Name

mtools.conf - mtools configuration files

Description

This manual page describes the configuration files for mtools. They are called `'/etc/mtools.conf'` and `'˜/.mtoolsrc'`. If the environmental variable `MTOOLSRC` is set, its contents is used as the filename for a third configuration file. These configuration files describe the following items:

* Global configuration flags and variables

* Per drive flags and variables

Location of the configuration files

`'/etc/mtools.conf'` is the system-wide configuration file, and `'˜/.mtoolsrc'` is the user's private configuration file.

On some systems, the system-wide configuration file is called `'/etc/default/mtools.conf'` instead.

General configuration file syntax

The configuration files is made up of sections. Each section starts with a keyword identifying the section followed by a colon. Then follow variable assignments and flags. Variable assignments take the following form:
name=value

Flags are lone keywords without an equal sign and value following them. A section either ends at the end of the file or where the next section begins.

Lines starting with a hash (#) are comments. Newline characters are equivalent to whitespace (except where ending a comment). The configuration file is case insensitive, except for item enclosed in quotes (such as filenames).

Default values

For most platforms, mtools contains reasonable compiled-in defaults for physical floppy drives. Thus, you usually don't need to bother with the configuration file, if all you want to do with mtools is to access your floppy drives. On the other hand, the configuration file is needed if you also want to use mtools to access your hard disk partitions and DOSEMU image files.

Global variables

Global flags may be set to 1 or to 0.

The following global flags are recognized:

`MTOOLS_SKIP_CHECK`
: If this is set to 1, mtools skips most of its sanity checks. This is needed to read some Atari disks which have been made with the earlier ROMs, and which would not be recognized otherwise.

`MTOOLS_FAT_COMPATIBILITY`
: If this is set to 1, mtools skips the fat size checks. Some disks have a bigger FAT than they really need to. These are rejected if this option is not set.

`MTOOLS_LOWER_CASE`
: If this is set to 1, mtools displays all-upper-case short filenames as lowercase. This has been done to allow a behavior which is consistent with older versions of mtools which didn't know about the case bits.

`MTOOLS_NO_VFAT`
: If this is set to 1, mtools won't generate VFAT entries for filenames which are mixed-case, but otherwise legal dos filenames. This is useful when working with DOS versions which can't grok VFAT long names, such as FreeDOS.

`MTOOLS_DOTTED_DIR`
: In a wide directory, prints the short name with a dot instead of spaces separating the basename and the extension.

`MTOOLS_NAME_NUMERIC_TAIL`
: If this is set to one (default), generate numeric tails for all long names (~1). If set to zero, only generate numeric tails if otherwise a clash would have happened.

`MTOOLS_TWENTY_FOUR_HOUR_CLOCK`
: If 1, uses the European notation for times (twenty four hour clock), else uses the UK/US notation (am/pm)

`MTOOLS_LOCK_TIMEOUT`
: How long, in seconds, to wait for a locked device to become free. Defaults to 30.

Example: Inserting the following line into your configuration file instructs mtools to skip the sanity checks:

```
MTOOLS_SKIP_CHECK=1
```

Global variables may also be set via the environment:

```
export MTOOLS_SKIP_CHECK=1
```

Global string variables may be set to any value:

`MTOOLS_DATE_STRING`
: The format used for printing dates of files. By default, is dd-mm-yyyy.

Per drive flags and variables

General information

Per drive flags and values may be described in a drive section. A drive section starts with `drive` "*driveletter*" :

Then follow variable-value pairs and flags.

This is a sample drive description:

```
drive a:
  file="/dev/fd0" use_xdf=1
```

Location information

For each drive, you need to describe where its data is physically stored (image file, physical device, partition, offset).

`file`
: The name of the file or device holding the disk image. This is mandatory. The file name should be enclosed in quotes.

