Mtools

This manual is for Mtools (version 4.0.49, June 2025), which is a collection of tools to allow Unix systems to manipulate MS-DOS files.

Copyright © 2007, 2009 Free Software Foundation, Inc. Copyright © 1996-2005,2007-2011,2013 Alain Knaff.

> Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, with no Front-Cover Texts, and with no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

Table of Contents

Introduction **1**

1 Where to get mtools **2**

2 Common features of all mtools commands **3**

2.1 Options and filenames 3
2.2 Drive letters 3
2.3 Sizes and offsets 4
2.4 Current working directory 4
2.5 VFAT-style long file names 4
2.6 Name clashes 5
2.7 Case sensitivity of the VFAT file system 6
2.8 high capacity formats 6
2.8.1 More sectors 7
2.8.2 Bigger sectors 7
2.8.3 2m 7
2.8.4 XDF 7
2.9 Exit codes 8
2.10 Bugs 8
2.11 Credits 8

3 How to configure mtools for your environment **9**

3.1 Description 9
3.2 Location of the configuration files 9
3.2.1 General configuration file syntax 9
3.3 Default values 9
3.4 Global variables 9
3.5 Per drive flags and variables 10
3.5.1 General information 10
3.5.2 Location information 11
3.5.3 Disk Geometry Configuration 11
3.5.4 Open Flags 13
3.5.5 General Purpose Drive Variables 13
3.5.6 General Purpose Drive Flags 14
3.5.7 Supplying multiple descriptions for a drive 15
3.6 Location of configuration files and parsing order 16
3.7 Backwards compatibility with old configuration file syntax 16

4 Command list **17**

4.1 Floppyd 17
4.1.1 Authentication 17

4.1.2 Command line options ... 17
4.1.3 Connecting to floppyd ... 18
4.1.4 Examples: ... 18
4.2 Floppyd_installtest ... 18
4.3 Mattrib ... 19
4.4 Mbadblocks ... 19
4.4.1 Command line options ... 20
4.4.2 Bugs ... 20
4.5 Mcat ... 20
4.6 Mcd ... 20
4.7 Mcopy ... 21
4.7.1 Bugs ... 22
4.8 Mdel ... 22
4.9 Mdeltree ... 22
4.10 Mdir ... 22
4.11 Mdoctorfat ... 23
4.12 Mdu ... 23
4.13 Mformat ... 23
4.13.1 Number of sectors per cluster ... 26
4.14 Mkmanifest ... 27
4.14.1 Example ... 27
4.14.2 Bugs ... 27
4.15 Minfo ... 28
4.16 Mlabel ... 28
4.17 Mmd ... 28
4.18 Mmount ... 28
4.19 Mmove ... 29
4.20 Mpartition ... 29
4.20.1 Choice of partition type ... 30
4.21 Mrd ... 31
4.22 Mren ... 31
4.23 Mshortname ... 31
4.24 Mshowfat ... 31
4.25 Mtoolstest ... 32
4.26 Mtype ... 32
4.27 Mzip ... 32
4.27.1 Bugs ... 33

5 Architecture specific compilation flags ... 34

6 Porting mtools to architectures which are not supported yet ... 35

Command Index ... 38

Variable index **39**

Concept index **40**

Introduction

Mtools is a collection of tools to allow Unix systems to manipulate MS-DOS files: read, write, and move around files on an MS-DOS file system (typically a floppy disk). Where reasonable, each program attempts to emulate the MS-DOS equivalent command. However, unnecessary restrictions and oddities of DOS are not emulated. For instance, it is possible to move subdirectories from one subdirectory to another.

Mtools is sufficient to give access to MS-DOS file systems. For instance, commands such as `mdir a:` work on the `a:` floppy without any preliminary mounting or initialization (assuming the default `/etc/mtools.conf` works on your machine). With mtools, one can change floppies too without unmounting and mounting.

This manual is for Mtools (version 4.0.49, June 2025), which is a collection of tools to allow Unix systems to manipulate MS-DOS files.

