# Autodir HOWTO

Venkata Ramana Enaganti <ramana <> intraperson dot com>

2004-09-23

Revision History

| | | |
|---|---|---|
| Revision 1.04 | 2007-5-25 | VRE |
| | Minor updates | |
| Revision 1.03 | 2006-09-15 | GaMA |
| | Review requested by author. | |
| Revision 1.02 | 2004-12-25 | VRE |
| | Minor updates | |
| Revision 1.00 | 2004-09-23 | VRE |
| | Initial release, reviewed by Rahul Sundaram at TLDP | |
| Revision 0.32 | 2004-09-13 | VRE |
| | New sections like requirements and others. | |
| Revision 0.10 | 2004-06-24 | VRE |
| | second draft | |
| Revision 0.9 | 2004-06-11 | VRE |
| | first draft | |

**Abstract**

This HOWTO is about the **Autodir** installation, configuration and other issues related to **Autodir**. The **Autodir** system is often applied for making home directories available in an easy way.

## Table of Contents

## Introduction

**Autodir** offers a simple and effective means to create directories like home directories in a transparent manner. It relies on the autofs [htp://www.autofs.org] protocol for its operation.

This document explains how to create directories on demand using **Autodir** in a transparent way to the applications. This document also explains using the transparent backup feature that is possible with **Autodir**, without system downtime for backup purpose; this applies for all directories managed by **Autodir**.

## Copyright and License

## Disclaimer

No liability for the contents of this document can be accepted. Use the concepts, examples and information at your own risk. There may be errors and inaccuracies, that could be damaging to your system. Proceed with caution, and although this is highly unlikely, the author(s) do not take any responsibility.

All copyrights are held by their by their respective owners, unless specifically noted otherwise. Use of a term in this document should not be regarded as affecting the validity of any trademark or service mark. Naming of particular products or brands should not be seen as endorsements.

## Feedback

Feedback is most certainly welcome for this document. Send your additions, comments and criticisms to the following email address : `<ramana <> intraperson dot com>`.

## New Versions of this Document

The latest version of this HOWTO will be made available from http://www.intraperson.com/autodir/.

## Credits / Contributors

In this document, I have the pleasure of acknowledging for language and technical review work:

- Rahul Sundaram`<rahulsundaram@yahoo.co.in>`

- Machtelt Garrels

## Before going into the details...

After releasing intraperson beta, I started working on a administration guide that deals with the administration aspects of **intraPerson**. For more details check http://www.intraperson.com. But I was stuck with one simple thing. It is easy to create users in LDAP - at least I think so - but how to create home directories for those users in LDAP wherever those LDAP accounts are imported?

I found some solutions, but I was not satisfied as every solution has serious drawbacks. After leafing through the Autofs documents and hacking a bit, I came to the conclusion that the Autofs protocol might offer a much better solution to this challenge.

The result is **Autodir**, based on the Autofs protocol.

## Why not pam_mkhomedir?

The PAM module `pam_mkhomedir` uses Pluggable Authentication Module architecture for its operation. As such, there are some limitations associated with it. For instance:

- Some servers may not authenticate users but they may expect user directories to exist. This means they do not use PAM, and in turn, `pam_mkhomedir` does not get a chance to create home directories. The notorious example is on email servers.
- PAM is always an optional component for authentication. Some services may not use PAM at all and use a different method to authenticate users. In this case `pam_mkhomedir` is never going to be used.
- Generally `/home` is owned by root and only root users can create home directories in it. Therefore the service that wishes to create home directories through PAM must be run as root, or else the home directory must have the same permissions as, for instance, `/tmp`.

Finally, **Autodir** is much wider in scope and supports many more interesting features.

## Where can Autodir be used?

- Where user accounts reside in centralized database like LDAP, SQL, NIS, NIS+ or other databases, from which user and groups are imported to other systems. To create, for example home, group directories in those systems which import these accounts from centralized database, on demand.
- To exploit its transparent backup feature for 24*7 online systems.
- It can even be used when accounts are in a local system, to some extent hiding what accounts exist in the `/home` directory, for example.