`partition`
: Tells mtools to treat the drive as a partitioned device, and to use the given partition. Only primary partitions are accessible using this method, and they are numbered from 1 to 4. For logical partitions, use the more general `offset` variable. The `partition` variable is intended for removable media such as Syquest disks, ZIP drives, and magneto-optical disks. Although traditional DOS sees Syquest disks and magneto-optical disks as `‘giant floppy disks’` which are unpartitioned, OS/2 and Windows NT treat them like hard disks, i.e. partitioned devices. The `partition` flag is also useful DOSEMU hdimages. It is not recommended for hard disks for which direct access to partitions is available through mounting.

`offset`
: Describes where in the file the MS-DOS file system starts. This is useful for logical partitions in DOSEMU hdimages, and for ATARI ram disks. By default, this is zero, meaning that the file system starts right at the beginning of the device or file.

Disk Geometry Configuration

Geometry information describes the physical characteristics about the disk. Its has three purposes:

formatting
: The geometry information is written into the boot sector of the newly made disk. However, you may also describe the geometry information on the command line. See section mformat, for details.

filtering
: On some Unixes there are device nodes which only support one physical geometry. For instance, you might need a different node to access a disk as high density or as low density. The geometry is compared to the actual geometry stored on the boot sector to make sure that this device node is able to correctly read the disk. If the geometry doesn't match, this drive entry fails, and the next drive entry bearing the same drive letter is tried. See section multiple descriptions, for more details on supplying several descriptions for one drive letter.

  If no geometry information is supplied in the configuration file, all disks are accepted. On Linux (and on SPARC) there exist device nodes with configurable geometry (`'/dev/fd0'`, `'/dev/fd1'` etc), and thus filtering is not needed (and ignored) for disk drives. (Mtools still does do filtering on plain files (disk images) in Linux: this is mainly intended for test purposes, as I don't have access to a Unix which would actually need filtering).

  If you do not need filtering, but want still a default geometry for mformatting, you may switch off filtering using the `mformat_only` flag.

  If you want filtering, you should supply the `filter` flag. If you supply a geometry, you must supply one of both flags.

initial geometry
: On devices that support it (usually floppy devices), the geometry information is also used to set the initial geometry. This initial geometry is applied while reading the boot sector, which contains the real geometry. If no geometry information is supplied in the configuration file, or if the`mformat_only` flag is supplied, no initial configuration is done.

  On Linux, initial geometry is not really needed, as the configurable devices are able to auto-detect the disk type accurately enough (for most common formats) to read the boot sector.

Wrong geometry information may lead to very bizarre errors. That's why I strongly recommend that you add the `mformat_only` flag to your drive description, unless you really need filtering or initial geometry.

The following geometry related variables are available:

`cylinders`
`tracks`
: The number of cylinders. (`cylinders` is the preferred form, `tracks` is considered obsolete)

`heads`
: The number of heads (sides).

`sectors`
: The number of sectors per track.

Example: the following drive section describes a 1.44M drive:

```
drive a:
  file="/dev/fd0H1440"
  fat_bits=12
  cylinders=80 heads=2 sectors=18
```

mformat_only

The following shorthand geometry descriptions are available:

`1.44m`
: high density 3 1/2 disk. Equivalent to: `fat_bits=12 cylinders=80 heads=2 sectors=18`

`1.2m`
: high density 5 1/4 disk. Equivalent to: `fat_bits=12 cylinders=80 heads=2 sectors=15`

`720k`
: double density 3 1/2 disk. Equivalent to: `fat_bits=12 cylinders=80 heads=2 sectors=9`

`360k`
: double density 5 1/4 disk. Equivalent to: `fat_bits=12 cylinders=40 heads=2 sectors=9`

The shorthand format descriptions may be amended. For example, `360k sectors=8` describes a 320k disk and is equivalent to: `fat_bits=12 cylinders=40 heads=2 sectors=8`

Open Flags

Moreover, the following flags are available:

`sync`
: All i/o operations are done synchronously

`nodelay`
: The device or file is opened with the O_NDELAY flag. This is needed on some non-Linux architectures.

`exclusive`
: The device or file is opened with the O_EXCL flag. On Linux, this ensures exclusive access to the floppy drive. On most other architectures, and for plain files it has no effect at all.

