imagemounter Documentation

Release 1.5.1

Ralph Broenink, Peter Wagenaar

Aug 06, 2017

Contents

imagemounter is a command-line utility and Python package to ease the mounting and unmounting of EnCase, Affuse, vmdk and dd disk images (and other formats supported by supported tools). It supports mounting disk images using xmount (with optional RW cache), affuse, ewfmount, vmware-mount and qemu-nbd; detecting DOS, BSD, Sun, Mac and GPT volume systems; mounting FAT, Ext, XFS UFS, HFS+, LUKS and NTFS volumes, in addition to some less known filesystems; detecting (nested) LVM volume systems and mounting its subvolumes; and reconstructing Linux Software RAID arrays.

In its default mode, imagemounter will try to start mounting the base image on a temporary mount point, detect the volume system and then mount each volume seperately. If it fails finding a volume system, it will try to mount the entire image as a whole if it succeeds in detecting what it actually is.

Note: Not all combinations of file and volume systems have been tested. If you encounter an issue, please try to change some of your arguments first, before creating a new GitHub issue.

Warning: Mounting disks and volumes from unknown sources may pose an important security risk (especially since you probably need to run imagemounter as root).

CHAPTER 1

Contents

Installation

If you need an installation with basic support, you are suggested to run the following commands:

```
apt-get install python-setuptools xmount ewf-tools afflib-tools sleuthkit
pip install imagemounter
imount --check
```

The latter command will list all other packages you could install to expand the capabilities of imagemounter.

Python packages

This package does not require other packages, though the `termcolor` package is recommended if you are using the **`imount`** command line utility with the *`--color`* argument.

If you wish to use `pytsk3` support, you require *python-dev* and *libtsk-dev*. For compilation, the *build-essential* package from your distribution is also required. After that, you can easily install the `pytsk3` package from PyPI (**`pip`** requires the `--pre` flag to allow installing the package).

Other dependencies

This package highly depends on other utilities to be present on your system. For a full installation, you require more tools. You can run `imount --check` to get a full list of all required tools.

A basic installation contains at least one of the mount tools. Highly recommended is also `fsstat`, others are required for specific file system types.

You can install `vmware-mount` by installing VMware Workstation on your system.

Command-line usage

One of the core functionalities of *imagemounter* is the command-line utility **imount** that eases the mounting and unmounting of different types of disks and volumes. In its most basic form, the utility accepts a positional argument pointing to a disk image, disk or volume, e.g.:

```
imount disk.E01
```

Multiple files can be passed to this command, allowing the mounting of volume systems that span multiple disks, which can be useful for those wishing to reconstruct a system that entailed multiple disks or for reconstructing RAID arrays.

By default, **imount** will mount each single volume in `/tmp` and wait until you confirm an unmount operation. Common usage is therefore to keep **imount** running in a separate window and perform other operations in a second window.

Arguments

The **imount** utility requires one (or more) positional arguments and offers the ability to pass several optional arguments.

<image> [<image> ...]
: The positional argument(s) should provide the path(s) to the disk images you want to mount. Many different formats are supported, including the EnCase evidence format, split dd files, mounted hard drives, etc. In the case of split files, you can refer to the folder containing these files.

  If you specify more than one file, all files are considered to be part of the same originating system, which is relevant for the *--reconstruct* command-line option.

Arguments that immediately exit

Some useful facilities.

--help
-h
: Shows a help message and exits.

--version
: Shows the current version and exits.

--check
: Shows which third-party utilities you have installed for a correct functioning of imagemounter.

--unmount
-u
: Option that will try to identify leftover files from previous **imount** executions and try to delete these. This will, for instance, clean leftover `/tmp/im_...` mounts and mountpoints. This command will allow you to review the actions that will be taken before they are done.

  Can be combined with *--casename*, *--mountdir* and *--pretty* to specify which mount points to delete.

CLI behaviour

The next four command-line options alter the behaviour of the **imount** utility, but does not affect the behaviour of the underlying *imagemounter* module.

--wait
-w
Pauses the execution of the program on all warnings.

--keep
-k
Skips the unmounting at the end of the program.

--no-interaction
Never ask for input from the user, implies *--keep*.

--only-mount
Comma-separated list of volume indexes you want to mount. Other volumes are skipped.

--verbose
-v
Show verbose output. Repeat for more verbosity (up to 4).

--color
--no-color
Force toggle colorizing the output. Verbose message will be colored blue, for instance. Requires the `termcolor` package.

Additional features

This command-line option enables an additional and useful feature.

--reconstruct
-r
Attempts to reconstruct the full filesystem tree by identifying the last mountpoint of each identified volume and bindmounting this in the previous root directory. For instance, if volumes have previously been mounted at `/`, `/var` and `/home`; `/var` and `/home` will be bind-mounted in `/`, providing you with a single filesystem tree in the mount location of `/` that is easily traversible.

This only works with Linux-based filesystems and only if `/` can be identified.

Implies `--stats`.

--carve
Carves the filesystem for missing files.

--vshadow
Also mounts volume shadow copies

Mount behaviour

These arguments alter some pieces of the mount behaviour of *imagemounter*, mostly to ease your work.

--mountdir <directory>
-md <directory>
Specifies the directory to place volume mounts. Defaults to a temporary directory.

--pretty
-p
Uses pretty names for volume mount points. This is useful in combination with *--mountdir*, but you should be careful using this option. It does not provide a fallback when the mount point is not available or other issues arise. It can also not be cleaned with `--clean`.

--casename

-cn
: Use to specify the case name, which is used in pretty mounts, but also for the location of the mountdir. Useful if you want to be able to identify the mountpoints later.

--read-write
-rw
: Will use read-write mounts. Written data will be stored using a local write cache.

  Implies `--method xmount`.

Advanced options

While *imagemounter* will try to automatically detect as much as possible, there are some cases where you may wish to override the automatically detected options. You can specify which detection methods should be used and override the volume system and file system types if needed.

--disk-mounter <method>
-m <method>
: Specifies the method to use to mount the base image(s). Defaults to automatic detection, though different methods deliver different results. Available options are *xmount*, *affuse* and *ewfmount* (defaulting to *auto*).

  If you provide *dummy*, the base is not mounted but used directly.

--volume-detector <method>
-d <method>
: Specifies the volume detection method. Available options are *pytsk3*, *mmls*, *parted* and *auto*, which is the default. Though *pytsk3* and *mmls* should in principle deliver identical results, *pytsk3* can be considered more reliable as this uses the C API of The Sleuth Kit (TSK). However, it also requires `pytsk3` to be installed, which is not possible with Py3K.