## What Autodir is not

**Autodir** can create directories but it does not remove them once user and/or group entries are removed from the system accounts database. Use custom made scripts from cron for this.

## Differences between Autodir and Autofs

Issue arises when you are already using the Autofs package, handling the mounts of (home) directories. Let's look at the differences between the two packages:

- The main purpose of autofs is to deal with network mounts on demand instead of mounting all at the same time, which results in preserving system resources. Though there is some support in the autofs package to mount home directories on demand, the requirement is that *these home directories must exist already*.

  On the other hand, **Autodir** specializes *only* in local directory creation and mounting them on demand.

  **Autodir** can also create real directories in disk file systems, which do not reside in one single flat base directory. This is how utilities like **useradd** create by default. In a standard file system setup, all home directories reside in the base `/home` directory. For file systems like ext2 and ext3 performance will degrade if a large number of home directories exist in one single base directory.

  For applications accessing these directories, **Autodir** presents all directories for them in a *single* autofs mounted virtual base directory *on demand*; actual directories are created in subdirectories of some other directory in hierarchical style.

  For example, the real home for a user with username `user1` will be created as `/autohome/u/us/user1` if configured that way, but mounted in `/home` on demand for applications accessing the home directory in `/home/user1`.

  Permissions for the real base directory, where the actual home directories are kept (`/autohome` in the above example), are kept in such a way that `/autohome` can not be accessed by anyone except by root.

  This mounting of directories on demand and unmounting when not in use presents an interesting opportunity: the ability to tell whether a directory is in use or not. If a directory is not in use, a program like a backup application can be safely started when a directory is unmounted.

  **Autodir** exploits this capability by starting the command-line mentioned backup whenever a directory becomes unused.

- There is one more important issue to be presented if you are an administrator reading this document. **Autodir** does not call the external programs **mount** and **umount**, as is the case with the autofs package; rather, it uses system calls directly. As a side effect, it is faster and more reliable, but the `mtab` file is not updated. I felt this was not necessary as all mounts and unmounts are local directories.

- Another minor difference is that **Autodir** is completely *multi-threaded*. Autofs is also expected to be multi-threaded in future versions.

# How it works

**Autodir** uses modules to get specific functionality. The core **Autodir** implements generic functionality that modules can exploit and add specific functionality to.

At any moment only one module can be added to **Autodir**. If there are two modules, for example `autohome` and `autogroup`, then two processes of **Autodir** should be created so that each process can have one of the required modules attached to it.

For further explanation I chose the `autohome` module which handles transparent home directory creation.

## Assumptions

- The `autohome` module creates user home directories on demand if these do not exist already.
- It is assumed that user accounts exists, but the accompanying home directories do not - either because these accounts were created with the `-M` option with **useradd** or because these

accounts were imported from LDAP, NIS or some other external database for which home directories are yet to be created.

- It also assumed *for this explanation only* that all user home directories are expected to be in the `/home` directory.

## KISS

Keep it Simple: Some fine details are intentionally kept aside to make the explanation easy to understand.

First the autofs file system is mounted on the `/home` directory by **Autodir**. The Linux kernel is informed that `/home` is managed by a user space application, **Autodir**, from now on.

## Autofs?

Do not bother too much about the autofs file system if you do not understand about it. Just think of it as a special kind of file system, similar to memory based file systems but with some additional special properties.

```
                             +----------------+
                             |  Linux Kernel  |
                             +----------------+
                               /        \
                              /          \
                             /            \
                            /              \
+-------------+       +--------+       +------------+        +-----------------
| Application |------>| /home  |<----->|  Autodir   |<------>| autohome module
+-------------+       +--------+       +------------+        +-----------------
                           \                                          /
                            \  +----------------+                    /
                             +-|   /autohome    |<------------------+
                               +----------------+
```

Whenever an application or daemon needs access to a user's home directory, for example `/home/user-home1`, they directly enter into `/home/userhome1` to access it. The kernel, which notices this, informs **Autodir** if the `userhome1` directory does not yet exist already in `/home`.

**Autodir**, in turn, passes this request to the `autohome` module. The `autohome` module does not touch the `/home` directory. Instead it manages *real home directories* somewhere else, for example in `/autohome` as shown in the above figure.