General Purpose Drive Variables

The following general purpose drive variables are available. Depending to their type, these variables can be set to a string (precmd) or an integer (all others)

`fat_bits`
: The number of FAT bits. This may be 12 or 16. This is very rarely needed, as it can almost always be deduced from information in the boot sector. On the contrary, describing the number of fat bits may actually be harmful if you get it wrong. You should only use it if mtools gets the auto-detected number of fat bits wrong, or if you want to mformat a disk with a weird number of fat bits.

`codepage`
: Describes the DOS code page used for short filenames. This is a number between 1 and 999. By default, code page 850 is used. The reason for this is because this code page contains most of the characters that are also available in ISO-Latin-1. You may also specify a global code page for all drives by using the global `default_codepage` parameter (outside of any drive description). This parameters exists starting at version 4.0.0

`data_map`
: Remaps data from image file. This is useful for image files which might need additional zero-filled sectors to be inserted. Such is the case for instance for IBM 3174 floppy images. These images represent floppy disks with fewer sectors on their first cylinder. These missing sectors are not stored in the image, but are still counted in the filesystem layout. The data_map allows to fake these missing sectors for the upper layers of mtools. A data_map is a comma-separated sequence of source type and size. Source type may be `zero` for zero-filled sectors created by map, `skip` for data in raw image to be ignored (skipped), and nothing for data to be used as is (copied) from the raw image. Datamap is automatically complemented by an implicit last element of data to be used as is from current offset to end of file. Each size is a number followed by a unit: `s` for a 512 byte sector, `K` for Kbytes, `M` for megabytes, `G` for gigabytes, and nothing for single bytes.

Example:

`data_map=1s,zero31s,28s,skip1s` would be a map for use with IBM 3174 floppy images. First sector (`1s`, boot sector) is used as is. Then follow 31 fake zero-filled sectors (`zero31s`), then the next 28 sectors from image (`28s`) are used as is (they contain FAT and root directory), then one sector from image is skipped (`skip1s`), and finally the rest of image is used as is (implicit)

`precmd`

On some variants of Solaris, it is necessary to call 'volcheck -v' before opening a floppy device, in order for the system to notice that there is indeed a disk in the drive. `precmd="volcheck -v"` in the drive clause establishes the desired behavior.

`blocksize`

This parameter represents a default block size to be always used on this device. All I/O is done with multiples of this block size, independently of the sector size registered in the file system's boot sector. This is useful for character devices whose sector size is not 512, such as for example CD-ROM drives on Solaris.

Only the `file` variable is mandatory. The other parameters may be left out. In that case a default value or an auto-detected value is used.

General Purpose Drive Flags

A flag can either be set to 1 (enabled) or 0 (disabled). If the value is omitted, it is enabled. For example, `scsi` is equivalent to `scsi=1`

`nolock`

Instruct mtools to not use locking on this drive. This is needed on systems with buggy locking semantics. However, enabling this makes operation less safe in cases where several users may access the same drive at the same time.

`scsi` When set to 1, this option tells mtools to use raw SCSI I/O instead of the standard read/write calls to access the device. Currently, this is supported on HP-UX, Solaris and SunOS. This is needed because on some architectures, such as SunOS or Solaris, PC media can't be accessed using the `read` and `write` system calls, because the OS expects them to contain a Sun specific "disk label".

As raw SCSI access always uses the whole device, you need to specify the "partition" flag in addition

On some architectures, such as Solaris, mtools needs root privileges to be able to use the `scsi` option. Thus mtools should be installed setuid root on Solaris if you want to access Zip/Jaz drives. Thus, if the `scsi` flag is given, `privileged` is automatically implied, unless explicitly disabled by `privileged=0`

Mtools uses its root privileges to open the device, and to issue the actual SCSI I/O calls. Moreover, root privileges are only used for drives described in a system-wide configuration file such as `'/etc/mtools.conf'`, and not for those described in `'~/.mtoolsrc'` or `'$MTOOLSRC'`.