--vstypes <types>
: Specifies the type of the volume system, defaulting to *detect*. However, detection may not always succeed and valid options are *dos*, *bsd*, *sun*, *mac*, *gpt* and *dbfiller*, though the exact available options depend on the detection method and installed modules on the operating system.

--fstypes <types>
: Specifies the filesystem of a volume to use. Available options include *ext*, *ufs*, *ntfs*, *luks*, *lvm* and *unknown*, with the latter simply mounting the volume without specifying type. See the command-line help for all available volume types.

  Filesystem types are specified for each volume separately. You can use subvolumes, examples including:

```
1=ntfs
2=luks,2.0=lvm,2.0.1=ext
```

  If you wish to specify a fallback to use if automatic detection fails, you can use the special question mark (?) volume index. If you wish to override automatic detection at all for all unspecified volumes, you can use the asterisk (*) volume type. There is no point is specifying both a question mark and an asterisk.

--keys <keys>
: Allows the specification of key information for each volume separately. This is similar to *--fstypes*. The format of the specifc value depends on the volume type.

  For BDE, you can use a single letter, followed by a colon, followed by the value. This leads to the following accepted formats, similar to how the **bdemount** command interprets input:

```
k:full volume encryption and tweak key
p:passphrase
```

```
r:recovery password
s:file to startup key (.bek)
```

For LUKS, you can use a similar format:

```
p:passphrase
f:key-file
m:master-key-file
```

--lazy-unmount
Enables to unmount the volumes and disk lazily when the direct unmounting of the volumes fails.

Advanced toggles

imount has some facilities that automatically detect some types of disks and volumes. However, these facilities may sometimes fail and can be disabled if needed.

--single
--no-single
imount will, by default, try to detect whether the disk that is being mounted, contains an entire volume system, or only a single volume. If you know your volumes are not single volumes, or you know they are, use *--no-single* and *--single* respectively.

Where *--single* forces the mounting of the disk as a single volume, *--no-single* will prevent the identification of the disk as a single volume if no volume system is found.

Python interface

While **imount** heavily utilizes the Python API of *imagemounter*, this API is also available for other classes.

Data structure

The basic structure of *imagemounter* is the *imagemounter.ImageParser* class, which provides access to underlying *imagemounter.Disk* and *imagemounter.Volume* objects. Each file name passed to a new *imagemounter.ImageParser* object results in one *imagemounter.Disk* object. *imagemounter.Volume* objects are created by analysis of the `Disk` object (each volume generates one object, even if it is not mountable), and each *imagemounter.Volume* can have one or more subvolumes.

For instance, a LUKS volume may contain a LVM system that contains a Ext volume. This would create a `Disk` with a `Volume` containing a `Volume` which contains the actual Ext `Volume`. Subvolumes are managed through `imagemounter.VolumeSystem`s, which is used by both the :class:`Volume` and `Disk` classes.

Most operations are managed on a `Volume` level, although individual disk file mounting (and volume detection) is performed on a `Disk` level and reconstruction is performed on a `ImageParser` level. This means the following main parts make up the Python package:

- *imagemounter.ImageParser*, maintaining a list of Disks, providing several methods that are carried out on all disks (e.g. mount) and reconstruct.
- *imagemounter.Disk*, which represents a single disk iamge and can be mounted, and maintain volumes. It is also responsible for maintaining the write cache. Although a Disk is able to detect volumes, a Volume has similar capabilities.

- *imagemounter.Volume*, which can detect its own type and fill its stats, can be mounted, and maintain subvolumes.
- *imagemounter.VolumeSystem*, which is used to manage subvolumes and can detect volumes from a volume system.

All three classes maintain an `init()` method that yields the volumes below it. You should call `clean()` on the parser as soon as you are done; you may also call `unmount()` on separate volumes or disks, which will also unmount all volumes below it. Warning: unmounting one of the RAID volumes in a RAID array, causes the entire array to be unmounted.

Reference

If you utilize the API, you typically only require the *ImageParser* object, e.g.:

```
parser = ImageParser(['/path/to/disk'])
for v in parser.init():
    print v.size
root = parser.reconstruct()
print root.mountpoint
parser.clean()
```

The best example of the use of the Python interface is the **imount** command. The entirety of all methods and attributes is documented below.

ImageParser

class imagemounter.**ImageParser**(*paths=()*, *force_disk_indexes=False*, *casename=None*, *read_write=False*, *disk_mounter=u'auto'*, *volume_detector=u'auto'*, *vstypes=None*, *fstypes=None*, *keys=None*, *mountdir=None*, *pretty=False*, ***args*)

Root object of the *imagemounter* Python interface. This class should be sufficient allowing access to the underlying functions of this module.

Instantiation of this class does not automatically mount, detect or analyse *Disk* s, though it initialises each provided path as a new *Disk* object.

Parameters

- **paths** (*iterable*) – list of paths to base images that should be mounted
- **force_disk_indexes** – if True, a Disk index is always included. If False, will only use Disk indexes if more than 1 Disk is provided to the paths
- **casename** – the name of the case, used when prettifying names
- **read_write** (*bool*) – indicates whether disks should be mounted with a read-write cache enabled
- **disk_mounter** (*str*) – the method to mount the base images with
- **fstypes** (*dict*) – dict mapping volume indices to file system types to use; use * and ? as volume indexes for additional control. Only when ?=none, unknown will not be used as fallback.
- **keys** (*dict*) – dict mapping volume indices to key material
- **mountdir** (*str*) – location where mountpoints are created, defaulting to a temporary location

- **pretty** (*bool*) – indicates whether pretty names should be used for the mountpoints
- **args** – ignored

add_disk(*path*, *force_disk_indexes=True*, ***args*)
Adds a disk specified by the path to the ImageParser.

Parameters

- **path** – The path to the disk volume
- **force_disk_indexes** – If true, always uses disk indexes. If False, only uses disk indexes if this is the second volume you add. If you plan on using this method, always leave this True. If you add a second disk when the previous disk has no index, an error is raised.
- **args** – Arguments to pass to the constructor of the Disk.

init(*single=None*, *swallow_exceptions=True*)
Handles all important disk-mounting tasks, i.e. calls the *Disk.init()* function on all underlying disks. It yields every volume that is encountered, including volumes that have not been mounted.