Copyright © 2007, 2009 Free Software Foundation, Inc. Copyright © 1996-2005,2007-2011,2013 Alain Knaff.

> Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, with no Front-Cover Texts, and with no Back-Cover Texts. A copy of the license is included in the section entitled “GNU Free Documentation License”.

1 Where to get mtools

Mtools can be found at the following places (and their mirrors):

`http://ftp.gnu.org/gnu/mtools/mtools-4.0.49.tar.gz`

These patches are named `mtools-`*version*`-`*ddmm*`.taz`, where version stands for the base version, *dd* for the day and *mm* for the month. Due to a lack of space, I usually leave only the most recent patch.

There is an mtools mailing list at info-mtools @ gnu.org . Please send all bug reports to this list. You may subscribe to the list at https://lists.gnu.org/mailman/listinfo/info-mtools. (N.B. Please remove the spaces around the "@". I left them there in order to fool spambots.) Announcements of new mtools versions will also be sent to the list, in addition to the Linux announce newsgroups. The mailing list is archived at http://lists.gnu.org/pipermail/info-mtools/

2 Common features of all mtools commands

2.1 Options and filenames

MS-DOS filenames are composed of a drive letter followed by a colon, a subdirectory, and a filename. Only the filename part is mandatory, the drive letter and the subdirectory are optional. Filenames without a drive letter refer to Unix files. Subdirectory names can use either the '/' or '\' separator. The use of the '\' separator or wildcards requires the names to be enclosed in quotes to protect them from the shell. However, wildcards in Unix filenames should not be enclosed in quotes, because here we **want** the shell to expand them.

The regular expression "pattern matching" routines follow the Unix-style rules. For example, '`*`' matches all MS-DOS files in lieu of '`*.*`'. The archive, hidden, read-only and system attribute bits are ignored during pattern matching.

All options use the `-` (minus) as their first character, not `/` as you'd expect in MS-DOS.

Most mtools commands allow multiple filename parameters, which doesn't follow MS-DOS conventions, but which is more user-friendly.

Most mtools commands allow options that instruct them how to handle file name clashes. See Section 2.6 [name clashes], page 5, for more details on these.

All commands accept the `-i` flag which allows to specify an image file (See Section 2.2 [drive letters], page 3).

All commands accept the `-V` flag which prints the version, and most accept the `-v` flag, which switches on verbose mode. In verbose mode, these commands print out the name of the MS-DOS files upon which they act, unless stated otherwise. See Chapter 4 [Commands], page 17, for a description of the options which are specific to each command.

2.2 Drive letters

The meaning of the drive letters depends on the target architectures. However, on most target architectures, drive A is the first floppy drive, drive B is the second floppy drive (if available), drive J is a Jaz drive (if available), and drive Z is a Zip drive (if available). On those systems where the device name is derived from the SCSI id, the Jaz drive is assumed to be at SCSI target 4, and the Zip at SCSI target 5 (factory default settings). On Linux, both drives are assumed to be the second drive on the SCSI bus (/dev/sdb). The default settings can be changes using a configuration file (see Chapter 3 [Configuration], page 9).

The drive letter : (colon) has a special meaning. It is used to access image files which are directly specified on the command line using the `-i` options.

Example:

```
mcopy -i my-image-file.bin ::file1 ::file2 .
```

This copies `file1` and `file2` from the image file (`my-image-file.bin`) to the `/tmp` directory.

You can also supply an offset within the image file by including `@@`*offset* into the file name.

Example:

```
mcopy -i my-image-file.bin@@1M ::file1 ::file2 .
```

This looks for the image at the offset of 1M in the file, rather than at its beginning.

2.3 Sizes and offsets

Certain commands, such as `mformat`, allow you to specify a hard disk size.

The size is a number, followed by an optional unit suffix.

The number is interpreted as a hexadecimal number if it starts with `0x`, as an octal number if it starts with the digit `0`, or as a decimal number if it starts with any other digit.

