which is responsible for parsing map options and issuing mount and umount commands, and the second part is a file system implemented in the kernel. The daemon is further broken up into the actual daemon that works along with a set of loadable modules. *All* Linux distributions ship with the autofs daemon and init.d startup script to start the automounter and preinstall a /etc/auto.master and /etc/auto.net configuration file. The timeout variable is set in /etc/sysconfig/ autofs.conf. The installed file sets a default of 60 seconds of inactivity before unmounting the device. However, the timeout value set to 900 is recommended as a best practice.

For example, if we have the following map in the auto.master:

```
/net          /etc/auto.net
```

The autofs starts with the init script that parses the auto.master map and spawns one automounter daemon for each mountpoint listed. The above-mentioned example will result in an automount command with the following parameters:

```
/usr/bin/automount                \
/net   program       /etc/auto.net
```

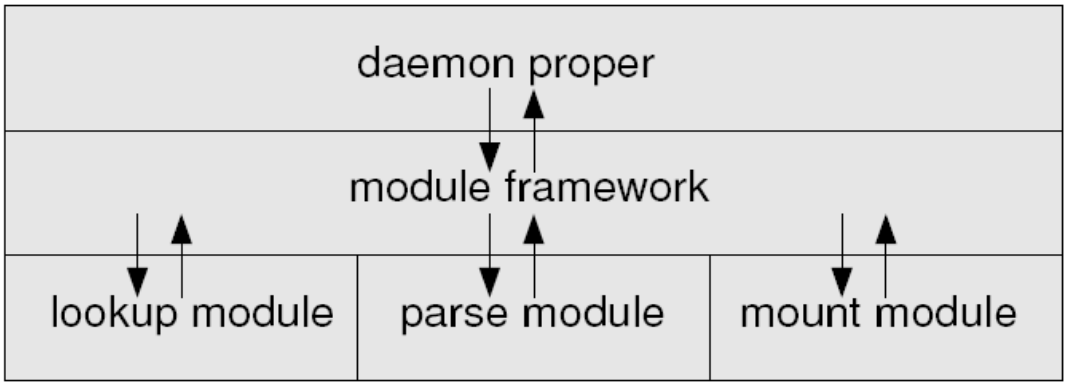
Therefore the daemon takes as its options a mountpoint, the type of the map to be loaded, and the name of the map to be loaded.

There are three types of loadable modules: *lookup, parse, and mount.*

The *lookup* modules are used to look up a given key in a map and have code that understands how to get information from a map source like files, NIS, and NISPLUS.

The *parse* module is responsible for parsing the value part of the key value pair.

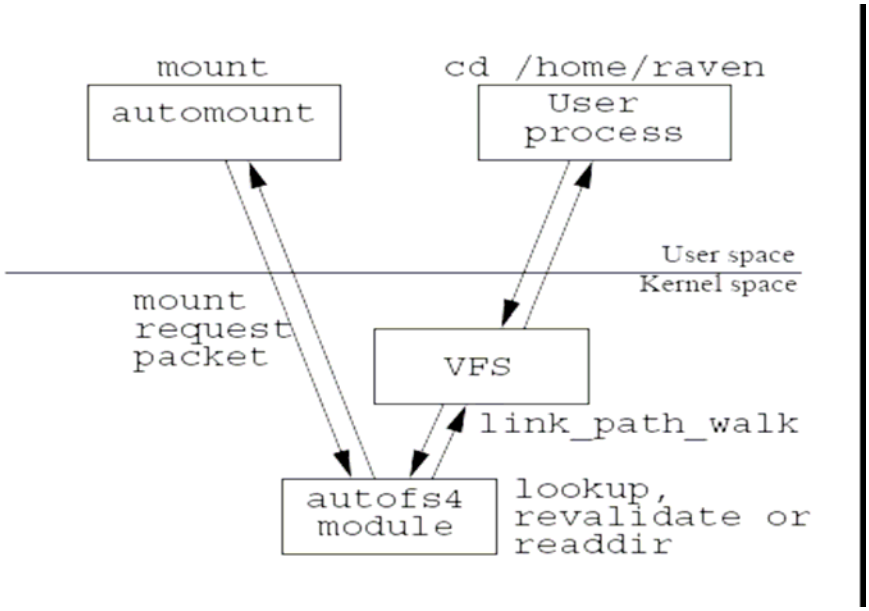
The *mount* module does the actual mounting of AUTOFS, NFS, generic, and so on. This module also knows about how to pass arguments on to the mount command. In the case of NFS, this module is also responsible for parsing replicated server entries.