Parameters

- **single** (*bool* | *None*) – indicates whether the *Disk* should be mounted as a single disk, not as a single disk or whether it should try both (defaults to `None`)
- **swallow_exceptions** – specify whether you want the init calls to swallow exceptions

Return type generator

init_volumes(*single=None*, *only_mount=None*, *swallow_exceptions=True*)
Detects volumes (as volume system or as single volume) in all disks and yields the volumes. This calls *Disk.init_volumes()* on all disks and should be called after *mount_disks()*.

Return type generator

reconstruct()
Reconstructs the filesystem of all volumes mounted by the parser by inspecting the last mount point and bind mounting everything.

Raises NoRootFoundError if no root could be found

Returns the root *Volume*

clean(*remove_rw=False*, *allow_lazy=False*)
Cleans all volumes of all disks (*Volume.unmount()*) and all disks (*Disk.unmount()*). Volume errors are ignored, but returns immediately on disk unmount error.

Parameters

- **remove_rw** (*bool*) – indicates whether a read-write cache should be removed
- **allow_lazy** (*bool*) – indicates whether lazy unmounting is allowed

Returns whether the command completed successfully

Return type boolean

Raises

- **SubsystemError** – when one of the underlying commands fails. Some are swallowed.
- **CleanupError** – when actual cleanup fails. Some are swallowed.

Most methods above, especially *init()*, handle most complicated tasks. However, you may need some more fine-grained control over the mount process, which may require you to use the following methods. Each of these methods passes their activities down to all disks in the parser and return whether it succeeded.

rw_active()
Indicates whether a read-write cache is active in any of the disks.

Return type bool

get_volumes()
Gets a list of all volumes of all disks, concatenating *Disk.get_volumes()* of all disks.

Return type list

get_by_index(*index*)
Returns a Volume or Disk by its index.

mount_disks()
Mounts all disks in the parser, i.e. calling *Disk.mount()* on all underlying disks. You probably want to use *init()* instead.

Returns whether all mounts have succeeded

Return type bool

For completeness, this is a list of all attributes of *ImageParser*:

disks
List of all *Disk* objects.

paths
casename
fstypes
keys
vstypes
mountdir
pretty
See the constructor of *ImageParser*.

Disk

class imagemounter.**Disk**(*parser*, *path*, *index=None*, *offset=0*, *block_size=512*, *read_write=False*, *vstype=u''*, *disk_mounter=u'auto'*, *volume_detector=u'auto'*)
Representation of a disk, image file or anything else that can be considered a disk.

Instantiation of this class does not automatically mount, detect or analyse the disk. You will need the *init()* method for this.

Only use arguments offset and further as keyword arguments.

Parameters

- **parser** (*ImageParser*) – the parent parser
- **path** (*str*) – the path of the Disk
- **index** (*str*) – the base index of this Disk
- **offset** (*int*) – offset of the disk where the volume (system) resides
- **block_size** (*int*) –

- **read_write** (*bool*) – indicates whether the disk should be mounted with a read-write cache enabled
- **vstype** (*str*) – the volume system type to use.
- **disk_mounter** (*str*) – the method to mount the base image with
- **volume_detector** (*str*) – the volume system detection method to use

init(*single=None*, *only_mount=None*, *swallow_exceptions=True*)

Calls several methods required to perform a full initialisation: *mount()*, and `mount_volumes()` and yields all detected volumes.

Parameters **single** (*bool* | *None*) – indicates whether the disk should be mounted as a single disk, not as a single disk or whether it should try both (defaults to `None`)

Return type generator

mount()

Mounts the base image on a temporary location using the mount method stored in *method*. If mounting was successful, *mountpoint* is set to the temporary mountpoint.

If *read_write* is enabled, a temporary read-write cache is also created and stored in *rwpath*.

Returns whether the mounting was successful

Return type bool

detect_volumes(*single=None*)

Generator that detects the volumes from the Disk, using one of two methods:

•Single volume: the entire Disk is a single volume

•Multiple volumes: the Disk is a volume system

Parameters **single** – If *single* is `True`, this method will call `init_single_volumes()`. If *single* is False, only `init_multiple_volumes()` is called. If *single* is None, `init_multiple_volumes()` is always called, being followed by `init_single_volume()` if no volumes were detected.

init_volumes(*single=None*, *only_mount=None*, *swallow_exceptions=True*)

Generator that detects and mounts all volumes in the disk.

Parameters

- **single** – If *single* is `True`, this method will call `init_single_volumes()`. If *single* is False, only `init_multiple_volumes()` is called. If *single* is None, `init_multiple_volumes()` is always called, being followed by `init_single_volume()` if no volumes were detected.
- **only_mount** (*list*) – If set, must be a list of volume indexes that are only mounted.
- **swallow_exceptions** (*bool*) – If True, Exceptions are not raised but rather set on the instance.

unmount(*remove_rw=False*, *allow_lazy=False*)

Removes all ties of this disk to the filesystem, so the image can be unmounted successfully.

Raises

- **SubsystemError** – when one of the underlying commands fails. Some are swallowed.
- **CleanupError** – when actual cleanup fails. Some are swallowed.

The following methods are only required if you want some fine-grained control, typically if you are not using *init()*.

get_disk_type()

rw_active()
Indicates whether anything has been written to a read-write cache.

get_fs_path()
Returns the path to the filesystem. Most of the times this is the image file, but may instead also return the MD device or loopback device the filesystem is mounted to.

Return type str

get_raw_path()
Returns the raw path to the mounted disk image, i.e. the raw `.dd`, `.raw` or `ewf1` file.

Return type str

get_volumes()
Gets a list of all volumes in this disk, including volumes that are contained in other volumes.

The following attributes are also available:

index
Disk index. May be None if it is the only disk of this type.

mountpoint
The mountpoint of the disk, after a call to *mount()*.

rwpath
The path to the read-write cache, filled after a call to *mount()*.

volumes
VolumeSystem of all direct child volumes of this disk, excluding all subvolumes. See *get_volumes()*.

method
Used to store the base mount method. If it is set to `auto`, this value will be overwritten with the actually used mount method after calling *mount()*.

See also the constructor of *Disk*.

parser
paths
offset
read_write
disk_mounter
See the constructor of *Disk*.