The `autohome` module creates a real home directory if it does not exist in the `/autohome` directory. After it is successfully created or failed to be created, whatever the outcome, `autohome` reports back to **Autodir**. When the directory creation task has completed successfully, the path to real home directory is provided to **Autodir**.

If the `autohome` module reports success, **Autodir** creates `userhome1` directory under `/home` and mounts the *real home directory* from `/autohome` on it. At the end of the process, **Autodir** informs the kernel whether the whole operation was successful or not. Accordingly, the kernel allows applications to enter the directory, or, in case of failure, it reports that no such directory exists.

## Some definitions

Before going further it is better to understand the following terms to simplify explanation.

| | |
|---|---|
| **Virtual directories** | These directories do not exist on disk. Instead these are created and deleted on demand in memory. If the system reboots all these directories vanish. In the previous figure, all directories under `/home` are *virtual directories*. |
| **Virtual base directory** | This is the directory that holds all *Virtual directories*. This directory *does* exist on disk and therefore it remains even after reboot. In the previous figure `/home` is *virtual base directory*. |
| **Real directories** | These are the directories that actually reside on the disk. Even after reboot, these remain intact. In the previous figure all directories created under `/autohome` are *real directories*. |
| **Real base directory** | This is the directory that holds all real directories. In the above figure `/autohome` is *real base directory*. |

Each *virtual directory* is mapped to a *real directory*. This means that whatever is written to or modified in the *virtual directory* is actually sent to the *real directory*.

On reboot of the system *real directories* and their content remain intact. But *virtual directories* are again created on demand, exactly as they were before.

*Virtual directories* are removed if these are not used for a specified period of time, and created again if necessary. When a *Virtual directory* is removed, the backup program is started on the corresponding *real directory* - if backup is configured.

### Important

Applications should access only *virtual directories*. *Real directories* are hidden from applications. Only the root user can see them. There is one exception: backup programs always access the *real directories* only.

## Directory organization in the real base directory

Why should there be any special organization in the *real base directory*? If we just create all *real directories* in one *real base directory* there could be a performance penalty when there are a large number of *real directories* to be created. File systems like ext2/ext3 are not optimized for this kind of flat directory structure.

It would be much better if the *real base directory* is divided into more subdirectories, or even to divide these subdirectories again into more subdirectories. And in the final subdirectories actual home directories are kept.

There are three types of directory organization:

| | |
|---|---|
| **level 0** | Actually no organization. All home directories are created directly under *real base directory*. |

**level 1**
: The *Real base directory* is divided into more subdirectories. The subdirectory names are derived from the first character of the final directory to be created. For example, if the `user1` directory is to be created, first a directory named 'u' is created under *real base directory*. Then in that subdirectory the actual directory `user1` is created as `/<real_base_directory>/u/user1`.

**level 2**
: Same as level 1 organization but after the first level of subdirectories, a second level of subdirectories is created. The names here are the first two characters of the final directory to be created. For example, for user `user1`, as in the above example, the `/<real_base_directory>/u/us/user1` directory is created.

## Virtual directory expiration

When an application tries to access a *virtual directory* in a *virtual base directory*, **Autodir** creates the *virtual directory* in it if it does not exist already and mounts the *real directory* on it from the *real base directory*. Once this is done, a timer is started. If the *virtual directory* is not accessed from the *virtual base directory* by any application for the specified period of time, this directory is removed and the corresponding *real directory* in the *real base directory* is marked for backup.

The time period to wait for expiration can be given through a command line option to **Autodir**.

## Backup support

**Autodir** supports backup program launching when a *virtual directory* is removed after a period of inactivity. Removal of the *virtual directory* is itself an assurance that no other application can access the content and modify it.

Just like there is wait time for expiring a *virtual directory*, for backup **Autodir** also waits during some time after the expiry of the *virtual directory*, prior to starting the backup. This time period can be configured through a command line option to **Autodir**.