Autofs uses VFS (Virtual File System Switch) to interface with the kernel. VFS is a software layer that handles all system calls related to standard UNIX® file systems. The VFS uses the callback functions to carry out standard file system operations. The primary objects are the *superblock,* the *inode,* the *dentry,* and the *file objec*t. The *dentry* object represents a single component of a directory path. The VFS does a lookup of the filename path and resolves to a corresponding *dentry* by traversing each of its path components.

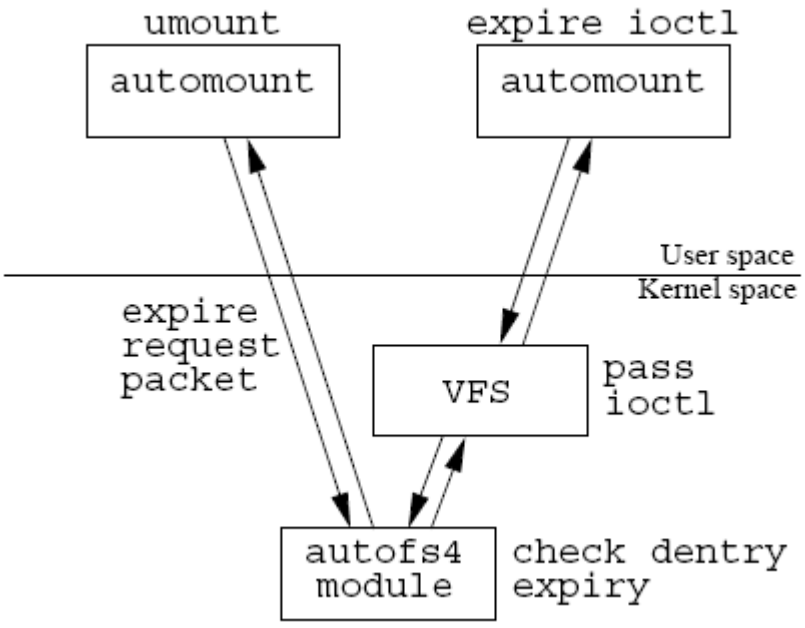
How autofs Does a Lookup

Mount requests are triggered when commands or functions such as a cd, ls, or open cause the VFS to traverse a directory path within the autofs file system. This in turn calls the autofs4 function *lookup* to check if the directory doesn't exist, or revalidate if it does. There are two ways autofs can decide whether a mount needs to be triggered. First, if the directory doesn't exist, *lookup* creates a negative *dentry* and passes it to the *revalidate* function. *Revalidate* knows that a mount needs to be requested when it sees a negative dentry, so it sends a mount request packet to the *automount* daemon. The daemon then issues a mount command and returns a status when done. For the second case, when the directory exists, the *revalidate* function is called and decides whether a mount request needs to be sent by checking if the *dentry* is an empty directory and not already a mountpoint. If this is the case, then a mount request packet is sent to the daemon.



How autofs Expires

Expiration of mounts is achieved by calling the autofs *expire ioctl* procedure. The autofs daemon does this when it receives an alert, which is set at a threshold of one quarter of the mount timeout. The daemon looks for mounted file systems under the path on which it is mounted and asks the autofs kernel module if it can expire them. If the kernel module decides that the daemon can expire a mounted *dentry*, then it sends an expire request packet to the daemon, which in turn issues an umount command and returns a status when done.