Volume

class imagemounter.**Volume**(*disk*, *parent=None*, *index=u'0'*, *size=0*, *offset=0*, *flag=u'alloc'*, *slot=0*, *fstype=u''*, *key=u''*, *vstype=u''*, *volume_detector=u'auto'*)
Information about a volume. Note that every detected volume gets their own Volume object, though it may or may not be mounted. This can be seen through the *mountpoint* attribute – if it is not set, perhaps the `exception` attribute is set with an exception.

Creates a Volume object that is not mounted yet.

Only use arguments as keyword arguments.

Parameters

- **disk** (*Disk*) – the parent disk
- **parent** – the parent volume or disk.
- **index** (*str*) – the volume index within its volume system, see the attribute documentation.
- **size** (*int*) – the volume size, see the attribute documentation.
- **offset** (*int*) – the volume offset, see the attribute documentation.
- **flag** (*str*) – the volume flag, see the attribute documentation.
- **slot** (*int*) – the volume slot, see the attribute documentation.
- **fstype** (*str*) – the fstype you wish to use for this Volume. May be ?<fstype> as a fallback value. If not specified, will be retrieved from the ImageParser instance instead.
- **key** (*str*) – the key to use for this Volume.
- **vstype** (*str*) – the volume system type to use.
- **volume_detector** (*str*) – the volume system detection method to use

init(*only_mount=None*, *swallow_exceptions=True*)

Generator that mounts this volume and either yields itself or recursively generates its subvolumes.

More specifically, this function will call `load_fsstat_data()` (iff *no_stats* is False), followed by *mount()*, followed by a call to *detect_mountpoint()*, after which `self` is yielded, or the result of the *init()* call on each subvolume is yielded

Parameters

- **only_mount** – if specified, only volume indexes in this list are mounted. Volume indexes are strings.
- **swallow_exceptions** – if True, any error occuring when mounting the volume is swallowed and added as an exception attribute to the yielded objects.

init_volume(*fstype=None*)

Initializes a single volume. You should use this method instead of *mount()* if you want some sane checks before mounting.

unmount(*allow_lazy=False*)

Unounts the volume from the filesystem.

Raises

- **SubsystemError** – if one of the underlying processes fails
- **CleanupError** – if the cleanup fails

The following methods offer some more information about the volume:

get_description(*with_size=True*, *with_index=True*)

Obtains a generic description of the volume, containing the file system type, index, label and NTFS version. If *with_size* is provided, the volume size is also included.

get_safe_label()

Returns a label that is safe to add to a path in the mountpoint for this volume.

get_formatted_size()

Obtains the size of the volume in a human-readable format (i.e. in TiBs, GiBs or MiBs).

get_volumes()

Recursively gets a list of all subvolumes and the current volume.

These functions offer access to some internals:

determine_fs_type()
Determines the FS type for this partition. This function is used internally to determine which mount system to use, based on the file system description. Return values include *ext*, *ufs*, *ntfs*, *lvm* and *luks*.

Note: does not do anything if fstype is already set to something sensible.

get_raw_path(*include_self=False*)
Retrieves the base mount path of the volume. Typically equals to *Disk.get_fs_path()* but may also be the path to a logical volume. This is used to determine the source path for a mount call.

The value returned is normally based on the parent's paths, e.g. if this volume is mounted to a more specific path, only its children return the more specific path, this volume itself will keep returning the same path. This makes for consistent use of the offset attribute. If you do not need this behaviour, you can override this with the include_self argument.

This behavior, however, is not retained for paths that directly affect the volume itself, not the child volumes. This includes VSS stores and LV volumes.

mount(*fstype=None*)
Based on the file system type as determined by *determine_fs_type()*, the proper mount command is executed for this volume. The volume is mounted in a temporary path (or a pretty path if *pretty* is enabled) in the mountpoint as specified by *mountpoint*.

If the file system type is a LUKS container or LVM, additional methods may be called, adding subvolumes to *volumes*

Raises

- **NotMountedError** – if the parent volume/disk is not mounted
- **NoMountpointAvailableError** – if no mountpoint was found
- **NoLoopbackAvailableError** – if no loopback device was found
- **UnsupportedFilesystemError** – if the fstype is not supported for mounting
- **SubsystemError** – if one of the underlying commands failed

bindmount(*mountpoint*)
Bind mounts the volume to another mountpoint. Only works if the volume is already mounted.

Raises

- **NotMountedError** – when the volume is not yet mounted
- **SubsystemError** – when the underlying command failed

carve(*freespace=True*)
Call this method to carve the free space of the volume for (deleted) files. Note that photorec has its own interface that temporarily takes over the shell.

Parameters **freespace** (*bool*) – indicates whether the entire volume should be carved (False) or only the free space (True)

Returns string to the path where carved data is available

Raises

- **CommandNotFoundError** – if the underlying command does not exist
- **SubsystemError** – if the underlying command fails
- **NoMountpointAvailableError** – if there is no mountpoint available

- **NoLoopbackAvailableError** – if there is no loopback available (only when volume has no slot number)

detect_volume_shadow_copies()
Method to call vshadowmount and mount NTFS volume shadow copies.

Returns iterable with the *Volume* objects of the VSS

Raises

- **CommandNotFoundError** – if the underlying command does not exist
- **SubSystemError** – if the underlying command fails
- **NoMountpointAvailableError** – if there is no mountpoint available

detect_mountpoint()
Attempts to detect the previous mountpoint if this was not done through `load_fsstat_data()`. This detection does some heuristic method on the mounted volume.

The following details may also be available as attributes:

size
The size of the volume in bytes.

offset
The offset of the volume in the disk in bytes.

index
The index of the volume in the disk. If there are subvolumes, the index is separated by periods, though the exact format depends on the detection method and its format.

slot
Internal slot number of the volume.

flag
Indicates whether this volume is allocated (*alloc*), unallocated (*unalloc*) or a meta volume (*meta*).

block_size
The block size of this volume.

fstype
The volume file system type used internally as determined by *determine_fs_type()*.

key
The key used by some crypto methods.

info
A dict containing information about the volume. Not all keys are always available. Some common keys include:

•fsdescription – A description of the file system type, usually set by the detection method

•lastmountpoint – The last mountpoint of this volume. Set by `load_fsstat_data()` or *detect_mountpoint()* and only available for UFS and Ext volumes

•label – The volume label as detected by `load_fsstat_data()`

•version – The volume version as detected by `load_fsstat_data()`

•statfstype – The volume file system type as detected by `load_fsstat_data()`

•guid – The volume GUID

•volume_group – Used for LVM support

The contents of the info dict are not considered part of a stable API and are subject to change in the future.

mountpoint
The mountpoint of the volume after *mount()* has been called.

loopback
The loopback device used by the volume after *mount()* (or related methods) has been called.

was_mounted
is_mounted
Booleans indicating that the volume has successfully been mounted during its lifetime, and is currently mounted

volumes
parent
volumes contains a *VolumeSystem* of all subvolumes of this volume; *parent* contains the parent volume (if any).

disk
stats
fstypes
pretty
mountdir
args
See the constructor of *Volume*.