By design, backup programs are expected to operate on *real directories* but not on *virtual directory*. If the backup program try to access a *virtual directory*, then **Autodir** assumes some regular application is in need of that directory and the backup program is killed, even if the *virtual directory* accessing process is the backup program itself.

A separate backup process for each *real directory* is used. The backup program can be given arguments of *real directories* on which to operate.

### Note

Backup support is independent of any particular module being used. It is applicable to all modules with **Autodir**.

### Backup = real!

Backup programs should never access *virtual directory* or *virtual base directory*.

### Time off

The backup feature is not much use if the *virtual directories* are being accessed by applications all the time.

## Backup program requirements

**Autodir** demands some extra requirements from the backup program being used: when the backup is working on the *real directory*, with corresponding expired *virtual directory*, and that *virtual directory* is requested again by an application while the backup is running, the backup process is killed. First a `SIGTERM` is sent to gracefully stop it. But if it does not shut down in time - and it has one second to do this - a `SIGKILL` will be sent, which is guaranteed to stop the backup.

### Note

Only when the backup has stopped the application is given access to the requested *virtual directory*.

### Important

Whatever backup is used, it should be able to recover from this signal gracefully, not causing unrecoverable side effects.

One more important issue is the environment in which the backup runs. All backup programs run as root. But at the same time all unnecessary root privileges are taken away using POSIX capabilities. In other words these backup programs can read any file or directory that belongs to any user on the system and nothing more than that. Other than that, the backup process behaves like an ordinary user level process.

## Module options

There are two kinds of options that can be passed to **Autodir**. In the first type, options are for **autodir** itself and are common irrespective of which module is used. The other type of options are specific to the module being used. These options are called suboptions and are passed to the module being used; they are different from the main `-o` option. This is similar to the suboptions used with the **mount** command.

For example, suboptions to the example module `autohome` can be passed as follows:

```
-o 'realpath=/tmp/autohome,level=2,noskel'
```

Here `realpath`, `level` and `noskel` are suboptions for `autohome` module.

## Autodir requirements

- You will need a Linux kernel equal to or later than version 2.4. These kernel versions support mounting one directory on another directory. At this moment **Autodir** is not ported to other Unices but this may change in the future.
- **Autodir** requires the autofs kernel module based on protocol version 4. But it does not require the autofs user level package. The autofs kernel module is pretty standard and almost all distributions include it.

## Autofs kernel module

**Autodir** uses the autofs kernel module for its operation. The kernel module `autofs` must be loaded before starting **autodir**.

This can be done as the root user, using the **modprobe** command as follows:

```
# modprobe autofs
```

## Importing user and group accounts

If user and group accounts reside in a centralized database these must be imported before starting **Autodir**. How to do this is out of the scope of this HOWTO. There are a number of documents which explain in a clear way how to do this.

## Getting Autodir

At this moment **Autodir** is available in tar and rpm formats. More information can be found at http://www.intraperson.com/autodir/.

After downloading the source, follow these simple steps to install :

- Unpack the source.

  ```
  $ tar zxvf <tar file name>
  ```

- Move to the expanded directory and execute the following commands:

  ```
  $ ./configure
  ```

  ```
  $ make
  ```

  ```
  # make install
  ```

### No go?

The `configure` script checks for the required libraries. If these are not present it will stop.

## Managing home directories

This section will explain how to configure **Autodir** so that user home directories are created on demand. For this purpose the `autohome` module, which deals with specifics of home directory creation, is used.

To load the `autohome` module with **Autodir**, use the `-m` option. For example, `-m /usr/lib/autodir/autohome.so`.

### User/home matching

When an application tries to access a home directory, that home directory is used to check if there is any user with the same user name as the directory name being accessed. If a user name exists, then the home directory is created. Otherwise the message “no such file or directory” is reported back to the application.

### User accounts

`Autohome` does not deal with the creation of user accounts on local systems, in LDAP or in any other database. It only deals with creating home directories once these accounts exist and are imported to the local system from databases like LDAP and NIS.

### Limitations

It is worth mentioning one limitation of the `autohome` module. It expects that user name and home directory are related to each other. For example, for user `user1` the home directory should be `/home/user1` or `/some/directory/name/user1` but not `/some/directory/name/userhome1`. This can be supported but it will be a burden on system resources as each password entry has to be examined from first to last.