However, the *expire ioctl* may generate some race conditions while expiring multiple mounted directories at the same time or when a kernel check happens to count a directory hierarchy, or when an application

traverses a directory hierarchy that it is trying to expire or when an unmount fails. This race condition may prove to have trivial impact to the automount daemon.

Autofs Tuning

There are two aspects of the automounter that need to be considered for optimum performance.

1) The RETRY count on the mount attempts.

   By default the 'retry=' value is set to '0' in the master map. When retry is set to 0, typical automounters will make a single remote procedure call over UDP, with a timeout of 15 seconds. This means that unless the mount request is serviced within 15 seconds, the automount attempt will fail. It also means that if the UDP packet is dropped somewhere in the network stacks of the client or server, or in the network path, the automount attempt will fail. If possible, the automounter should use TCP (add '–proto=tcp' for a mount entry in the master or indirect map as mentioned in example #1 below) instead of UDP as options in the maps for mounting. The autofs will make one or two attempts to mount the NFS file system. This may not be a very nice thing if the file system that doesn't get mounted is a home directory as the user logs in, or a database as the DBMS starts running. The workaround to this issue is to override the automounter's default. The 'retry=n', where n is the number of retries, should be set appropriately with respect to environment requirements. When the option 'retry=1000*'* is set to the automounter maps, automount proves to be more robust. However, we do not recommend having the retry=0. If there is a "version mismatch error" while automounting a file system, then the following option in the automount configuration (master map) has to be used in order to avoid any NFS errors while using automounter:

   ```
   mountvers=2,nfsvers=3.
   ```

2) Duration of a mount.

   The idea of an automounter is that when NFS file systems are no longer used, then it should unmount them. This is a good thing, because from time to time, automounter maps are changed. If the automounter never unmounted anything, then the map updates would never be seen by the client. Ideally, the automounter would wait to attempt an unmount when it knew the file system hadn't been used for some amount of time. However, automounters don't have an interface to know if there are any processes currently with open files in the NFS file systems. As a result, the automounter has a simple approach: it waits some number (N) of seconds, and then attempts to unmount a file system, and does this every N seconds. If the file system is in use (busy), the unmount fails.

   It turns out that an unmount attempt of a busy file system can be really bad performancewise. An unmount attempt will flush all cached data, force all modified but unwritten blocks to be written to the NFS server, and flush all cached metadata (attributes, directories, and name cache). At the end of that, if there are still references to the file system, the unmount fails. This means that the processes benefiting from caching will now take latency hits as their working sets of cached data are rebuilt.

   Therefore tuning the automount duration higher is highly recommended. The autofs command has a -t or the –timeout option to set the duration to override the default of 60 seconds. You want to strike a balance between good performance and the benefits of resynchronization with automount map updates.

Limitations of autofs4.0 (autofs-4.1.3-187)

1) The master map semantics do not allow the init script to merge the corresponding maps of multiple keys while parsing the contents of the auto.master. The init script does not also handle the use of –null map. The -null map is used to mark a master map mountpoint as excluded from subsequent parsing. It also unmounts these entries during a reload of the master map.

2) Included maps that have the ability to include a map inline from within another map using the syntax '+mapname' are currently not supported in autofs4.
3) Linux autofs4 does not have the ability to handle a large number for mounts for two reasons:
    a. The number of devices available for mounts is limited. NFS and autofs use the anonymous block device major number. In a 2.4 stock kernel, this provides a maximum of 255 devices and hence a maximum of 255 mounts. However, later releases of 2.4 kernels had the value increased to 2048, which means 2048 mounts were supported. In the 2.6 kernels, the number of anonymous devices was substantially increased. In spite of the high number, the limit is never reached because of the port allocation limitation in the RPC layer, as mentioned in the second point.
    b. Autofs performs RPC probing to discover whether the target server is available before performing a mount. This process leads to as many as nine ports per mount being used during a mount, which causes rapid exhaustion of reserved port space. The RPC port allocation algorithm allows for a maximum of 800 concurrently mounted file systems when using UDP. The practical limit for TCP protocol mounts performed in rapid succession is around 100.
4) The anonymous device limitation and port reservation exhaustion mentioned earlier cause problems for multimounts in autofs4 as they are handled as single unit due to possible nesting dependencies within the mount hierarchy.
5) Autofs4 does a limited parsing of the /etc/nswitch.conf file. While autofs references the nsswitch file to locate the master map during startup, other consumers of the nsswitch file use the standard glibc interface to access the file, which is not conducive for autofs.
6) Autofsv4 offers limited support for direct mount.