VolumeSystem

class imagemounter.**VolumeSystem**(*parent*, *vstype=u''*, *volume_detector=u''*)
A VolumeSystem is a collection of volumes. Every *Disk* contains exactly one VolumeSystem. Each system contains several Volumes, which, in turn, may contain additional volume systems.

Creates a VolumeSystem.

Parameters

- **parent** – the parent may either be a *Disk* or a *Volume* that contains this VolumeSystem.
- **vstype** (*str*) – the volume system type to use.
- **volume_detector** (*str*) – the volume system detection method to use

detect_volumes(*vstype=None*, *method=None*, *force=False*)
Iterator for detecting volumes within this volume system.

Parameters

- **vstype** (*str*) – The volume system type to use. If None, uses *vstype*
- **method** (*str*) – The detection method to use. If None, uses *detection*
- **force** (*bool*) – Specify if you wnat to force running the detection if has_Detected is True.

preload_volume_data()
Preloads volume data. It is used to call internal methods that contain information about a volume.

__iter__()

__getitem__(*item*)

volumes
The list of all volumes in this system.

volume_source
The source of the volumes of this system, either *single* or *multi*.

has_detected
Boolean indicating whether this volume already ran its detection.

vstype
detection
args
See the constructor of *VolumeSystem*.

Unmounter

class imagemounter.**Unmounter**(*casename=None*, *pretty=False*, *mountdir=None*, *allow_greedy=True*, **args*, ***kwargs*)
Allows easy unmounting of left-overs of ImageParser calls.

Instantiation of this class automatically indexes the mountpoints and loopbacks currently on the system. However, in the time between calling any `find` function and actually unmounting anything, the system may change. This can be especially painful when using *preview_unmount()*.

Parameters

- **casename** (*str*) – The casename to be unmounted, see *ImageParser*
- **pretty** (*bool*) – Whether the volumes were mounted using pretty mount, see *Volume*
- **mountdir** (*str*) – The mountdir wheret he volumes were mounted, see *Volume*
- **allow_greedy** (*bool*) – When none of the parameters are specified, by default, a greedy method will try to find as much possible mount points as possible.

preview_unmount()
Returns a list of all commands that would be executed if the *unmount()* method would be called.

Note: any system changes between calling this method and calling *unmount()* aren't listed by this command.

unmount()
Calls all unmount methods in the correct order.

find_bindmounts()
Finds all bind mountpoints that are inside mounts that match the *re_pattern*

find_mounts()
Finds all mountpoints that are mounted to a directory matching *re_pattern* or originate from a directory matching *orig_re_pattern*.

find_base_images()
Finds all mountpoints that are mounted to a directory matching *orig_re_pattern*.

find_volume_groups()
Finds all volume groups that are mounted through a loopback originating from *orig_re_pattern*.

Generator yields tuples of vgname, pvname

find_loopbacks()
Finds all loopbacks originating from *orig_re_pattern*.

Generator yields device names

find_clean_dirs()
Finds all (temporary) directories according to the glob and re patterns that should be cleaned.

unmount_bindmounts()
Unmounts all bind mounts identified by *find_bindmounts()*

unmount_mounts()
Unmounts all mounts identified by *find_mounts()*

unmount_base_images()
Unmounts all mounts identified by *find_base_images()*

unmount_volume_groups()
Unmounts all volume groups and related loopback devices as identified by *find_volume_groups()*

unmount_loopbacks()
Unmounts all loopback devices as identified by *find_loopbacks()*

clean_dirs()
Does a final cleaning of the (temporary) directories according to *find_clean_dirs()*.

re_pattern
The regex pattern used to look for volume mountpoints.

glob_pattern
The glob pattern used to look for volume mountpoints. Always used in conjunction with the *re_pattern*.

orig_re_pattern
The regex pattern used to look for base mountpoints.

orig_glob_pattern
The glob pattern used to look for base mountpoints. Always used in conjunction with the *orig_re_pattern*.

be_greedy
If set, some more volumes and mountpoints may be found.

File and volume system specifics

This section contains specifics on different file systems and volume systems supported by imagemounter. This section is not complete and does not cover every edge case. You are invited to amend this section with additional details.

File systems

ext

ext2, ext3 and ext4 are all supported by imagemounter. All mounts use the ext4 drivers, allowing us to specify the `noload` argument to the mount subsystem.

UFS

Supported are UFS and UFS2. Similar to ext, we use the UFS2 driver to mount both types.

Depending on your OS, you may need to run `modprobe ufs` to enable UFS support in your kernel.

NTFS

For mounting NTFS, you may need the *ntfs-3g* package. The `show_sys_files` option is enabled by default.

This file system type may (accidentally) be detected when a BDE volume should be used instead.

HFS / HFS+

No additional details.

LUKS

For mounting LUKS volumes, the **`cryptsetup`** command is used. At this point, it is not possible to specify more options to the cryptsetup subsystem. To specify keys, use *`--keys`*. The following values are accepted:

```
p:passphrase
f:key-file
m:master-key-file
```

To determine whether a volume is a LUKS volume, `cryptsetup isLuks` is called. This method should return true; if it doesn't, imagemounter will also not be able to mount the volume. The next step is to create a loopback device that is used to call `cryptsetup luksOpen <device> <name>`, where name is of the form `image_mounter_luks_<number>`. Additional details of the volume are extracted by using `cryptsetup status`. The actual dd image of the volume is mounted in `/dev/mapper/<name>` by the OS.

The LUKS volume will get a subvolume at index 0 with the file system description `LUKS Volume`. When this volume is a LVM volume that is not be properly recognized by imagemounter, you could use something like the following to amend this:

```
imount image.E01 --fstypes=1=luks,1.0=lvm,1.0.0=ext --keys=1=passphrase
```

LUKS volumes are automatically unmounted by ending the script normally, but can't be unmounted by *`--unmount`*.

