imagemounter Documentation

Release 1.5.1

Ralph Broenink, Peter Wagenaar

March 15, 2016

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 and vmware-mount; 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 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.

CHAPTER 1

Important notes

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.

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!).

With *imount –clear* you can clear MOST temporary files and mounts, though this will not clean everything. If you used *–pretty* this tool can't do anything for you. It is therefore recommended to first try and mount your image without *–pretty*, to allow you to easily clean up if something crashes.

CHAPTER 2

Contents

2.1 Installation

If you need an installation with full support, including all optional dependencies, you could use the following commands:

```
apt-get install python-setuptools xmount ewf-tools afflib-tools sleuthkit lvm2 mdadm cryptsetup
pip install imagemounter
```

2.1.1 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).

2.1.2 Other dependencies

This package highly depends on other utilities to be present on your system. For a full installation, you require the following tools:

- Mount tools
  - **`xmount`**
  - **`ewfmount`**, part of *ewf-tools* package, see note below
  - **`affuse`**, part of *afflib-tools* package
  - **`vmware-mount`**, part of VMware Workstation
- Volume detection
  - **`mmls`**, part of *sleuthkit* package
  - `pytsk3`
  - **`disktype`**
  - **`file`**, part of *libmagic1* package
  - `python-magic`

- Statistics, e.g. last mountpoint of volumes
  - **fsstat**, part of *sleuthkit* package
- LVM volumes
  - **lvm** et al, all part of *lvm2* package
- RAID arrays
  - **mdadm**
- LUKS volumes
  - **cryptsetup**
- Compressed dd-images, iso's et al
  - **mountavfs**, part of *avfs* package
- Other Filesystems mount tools:
  - **vmfs-fuse**, part of *vmfs-tools* package
  - **mount.jffs2**, `mtd`, all part of *mtd-tools* package
  - **mount.squashfs**, part of *squashfs-tools* package
  - **mount.xfs**, part of *xfsprogs* package
  - **mount.cramfs**, part of standard (Ubuntu) installation
  - **mount.minix**, part of standard (Ubuntu) installation
  - **mount.vfat**, part of standard (Ubuntu) installation
  - **mount.iso9660**, standard (Ubuntu) installation

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

2.1.3 ewfmount on Ubuntu 13.10

Due to a bug with *ewf-tools* in Ubuntu <=13.10, it may be that **ewfmount** is not properly provided. This bug has been resolved in Ubuntu 14.04. If you are using Ubuntu 13.10, you can install *ewf-tools* with **ewfmount** as follows:

1. Download a recent build of *ewf-tools* from https://launchpad.net/ubuntu/+source/libewf/20130416-2ubuntu1 (choose your arch under 'Builds' and download all deb files under 'Built files')
2. Execute `sudo apt-get install libbfio1`
3. Execute `sudo dpkg -i ewf-tools_* libewf2_*`

2.2 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.

2.2.1 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*.

--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.

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.

--method <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.

--detection <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.

--vstype <type>
: 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.

--fsfallback <type>
: Specifies a fallback option for the filesystem of a volume if automatic detection fails. 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.

--fsforce
: Forces the use of the filesystem type specified with *--fsfallback* for all volumes. In other words, disables the automatic filesystem detection.

--fstypes <types>
: Allows the specification of filesystem type for each volume separately. You can use subvolumes, examples including:

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

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.

--stats
--no-stats
: With stats rerieval is enabled, additional volume information is obtained from the **fsstat** command. This could possibly slow down mounting and may cause random issues such as partitions being unreadable. However, this additional information will probably include some useful information related to the volume system and is required for commands such as *--reconstruct*.

  Stats retrieval is enabled by default, but *--stats* can be used to override *--no-stats*.

--raid

--no-raid
By default, a detection is ran to detect whether the volume is part of a (former) RAID array. You can disable the RAID check with *--no-raid*. If you provide both *--raid* and *--no-raid*, `raid` wins.

--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.

--disktype
--no-disktype
Forcibly enable or disable the use of **disktype** for additional disk information.

2.3 Python interface

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

2.3.1 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`.

Most operations are managed on a `Volume` level, although RAIDs (and volume detection) are managed 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, added to RAID, and detect and maintain volumes. It is also responsible for maintaining the write cache.
- *imagemounter.Volume*, which can detect its own type and fill its stats, can be mounted, and detect LVM (sub)volumes.

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.

2.3.2 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.

class imagemounter.**ImageParser**(*paths*, *casename=None*, ***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
- **casename** – the name of the case, used when prettifying names
- **args** – arguments that should be passed down to *Disk* and *Volume* objects

init(*single=None*, *raid=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)
- **raid** (*bool*) – indicates whether RAID detection is enabled

Return type generator

reconstruct()

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

Returns None on failure, or the root *Volume* on success

clean(*remove_rw=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

Returns whether the command completed successfully

Return type boolean

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

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

mount_raid()
Creates a RAID device and adds all devices to the RAID array, i.e. calling *Disk.add_to_raid()* on all underlying disks. Should be called before *mount_disks()*.

Returns whether all disks were successfully added

Return type bool

mount_single_volume()
Detects the full disk as a single volume and yields the volume. This calls *Disk.mount_single_volume()* on all disks and should be called after *mount_disks()*

Return type generator

mount_multiple_volumes()
Detects volumes in all disks (all mounted as a volume system) and yields the volumes. This calls *Disk.mount_multiple_volumes()* on all disks and should be called after *mount_disks()*.