The introduction of autofs5 in the 2.6.18+ kernels has addressed some of the above-mentioned caveats in autofs4.

Advantages of Using autofs5 (autofs-5.0.1-0.rc2.42)

1) autofs5 has a complete direct mount implementation. This paves the way for lazy mount/unmount of multi-mounts and host map implementations.
2) A lazy mount/unmount concept was introduced in autofs5. This helps a lot for implementing multimounts. It still creates the autofs directory hierarchy, but now it does not mount everything at once. Triggers are installed in the appropriate top-level directories, and are mounted upon access.

    Here is an example:

```
project       -rw
/                    /svlflr01:/vol/vol0              \
/bin   -ro           /svlflr01:/vol/vol02/bin   \
/user                /svlflr01:/vol/vol03/home  \
/bin/binary -ro      /svlfl03:/vol/vol04/binary \
/user/scratch        /svlfl03:/export/scratch
```

When 'project' is accessed, we mount the entry corresponding to / and install direct mount triggers for each offset within the list bounded by nesting points. In this case, we install direct mounts for /bin and /user. Similarly, when one of these mounts is triggered, we mount it and install the corresponding triggers. In the example we mount the entry for /bin and /user and then install triggers for /bin/binary and /user/scratch, respectively.

3) A lot of the resource issues are addressed with host maps. A separate module was implemented to handle the host maps. This module looks at the local host table rather than parsing the exports file in order to obtain the information for lookup when they are accessed.
4) A new parser for handling /etc/nsswitch.conf map source lookups was included in autofs5. This helps to do a name service switch for the master and submount maps more effectively. The automount daemon now calls the common lookup module instead of calling the lookup modules directly and iterates through the list of sources found by the parser while scanning through the contents of the /etc/nsswitch.

5) There is a separate module to parse the master map instead of the init script doing the parsing in autofs4. The new version of autofs also merges the multiple entries for a key in the master map.
6) The new autofs implementation includes the support of included maps (+). The lookup modules watch for the (+) as the first character in the map key and call the higher common lookup function to do the work and then continue after it returns. LDAP support is also integrated into the new version of autofs.

Troubleshooting autofs

The following information and configuration are required for troubleshooting autofs further. However, the data needs to be provided to appropriate RedHat support for analysis.

- autofs rpm version, obtained via 'rpm -q autofs'
- kernel version, obtained via 'uname -r'
- To check the status and start the autofs, the following commands may be used: /etc/init.d/autofs start or /etc/init.d/autofs status
- Contents of your autofs maps. This includes auto.master and at least the map which is causing problems.
- contents of /etc/nsswitch.conf
- contents of /etc/sysconfig/autofs
- Steps to reproduce the problem, or circumstances surrounding the problem.
- debug output.
    - Add a line like the following to your /etc/syslog.conf:

      daemon.* /var/log/debug
    - Tell autofs to log debugging information for the problematic map. This can be done by adding a --debug to the entry for this map in /etc/auto.master. For example:

      /misc /etc/auto.misc --timeout=900 --debug
      If, instead, you need to log debug output for all maps, you can do the following:
      For autofsv4, add the --debug option to the DAEMONOPTIONS variable specified in /etc/sysconfig/autofs.
      For autofsv5, set DEFAULT_LOGGING to "debug" in /etc/sysconfig/autofs.
    - Restart syslogd (or send it a HUP signal) (service syslog reload).
    - Restart autofs. Make sure it restarted. Check the logs, and ensure that the problematic mountpoint was shut down properly and subsequently restarted. Otherwise, you will not get any debug output in the logs.
    - Now you're set to get debug output. Reproduce the problem, and attach the logs with your bug report.
- If it's not a readily reproducible problem, then gather some system state when it does occur. The following are useful:
    - Current automount status:
    - ```
      # /etc/init.d/autofs status
      # ps auxwww | grep automount
      ```
      ```