The LUKS volume type may not be automatically detected in some cases.

BDE (Bitlocker)

Bitlocker Drive Encrypted volumes are mounted using the **`bdemount`** command. imagemounter allows you to provide the crypto material to the command using *`--keys`*. Examples of valid commands are:

```
imount image.E01 --fstypes=2=bde --keys=2=p:passphrase
imount image.E01 --fstypes=2=bde --keys=2=r:recovery_password
```

See the manpage for **`bdemount`** for all valid arguments that can be passed to the subsystem (crypto material is provided by replacing `<key>:<value>` with `-<key> <value>`).

The BDE volume will get a subvolume at index 0 with the file system description `BDE Volume`. imagemounter should normally correctly detect this subvolume to be a NTFS volume.

BDE volumes are automatically unmounted by ending the script normally, but in some cases may not be properly unmounted by *`--unmount`*.

The BDE volume type may not be properly recognized and may instead by recognized as NTFS volume. You can override this by explicitly stating the volume type as in the examples above.

LVM

LVM systems host multiple volumes inside a single volume. imagemounter is able to detect these volumes on most occassions, though it may not always be possible to detect the file system type of the volumes inside the LVM.

Mounting an LVM is done by mounting the volume to a loopback device and running `lvm pvscan`. This should return a list of all LVMs on the system, but by matching the mount point of the base image, the scirpt should be able to identify the volume group name. This name is then used to enable the LVM by running `vgchange -a y <name>`. Using `lvdisplay <name>`, the volumes inside the volume group are extracted. The volume themselves are found at the LV Path provided by this command.

Volumes inside a LVM are given the FS description `Logical Volume`. The file system types should be recognized properly by the detection methods, and otherwise `unknown` should work, but otherwise you could explicitly specify the file system type as follows:

```
imount image.E01 --fstypes=1=lvm,1.0=ext
```

Please note that many Linux based operating systems will try to mount LVMs for you. Although imagemounter tries to circumvent this automation, if you are unable to properly unmount, you should try to unmount through the interface of your OS first. Another useful command is `vgchange -a n` to disable all LVMs currently active (only use if you are not using a LVM for your own OS!).

Unmounting LVMs is supported both by properly closing from the script as well as by using *`--unmount`*

Linux Software RAID

Linux RAID volume support is provided by the `mdadm` command. A volume is added to a RAID array incrementally; the `mdadm` command is responsible for adding the volume to the correct array. The location of the RAID array is captured by imagemounter so it can be unmounted again. A subvolume will be added with the description `RAID volume` at index 0.

If the RAID volume can not be started directly after adding the volume, mounting will have succeeded, but the mountpoint will not be available yet. When another volume is added to the same RAID array, it will get the same (identical) subvolume as the original RAID volume. You should not mount it again. `init` will take care of both cases for you.

Warning: If, for any reason, you have multiple RAID volumes in the same RAID array, unmounting one of the volumes will also immediately unmount all other RAID volumes in the same array. Because of this, you should ensure that you keep all RAID volumes mounted until you are done building and examining a specific array.

RAID volumes are sometimes correctly detected, but there are also cases where the volume appears to *successfully* mount as another volume type. You should be very careful with this.

Note: A disk leveraging full disk RAID can be mounted as a single volume with the RAID filesystem type.

XFS

XFS is supported through the *xfsprogs* package.

ISO (ISO9660)

No additional details.

UDF

No additional details.

FAT

FAT volumes, independent of type, are mounted through the exFAT driver.

VMFS

VMFS is supported through the *vmfs-tools* package. Mounting is performed by finding a loopback device and using the `vmfs-fuse` command to mount this loopback on the mountpoint.

SquashFS

SquashFS is supported through the *squashfs-tools* package.

JFFS2

JFFS2 is supported through the *mtd-tools* package. JFFS2 is sometimes used by BIOS images and the like.

The following commands are executed to open a JFFS2 image, where `<size>` is given a buffer of 1.2 times the size of the volume:

```
modprobe -v mtd
modprobe -v jffs2
modprobe -v mtdram total_size=<size> erase_size=256
modprobe -v mtdblock
dd if=<path> of=/dev/mtd0
mount -t jffs2 /dev/mtdblock0 <mountpoint>
```

Warning: This filesystem type may not work while mounting multiple images of the same type at the same time.

Unmounting for this filesystem type is not fully supported.

CramFS

No additional details.

Minix

No additional details.

Dir

The dir filesystem type is not an actual mount type, but is used by imagemounter to indicate directories. This can be used in conjunction with the AVFS mount method, but basically just symlinks a directory to the mount location. It is provided for abstraction purposes.

Unknown

The unknown filesystem type is not an actual mount type, but used by imagemounter to indicate that the volume should be mounted without specifying the volume type. This is less specific and does not work in most cases (since it lacks the ability to provide additional options to the mount subsystem) but may result in the volume actually being able to be used.

The unknown filesystem type is used as fallback by default, and is for instance used if no specific volume type is provided by any of the detection methods other than 'Linux'. If you wish to override this default, and choose skipping mounting instead, you can also use the `none` filesystem type:

```
imount image.dd --fstypes=?=none
```

Volume systems

DOS (MBR)

In some cases, the DOS volume system is recognized as either a DOS or a GPT volume system. This appears to be a bug in The Sleuth Kit used by some detection methods. imagemounter works around this by choosing in this case for the GPT volume system and will log a warning. In the case that this is not the right choice, you must use `--vstype` to explicitly provide the correct volume system.

In the case you have picked the wrong volume system, you can easily spot this. If you see `GPT Safety Partition` popping up, you should have chosen GPT.

GPT

See the DOS/MBR volume system.

BSD

No additional details.

Sun

No additional details.

MAC

No additional details.

Detect

Lets the subsystem automatically decide the correct volume system type.

Release notes

We try to reduce backwards compatibility breakage only to major version releases, i.e. X.0.0. Minor releases (1.X.0) may include new features, whereas patch releases (1.0.X) will generally be used to fix bugs. Not all versions have followed these simple rules to date (and sometimes new features may creep up in patch releases), but we try to adhere them as much as possible :).

Release history

3.0.1 (2017-04-08)

- Add support for qcow2 (contributed by Jarmo van Lenthe)
- Allow use of lowercase e01 file extension when mounting a directory in imount CLI (contributed by sourcex)
- Add ADS support for NTFS volumes (contributed by Patrick Leedom)
- Ability to lazily call fusermount -uz when unmounting (contributed by Patrick Leedom)
- Fix regression in mounting LV volumes; the path was incorrectly detected in `get_raw_path()` for these volumes.
- Fix regression in detection of single volumes that would be detected as DOS/MBR based on file type.