### Knowing when not to use autohome

If the existing user password database is such that user home directories are distributed under different base directories, for example `/home/class1/user1`, `/home/class2/user2332`, then `autohome` configuration becomes complicated and is not recommended.

## Base directories for autohome

The next step in the setup procedure is to decide where the *virtual base directory* and *real base directory* for home directory creation will be located.

What are the *virtual base directory* and the *real base directory* in the context of the `autohome` module?

This all depends on how user accounts are created. If a user account created for user name user1 with home directory `/home/user1` then `/home` *will become the virtual base directory*.

Then what is the *real base directory*? This can be any directory. The only thing that you need to keep in mind is that there should be enough space, as all actual files are stored here instead of in the *virtual base directory*.

In most server configurations `/home` is a separate partition. But if `/home` is the *virtual base directory*, then files are not stored in that directory! The solution is not to mount a partition on `/home` but instead mount it somewhere else and make it the *real base directory*.

The **Autodir** option `-d` is used to specify the *virtual base directory*. For example: `autodir -d /home` assumes that `/home` is the *virtual base directory*.

It is somewhat tricky to specify the *real base directory*. The *real base directory* is managed by the `autohome` module so this option must be passed to the module through module suboptions. If the *real base directory* is `/var/autohome` then it is specified with the option `-o` as `-o realpath=/var/autohome`.

## Directory organization

Refer to directory organization under the real base directory for a detailed explanation of this topic.

`autohome` does support this kind of organization. The suboption used to specify the desired directory organization the `level` suboption, for instance: `-o level=2`.

## Misc suboptions for autohome

The suboption `skel` can be used if the skeleton path is not the default value `/etc/skel`: `-o skel=/some/other/dir`.

The suboption `noskel` can be used with `-o` to indicate not to copy any skeleton files to the home directories when these are created.

## Example

First, import your user accounts from a centralized database, for instance from LDAP.

Next, the `autofs` kernel module must be loaded into the Linux kernel. This can be done as described in autofs kernel module section.

If `/home` is to be used for home directories then `/home` will become the *virtual directory*; this is specified to **autodir** with the `-d /home` option.

Assuming that the `autohome` module is located in `/usr/lib/autodir/autohome.so`, this module can be loaded with **autodir** as `-m /usr/lib/autodir/autohome.so`. Note that the full path for the module is given.

The actual location of the real home directories is given with the `realpath` suboption. If it is `/autohome`, the location can be specified as `realpath=/autohome`.

With all these options **autodir** can be started as:

```
# autodir -d /home        \
  -m /usr/lib/autodir/autohome.so   \
  -o 'realpath=/autohome'     \
```

Once **Autodir** is started, initially the `/home` directory will be empty. Whether **Autodir** is working properly can be tested by changing directories to one of the home directories, as the root user or as the owner of the home directory.

## Managing group directories

The `autogroup` module is used for creating group directories on demand for common group access. It can be used with Samba, for example, to dynamically create shared directories for a group of people.

### Check

The `autogroup` module checks for the requested directory in valid groups from the system group database.

### Using autogroup for the creation of home directories

The `autogroup` module can be used to create home directories as well, provided that user private groups exist for each user. This way all group and home directories can be created in one place with one module. However, no skeleton files will be copied and the `autogroup` suboption `nopriv` should not be used.

The `autogroup` configuration is the same as the `autohome` module, but unlike `autohome`, the *virtual base directory* can be placed anywhere and any name can be given to it. It is not dictated by system accounts.

The module `autogroup` can be used with **Autodir** using the `-m` option. For example, `-m /usr/lib/autodir/autogroup.so`.