Return type generator

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

Return type generator

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

disks
List of all *Disk* objects.

paths
casename
args
See the constructor of *ImageParser*.

class imagemounter.**Disk**(*parser*, *path*, *offset=0*, *vstype=u'detect'*, *read_write=False*, *method=u'auto'*, *detection=u'auto'*, *multifile=True*, *index=None*, *mount_directories=True*, ***args*)
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.

Parameters

- **parser** (*ImageParser*) – the parent parser
- **offset** (*int*) – offset of the disk where the volume (system) resides
- **vstype** (*str*) – the volume system type
- **read_write** (*bool*) – indicates whether the disk should be mounted with a read-write cache enabled
- **method** (*str*) – the method to mount the base image with

- **detection** (*str*) – the method to detect volumes in the volume system with
- **multifile** (*bool*) – indicates whether *mount()* should attempt to call the underlying mount method with all files of a split file when passing a single file does not work
- **index** (*str*) – the base index of this Disk
- **mount_directories** (*bool*) – indicates whether directories should also be 'mounted'
- **args** – arguments that should be passed down to *Volume* objects

init(*single=None*, *raid=True*, *disktype=True*)

Calls several methods required to perform a full initialisation: *mount()*, *add_to_raid()* 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)
- **raid** (*bool*) – indicates whether RAID detection is enabled
- **disktype** (*bool*) – indicates whether disktype data should be loaded and used

Return type generator

unmount(*remove_rw=False*)

Removes all ties of this disk to the filesystem, so the image can be unmounted successfully. Warning: this method will destruct the entire RAID array in which this disk takes part.

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

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.

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

mount_volumes(*single=None*)

Generator that detects and mounts all volumes in the disk.

If *single* is `True`, this method will call `mount_single_volumes()`. If *single* is False, only *mount_multiple_volumes()* is called. If *single* is None, *mount_multiple_volumes()* is always called, being followed by *mount_single_volume()* if no volumes were detected.

mount_multiple_volumes()
Generator that will detect volumes in the disk file, generate *Volume* objects based on this information and call *init()* on these.

mount_single_volume()
Mounts a volume assuming that the mounted image does not contain a full disk image, but only a single volume.

A new *Volume* object is created based on the disk file and *init()* is called on this object.

This function will typically yield one volume, although if the volume contains other volumes, multiple volumes may be returned.

is_raid()
Tests whether this image (was) part of a RAID array. Requires **mdadm** to be installed.

add_to_raid()
Adds the disk to a central RAID volume.

This function will first test whether it is actually a RAID volume by using *is_raid()* and, if so, will add the disk to the array via a loopback device.

Returns whether the addition succeeded

The following attributes are also available:

name
Pretty name of the disk.

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

volume_source
The source of the volumes of this disk, either *single* or *multi*, filled after a call to *mount_volumes()*.

loopback
md_device
Used for RAID support.

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
path
offset
vstype
read_write
detection
multifile
args
See the constructor of *Disk*.

class imagemounter.**Volume**(*disk=None*, *stats=True*, *fsforce=False*, *fsfallback=u'unknown'*, *fstypes=None*, *pretty=False*, *mountdir=None*, ***args*)

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.

Parameters

- **disk** (*Disk*) – the parent disk
- **stats** (*bool*) – indicates whether *init()* should try to fill statistics
- **fsforce** (*bool*) – indicates whether the file system type in *fsfallback* should be used for all file systems
- **fsfallback** (*str*) – the file system type to use when automatic detection fails
- **fstypes** (*dict*) – dict mapping volume indices to file system types to (forcibly) use
- **pretty** (*bool*) – indicates whether pretty names should be used for the mountpoints
- **mountdir** (*str*) – location where mountpoints are created, defaulting to a temporary location
- **args** – additional arguments

init(*no_stats=False*)

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

More specifically, this function will call *fill_stats()* (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

unmount()

Unounts the volume from the filesystem.

The following methods offer some more information about the volume:

get_description(*with_size=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_size_gib()

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*.

get_raw_base_path()

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.

mount()

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, *open_luks_container()* is called only. If it is a LVM volume, *find_lvm_volumes()* is called after the LVM has been mounted. Both methods will add subvolumes to *volumes*

Returns boolean indicating whether the mount succeeded

bindmount(*mountpoint*)
Bind mounts the volume to another mountpoint. Only works if the volume is already mounted. Note that only the last bindmountpoint is remembered and cleaned.

Returns bool indicating whether the bindmount succeeded

fill_stats()
Using **fsstat**, adds some additional information of the volume to the Volume.

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

find_lvm_volumes(*force=False*)
Performs post-mount actions on a LVM. Scans for active volume groups from the loopback device, activates it and fills *volumes* with the logical volumes.

If *force* is true, the LVM detection is ran even when the LVM is not mounted on a loopback device.

open_luks_container()
Command that is an alternative to the *mount()* command that opens a LUKS container. The opened volume is added to the subvolume set of this volume. Requires the user to enter the key manually.

Returns the Volume contained in the LUKS container, or None on failure.

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.

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

fsdescription
A description of the file system type.

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

label
The volume label as detected by *fill_stats()*.

version
The volume version as detected by *fill_stats()*.

statfstype
The volume file system type as detected by *fill_stats()*.

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

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

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

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

exception
Contains an exception that occurred during a call to *mount()*.

was_mounted
Boolean indicating that the volume has successfully been mounted during its lifetime.

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

volume_group
lv_path
Attributes used for LVM support

luks_path
Attribute used for LUKS support

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

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_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()*

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.

2.4 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 :).

2.4.1 Release history

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 pip 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 –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)

i

Symbols

L

M

N

O

P

R

S

U

V

W