3.0.0 (2016-12-11)

This new release includes several backwards-incompatible changes, mostly because features were removed from the public API or have been renamed to obtain a more consistent API.

It was released after a long time of development, and does not even contain all features that were originally planned to go into this release, but it contains some important bugfixes that warranted a release.

New major features:

- Add volume shadow copy support for NTFS
- Add BDE (Bitlocker) support
- Addition of `--keys` CLI argument and corresponding argument to Volume class, allowing to specify key material for crypto mounts, supporting both BDE and LUKS.
- (Experimental) support for volume systems inside a volume. This is useful when e.g. a LVM volume contains in itself a MBR.
- A split between detection and initialization of volumes has been made. The basic way to access volumes as calling `init()`, but that mounted all volumes immediately. Now, `detect_*` methods have been added.
- Support `blkid` to retrieve FS type info
- Support for Linux RAID volumes
- (Still in development) interactive console, which will eventually become the primary means to interact with imagemounter.

Bugfixes:

- Calling `init()` will not automatically mount the volume when it is not `alloc`.
- Fix a bug where `.e01` files (lowercase) would not be recognized as Encase
- Fixed support for newer versions of `mmls`

- Fixed support for pytsk3 under Python 3 (contributed by insomniacslk)
- Fixed support for EnCase v7 (EX01) image files (contributed by pix)
- Improved detection of several volume types
- `index` is now always `str`
- `Volume.size` is now always `int`
- Improved the unmounter with generic loopback support

Removed and modified features:

- Stopped providing `None` and `False` results when things go wrong for most methods. Instead, numerous exceptions have been added. These exceptions should be catched instead, or when using `mount_volumes` or `init`, you can specify `swallow_exceptions` (default) to restore previous behaviour. This is useful, since iteration will continue regardless of exceptions.
- Moved the attributes `fstypes`, `vstypes`, `keys`, `mountdir` and `pretty` to the `ImageParser` instance, so it does not need to get passed down through the `*args` hack anymore. For instance, `fstypes` has been moved; the dict will be inspected upon Volume instantiation and stored in the `fstype` attribute. Other arguments and attributes have been eliminated completely, or have been replaced by arguments to specific methods.
- Added an intermediary class `VolumeSystem`. Both `Volume` and `Disk` now use this (iterable) base class in their `volumes` attribute. If you relied on `volumes` being a `list`, you should now use `list(volumes)`. If you relied on indexing of the attribute, you could now also use `disk[0]` or `volume[0]` for finding the correct volume index. `volume_source` was moved to this class, as have `vstype` and `volume_detector`.
- **Changes to the CLI:**
  - Removed `--fsforce` and `--fsfallback`. Use `*` and `?` as fstypes instead for the same effect. This should make the CLI more sensible, especially regarding the `--fsforce` argument. The default FS fallback is still `unknown`, which can only be overridden by specifying `--fstypes=?=none`. (You can now specify `--fstypes=TYPE`, which equals to `--fstypes=*=TYPE`)
  - Removed `--stats` and `--no-stats`. These only complicated things and `fsstat` has been working fine for years now.
  - Removed `--raid` and `--no-raid` (due to Volume RAID support)
  - Removed `--disktype` and `--no-disktype`.
  - Renamed `--method` to `--disk-mounter`.
  - Renamed `--detection` to `--volume-detector`.
  - Renamed `--vstype` to `--vstypes`, now accepting a dict, similar to `--fstypes`
  - Moved the `imount.py` file into a new `cli` module, where also a new experimental shell-style CLI is under development.
- **Changes specific to `ImageParser`:**
  - Added `add_disk` and made `paths` optional in constructor.
  - Added indexing of the *ImageParser* and added `get_volume_by_index` method.
  - Removed `mount_single_volume` and `mount_multiple_volumes`. Use `init_volumes` instead, or use a custom loop for more control.
  - Dropped support for a single string argument for `paths` in `__init__`. Additionally, dropped the `paths` attribute entirely.
- **Changes specific to `Disk`:**

  - Renamed `method` to `disk_mounter` (see also CLI)
  - Removed `name`, `avfs_mountpoint` and `md_device` from public API.
  - Removed Linux RAID Disk support. Instead, mount as a single volume, with the type of this volume being RAID. This greatly simplifies the `Disk` class. (This means that `loopback` has also been dropped from Disk)
  - Added `detect_volumes` method, which can be used to detect volumes.
  - Removed most `mount_*` methods. Moved `mount_volumes` to `init_volumes`. Functionality from the other methods can be restored with only a few lines of code.
  - Removed the need for the rather obsure `multifile` attribute of `mount`. Only `xmount` actually required this, so we just implicitly use it there.
  - Moved the `type` attribute to a method `get_disk_type`.
- **Changes specific to `Volume`:**
  - Renamed `get_raw_base_path` to `get_raw_path`
  - Renamed `get_size_gib` to `get_formatted_size`
  - Removed `get_magic_type`, `fill_stats`, `open_jffs2`, `find_lvm_volumes` and `open_luks_container` from public API.
  - Removed the `*_path`, `carvepoint` and `bindmountpoint` attributes from the public API. For `carvepoint`, the `carve` method now returns the path to the carvepoint. All data has been moved to the private `_paths` attribute. The `mountpoint` and `loopback` attributes are kept.
  - Removed `fsforce` and `fsfallback` arguments and attributes from Volume (see also CLI)
  - Added `init_volume`, which only mounts the single volume. It is used by `init` and the preferred way of mounting a single volume (instead of using `mount`)
  - Moved several attributes of `Volume` to a new `info` attribute, which is publicly accessible, but its contents are not part of a stable public API.
- **Changes specific to `VolumeSystem` (if you consider it on par with the functionality moved from Disk):**
  - Renamed `detection` to `volume_detector` (see also CLI)
  - Added a `VolumeSystem.detect_volumes()` iterable, which is the basic functionality of this class.
  - Moved `mount_single_volume` code from `Disk` to this class, adding the `single` volume detection method. The directory detection method has been incorporated in this new method.
- Dropped support for Python 3.2, since everyone seems to be doing that these days.