All suboptions explained in managing home directories are the same for `autogroup`, except `skel` and `noskel`, as these are meaningless for the `autogroup` module. Additionally, there are other suboptions specific to `autogroup`. These are given below.

| | |
|---|---|
| `nopriv` | Some Linux installations use user private groups. If directories for these groups are not to be created, then use this suboption. |

## Autodir options

In this section some of the options to **Autodir** are explained. Backup options are explained in the backup section.

| | |
|---|---|
| `-d` | Specifies the *virtual base directory*. If this path does not exist, it will be created. An absolute path is expected for this option. |
| `-t` | Expiration timeout for *virtual directories*. For more details refer to virtual directory expiration. |
| `-m` | Module to be used with **Autodir**. Currently `autohome` and `autogroup` are available. The full path to the module is expected. |
| `-o` | All suboptions that are to be passed to module are given here. This option passing syntax is similar to that of the **mount** command with its `-o` option. See specific module sections for more info. |
| `-f` | Run in the foreground and log all messages to the console. For debugging purpose and to see how **Autodir** works. |
| `-l` | This option expects a path name to a file in which **Autodir** will write its process id. |
| `-h` | Help about all options supported by **Autodir**. |
| `-v` | Version information about **Autodir**. |

## Backup options

These options are passed to **Autodir** to request backup support.

| | |
|---|---|
| `-b` | This is the main option to specify the backup program path and arguments to it. The path given should be an absolute path, otherwise **Autodir** does not accept it. |
| `-w` | Whenever a *virtual directory* is not used for a period of time, it is assumed inactive and it is unmounted. After unmounting the directory, whether to launch the backup immediately or to wait some more time is decided with this option. It takes arguments in seconds. It is the *minimum* time to wait before starting backup after *virtual directory* expiration. It should not exceed one day. |
| `-p` | This is the priority to be given to the backup process. *This is in the range of 1 to 40 inclusive*. Lower value means higher priority and vice versa. The default value is 30. |
| `-c` | This restricts the maximum number of backup processes at any given time. The default is 150. |

### Using quotes

The argument for the `-b` option is includes the absolute backup program path as well as its own arguments. Therefore it is recommended to use single quotes around this argument.

The `-b` option takes a path to executable file as well as arguments to it. However, the arguments are interpreted for a sequence of %x characters and replaced with predefined strings as follows:

| | |
|---|---|
| `%N` | Replaced with the *virtual directory* name. |

| | |
|---|---|
| `%L` | Replaced with an absolute path to the *real directory*. |
| `%K` | Replaced with host name. |
| Others | Others are fed to `strftime`. See the man page for `strftime` for more information. |

## Examples

```
# autodir -d /home       \
  -m /usr/lib/autodir/autohome.so   \
  -t 1000      \
  -f       \
  -o 'realpath=/autohome,level=1,skel=/etc/skel'  \
  -l /var/run/autodir.pid
```

```
# autodir -d /home      \
  -m /usr/lib/autodir/autohome.so   \
  -t 300      \
  -b '/bin/tar cf /tmp/%N%F.tar %L'  \
  -w 600      \
  -o 'realpath=/tmp/autohome,level=2,noskel' \
  -l /var/run/autodir.pid
```

```
# autodir -d /var/abase/      \
  -m /usr/lib/autodir/autogroup.so  \
  -t 300      \
  -b '/bin/tar cf /tmp/%N%F.tar %L'  \
  -w 86400     \
  -o 'nopriv,nosetgid,realpath=/var/realbase,level=0'
```

## RPM specific

**Autodir** can be installed from rpms as follows:

```
# rpm -ivh autodir-0.28-4.i386.rpm
```

When installed from rpms, two startup scripts are provided: `/etc/rc.d/init.d/autohome` and `/etc/rc.d/init.d/autogroup`. The first one is for starting **Autodir** with the `autohome` module, the second for starting with the `autogroup` module.

The script configuration files `/etc/sysconfig/autohome` and `/etc/sysconfig/autogroup` can be used to specify what options can be passed to **Autodir**.

## Further Information

- **Mailing list for autodir** http://lists.sourceforge.net/mailman/listinfo/intraperson-autodir.
- Official website is at http://www.intraperson.com/autodir/.
- Autofs mailing list http://linux.kernel.org/mailman/listinfo/autofs.

- The Automount HOWTO can be found at http://www.tldp.org
- Autofs Hacking http://www.goop.org/~jeremy/autofs [http://www.goop.org/~jeremy/autofs/].