- Output from sysrq-t:

```
# sysctl -w kernel/sysrq=1; echo t > /proc/sysrq-trigger
kernel.sysrq = 1
```

You will find the output in /var/log/messages.
To turn this off, after the problem occurs.

```
# sysctl -w kernel/sysrq=0 echo t > /proc/sysrq-trigger
 kernel.sysrq = 0
```

Automount Debugging in Solaris 10

Starting in the Solaris 10 release, you can use the /etc/default/autofs file to configure your autofs environment. Specifically, this file provides an additional way to configure your autofs commands and autofs daemons. The same specifications you would make on the command line can be made in this configuration file. However, unlike the specifications you would make on the command line, this file preserves your specifications, even during upgrades to your operating system. Additionally, you are no longer required to update critical startup files to ensure that the existing behavior of your autofs environment is preserved. You can make your specifications by providing values for the following keywords:

AUTOMOUNT_TIMEOUT =<*num*>

Sets the duration for a file system to remain idle before the file system is unmounted. This keyword is the equivalent of the **-t** argument for the automount command. The default value is 600, but 900 is the best practice recommendation. Appropriate timeout values may be specified depending on the environment requirements.

AUTOMOUNT_VERBOSE =TRUE | FALSE

Provides notification of autofs mounts, unmounts, and other nonessential events. This keyword is the equivalent of the **-v** argument for automount. The default value is FALSE.

AUTOMOUNTD_TRACE =TRUE | FALSE

Expands each remote procedure call (RPC) and displays the expanded RPC on standard output. This keyword is the equivalent of the **-T** argument for automountd. The default value, 0, turns off such tracing. Starting with 1, with each higher value, the verbosity of trace output increases.

In Solaris 10, after the parameters in /etc/defaults/autofs are changed, the autofs has to be restarted.

svcadm refresh autofs

All the RPC calls and the autofs mount/unmounts log to /var/svc/log/system-filesystem-autofs:default.log.

More Examples of autofs Implementation with Network Applicance Storage system.
Example #1

1. mkdir /mnt/filers
2. Add the following line to **/etc/auto.master:**
   - /mnt/filers /etc/auto.filers nodev,rw,intr,noquota
3. Create /etc/**auto.filers**. A sample '**auto.filers'** is listed below.
4. Tell automounter to reload the maps.
   - /sbin/service autofs reload

The /etc/auto.filers may have entries as follows:

```
sqlserver   -rw,intr,proto=tcp                  crusher:/vol/sqlserver
backups     -rw,intr,proto=tcp,timeout=900      pickard:/vol/backups
barkley             -rw                         barkley:/vol/vol0
bashir              -rw                         bashir:/vol/vol0
```

Example # 2

A good example of a typical indirect map from a UNIX host:

```
% cat /etc/auto.home
        joe             filera:/vol/vol6/home/joe
        fred            filerb:/vol/vol6/home/fred
        *               filerc: /vol/vol6/home/&
```

This would create automountpoints for /home/joe and /home/fred.

```
% cat    /etc/auto.direct
        /usr/man  filera: /vol/vol5/usr/man
        /usr/apps         filerb: /vol/vol7/usr/apps
```

This would create automountpoints for /usr/man and /usr/local.

The auto.master file may be as follows:

```
% cat /etc/auto.master

        /-              /etc/auto.direct    -ro
        /home           /etc/auto.home
```

References

www.usenix.org/events/lisa99/full_papers/labiaga/labiaga_html/index.html

www.linuxsymposium.org/2006/linuxsymposium_procv2.pdf