2.0.4 (2016-03-15)

- Add HFS+ support

2.0.3 (2015-08-02)

- Remove error prefix (`[-]`) from some of the warnings
- Do not warn about using unknown as fsfallback anymore

- Also work properly with the `python-magic` system package (in addition to the totally different `python-magic` PyPI package)
- *vmware-mount* Add `-r` to vmware-mount for readonly mounts
- *ntfs* Add force to mount options

2.0.2 (2015-06-17)

- Bugfix in *`--check`* regarding the `python-magic` module
- *vmware-mount* Fix vmware-mount support

2.0.1 (2015-06-17)

- Changed the default `fsfallback` to `unknown`, instead of `none`.

2.0.0 (2015-06-17)

- Introduce support for XFS, ISO, JFFS2, FAT, SquashFS, CramFS, VMFS, UDF and Minix (cheers martinvw!)
- Add ability to read the disk GUID using disktype, and read the filesystem magic for better detection of filesystems (cheers martinvw!)
- Add support for 'mounting' directories and compressed files using avfs (cheers martinvw!)
- Add support for detecting volumes using parted
- Introduce facility to carve filesystems for removed files, even in unallocated spaces
- Add *`--no-interaction`* for scripted access to the CLI
- Add *`--check`* for access to an overview of all dependencies of imagemounter
- Add *`--casename`* (and corresponding Python argument) to easily recognize and organize multiple mounts on the same system
- Change `--clean` to *`--unmount`*, supporting arguments such as *`--mountdir`* and *`--pretty`*, and made the code more robust and easier to read and extend
- Detect terminal color support and show color by default
- BSD is now called UFS
- `--stats` is now the default in the Python script
- NTFS mount now also shows the system files by default
- Do not stop when not running as root, but warn and probably fail miserably later on
- `fstype` now stores the detected file system type, instead of the `fstype` as determined by `fill_stats()`
- Logging now properly uses the Python logging framework, and there are now 4 verbosity levels
- Changes to how the pretty names are formatted
- Some Py2/Py3 compatibility fixes

1.5.3 (2015-04-08)

- Add support for `vmware-mount`

1.5.2 (2015-04-08)

- Ensure `Volume.size` is always int
- Fixed a GPT/DOS bug caused by TSK
- Add FAT support

1.5.1 (2014-05-22)

- Add disk index for multi-disk mounts

1.5.0 (2014-05-14)

- Add support for volume detection using mmls
- Python 3 support
- Bugfix in luksOpen

1.4.3 (2014-04-26)

- Experimental LUKS support

1.4.2 (2014-04-26)

- Bugfix that would prevent proper unmounting

1.4.1 (2014-02-10)

- Initial Py3K support
- Included script is now called `imount` instead of `mount_images`

1.4.0 (2014-02-03)

- `Disk` is now a seperate class
- Some huge refactoring
- Numerous bugfixes, including resolving issues with unmounting
- Rename `image_mounter` to `imagemounter`
- Remove `mount_images` alias

1.3.1 (2014-01-23)

- More verbosity with respect to failing mounts

1.3.0 (2014-01-23)

- Add support for single volume mounts
- Add support for dummy base mounting
- Add support for RAID detection and mounting

1.2.9 (2014-01-21)

- Improve support for some types of disk images
- Some changes in the way some command-line arguments work (removed `-vs`, `-fs` and `-fsf`)

1.2.8 (2014-01-08)

- Make :option:‘–stats the default
- Print the volume size and offset in verbose mode in the CLI
- Add imount as command line utility name

1.2.7 (2014-01-08)

- Add *`--keep`*

1.2.6 (2014-01-08)

- Use fallback commands for base image mounting if the normal one fails
- Add multifile option to Volume to control whether multifile argument passing should be attempted
- Fix error in backwards compatibility of mount_partitions
- Copy the label of a volume to the last mountpoint if it looks like a mountpoint

1.2.5 (2014-01-07)

- Ability to automatically detect the mountpoint based on files in the filesystem

1.2.4 (2013-12-16)

- Partition is now Volume
- Store the volume flag (alloc, unalloc, meta)

1.2.3 (2013-12-10)

- Add support for pretty mount point names

1.2.2 (2013-12-09)

- Fix issue where ‘extended’ is detected as ext (again)

1.2.1 (2013-12-09)

- Fix issue where 'extended' is detected as ext
- ImagePartition is now Volume

1.2.0 (2013-12-05)

- ImagePartition is now responsible for mounting and obtaining its stats, and detecting lvm volumes
- LVM partitions are now mounted using this new mount method
- Utilize the partition size for disk size, which is more reliable
- Renamed ImagePartition to Volume (no backwards compatibility is provided)
- Add unknown mount type, for use with `--fstype`, which mounts without knowing anything
- Support mounting a directory containing *.001/*.E01 files

1.1.2 (2013-12-05)

- Resolve bug with respect to determining free loopback device

1.1.1 (2013-12-04)

- Improve `--clean` by showing the commands to be executed beforehand

1.1.0 (2013-12-04)

- Do not add sudo to internal commands anymore
- `--loopback` is removed, detects it automatically now
- `--clean` is added; will remove all traces of an unsuccessful previous run

1.0.4 (2013-12-03)

- Add the any vstype
- Fix some errors in the `mount_images` script

1.0.3 (2013-12-02)

- Support forcing the fstype
- Improved LVM support
- Added some warnings to CLI

1.0.2 (2013-11-28)

- Improved NTFS support

1.0.1 (2013-11-28)

- `command_exists` now works properly

1.0.0 (2013-11-28)

- Now includes proper setup.py and versioning
- Add support for reconstructing the filesystem using bindmounts
- More reliable use of fsstat
- Overhauled Python API with more transparency and less CLI requirements
  - Store yielded information in a ImagePartition
  - Remove dependency on args and add them to the class explicitly
  - Do not depend on user interaction or CLI output in ImageParser or util, but do CLI in `__main__`
- Support for LVM
- Support for ewfmount
- Retrieve stats more reliably
- New CLI arguments:
  - Colored output with *`--color`*
  - Wait for warnings with *`--wait`*
  - Support for automatic method with `--method=auto`
  - Specify custom mount dir with *`--mountdir`*
  - Specify explicit volume system type with `--vstype`
  - Specify explicit file system type with `--fstype`
  - Specify loopback device with `--loopback` (required by LVM support)

Python Module Index

i

Index

Symbols

K

L

M

O

P

R

S

U

V

W