`privileged`

When set to 1, this instructs mtools to use its setuid and setgid privileges for opening the given drive. This option is only valid for drives described in the system-wide configuration files (such as `'/etc/mtools.conf'`, not `'~/.mtoolsrc'` or `'$MTOOLSRC'`). Obviously, this option is also a no op if mtools is not installed setuid or setgid. This option is implied by 'scsi=1', but again only for drives defined in system-wide configuration files. Privileged may also be set explicitly to 0, in order to tell mtools not to use its privileges for a given drive even if `scsi=1` is set.

Mtools only needs to be installed setuid if you use the `privileged` or `scsi` drive variables. If you do not use these options, mtools works perfectly well even when not installed setuid root.

`vold`

Instructs mtools to interpret the device name as a vold identifier rather than as a filename. The vold identifier is translated into a real filename using the `media_findname()` and `media_oldaliases()` functions of the `volmgt` library. This flag is only available if you configured mtools with the `--enable-new-vold` option before compilation.

`swap`

Consider the media as a word-swapped Atari disk.

`use_xdf`
If this is set to a non-zero value, mtools also tries to access this disk as an XDF disk. XDF is a high capacity format used by OS/2. This is off by default. See section XDF, for more details.

`mformat_only`
Tells mtools to use the geometry for this drive only for mformatting and not for filtering.

`filter`
Tells mtools to use the geometry for this drive both for mformatting and filtering.

`remote`
Tells mtools to connect to floppyd (see section floppyd).

Supplying multiple descriptions for a drive

It is possible to supply multiple descriptions for a drive. In that case, the descriptions are tried in order until one is found that fits. Descriptions may fail for several reasons:

1. because the geometry is not appropriate,
2. because there is no disk in the drive,
3. or because of other problems.

Multiple definitions are useful when using physical devices which are only able to support one single disk geometry. Example:

```
drive a: file="/dev/fd0H1440" 1.44m
drive a: file="/dev/fd0H720" 720k
```

This instructs mtools to use /dev/fd0H1440 for 1.44m (high density) disks and /dev/fd0H720 for 720k (double density) disks. On Linux, this feature is not really needed, as the /dev/fd0 device is able to handle any geometry.

You may also use multiple drive descriptions to access both of your physical drives through one drive letter:

```
drive z: file="/dev/fd0"
drive z: file="/dev/fd1"
```

With this description, `mdir z:` accesses your first physical drive if it contains a disk. If the first drive doesn't contain a disk, mtools checks the second drive.

When using multiple configuration files, drive descriptions in the files parsed last override descriptions for the same drive in earlier files. In order to avoid this, use the `drive+` or `+drive` keywords instead of `drive`. The first adds a description to the end of the list (i.e. it will be tried last), and the first adds it to the start of the list.

Location of configuration files and parsing order

The configuration files are parsed in the following order:

1. compiled-in defaults
2. `'/etc/mtools.conf'`
3. `'~/.mtoolsrc'`.

4. `‘$MTOOLSRC’` (file pointed by the `MTOOLSRC` environmental variable)

Options described in the later files override those described in the earlier files. Drives defined in earlier files persist if they are not overridden in the later files. For instance, drives A and B may be defined in `‘/etc/mtools.conf’` and drives C and D may be defined in `‘~/.mtoolsrc’` However, if `‘~/.mtoolsrc’` also defines drive A, this new description would override the description of drive A in `‘/etc/mtools.conf’` instead of adding to it. If you want to add a new description to a drive already described in an earlier file, you need to use either the `+drive` or `drive+` keyword.

Backwards compatibility with old configuration file syntax

The syntax described herein is new for version `mtools-3.0`. The old line-oriented syntax is still supported. Each line beginning with a single letter is considered to be a drive description using the old syntax. Old style and new style drive sections may be mixed within the same configuration file, in order to make upgrading easier. Support for the old syntax will be phased out eventually, and in order to discourage its use, I purposefully omit its description here.

See also

mtools