The following units are supported:

- `S` sectors (of 512 bytes)
- `K` Kilobytes
- `M` Megabytes
- `G` Gigabytes
- `T` Terabytes

If no unit is specified, the default depends on context. Mformat's `T` option, and mpartition's `l` option assume sectors, whereas all other places (mshowfat's `o` option, `offset` drive config option and `data_map` drive config option) assume bytes.

2.4 Current working directory

The `mcd` command (Section 4.6 [mcd], page 20) is used to establish the device and the current working directory (relative to the MS-DOS file system), otherwise the default is assumed to be `A:/`. However, unlike MS-DOS, there is only one working directory for all drives, and not one per drive. Moreover, if you set drive `::/`, mtools does not remember the associated image file, but only the fact that you are working on an image file at all.

2.5 VFAT-style long file names

This version of mtools supports VFAT style long filenames. If a Unix filename is too long to fit in a short DOS name, it is stored as a VFAT long name, and a companion short name is generated. This short name is what you see when you examine the disk with a pre-7.0 version of DOS. The following table shows some examples of short names:

Long name	MS-DOS name	Reason for the change
thisisatest	THISIS~1	filename too long
alain.knaff	ALAIN~1.KNA	extension too long
prn.txt	PRN~1.TXT	PRN is a device name
.abc	ABC~1	null filename
hot+cold	HOT_CO~1	illegal character

As you see, the following transformations happen to derive a short name:

- Illegal characters are replaced by underscores. The illegal characters are `;+=[]',\"*\\<>/?:|`.
- Extra dots, which cannot be interpreted as a main name/extension separator are removed
- A `~`*n* number is generated,
- The name is shortened so as to fit in the 8+3 limitation

The initial Unix-style file name (whether long or short) is also called the *primary* name, and the derived short name is also called the *secondary* name.

Example:

```
mcopy /etc/motd a:Reallylongname
```

Mtools creates a VFAT entry for Reallylongname, and uses REALLYLO as a short name. Reallylongname is the primary name, and REALLYLO is the secondary name.

```
mcopy /etc/motd a:motd
```

Motd fits into the DOS filename limits. Mtools doesn't need to derivate another name. Motd is the primary name, and there is no secondary name.

In a nutshell: The primary name is the long name, if one exists, or the short name if there is no long name.

Although VFAT is much more flexible than FAT, there are still names that are not acceptable, even in VFAT. There are still some illegal characters left (`\"*\\<>/?:|`), and device names are still reserved.

```
Unix name       Long name       Reason for the change
---------       ----------      ---------------------
prn             prn-1           PRN is a device name
ab:c            ab_c-1          illegal character
```

As you see, the following transformations happen if a long name is illegal:

- Illegal characters are replaces by underscores,
- A *-n* number is generated,

2.6 Name clashes

When writing a file to disk, its long name or short name may collide with an already existing file or directory. This may happen for all commands which create new directory entries, such as `mcopy`, `mmd`, `mren`, `mmove`. When a name clash happens, mtools asks you what it should do. It offers several choices:

`overwrite`
: Overwrites the existing file. It is not possible to overwrite a directory with a file.

`rename`
: Renames the newly created file. Mtools prompts for the new filename

`autorename`
: Renames the newly created file. Mtools chooses a name by itself, without prompting

`skip`
: Gives up on this file, and moves on to the next (if any)

To chose one of these actions, type its first letter at the prompt. If you use a lower case letter, the action only applies for this file only, if you use an upper case letter, the action applies to all files, and you won't be prompted again.

You may also chose actions (for all files) on the command line, when invoking mtools:

`-D o`
: Overwrites primary names by default.

`-D O`
: Overwrites secondary names by default.

`-D r` Renames primary name by default.

`-D R` Renames secondary name by default.

`-D a` Autorenames primary name by default.

`-D A` Autorenames secondary name by default.

`-D s` Skip primary name by default.

`-D S` Skip secondary name by default.

`-D m` Ask user what to do with primary name.

`-D M` Ask user what to do with secondary name.

Note that for command line switches lower/upper differentiates between primary/secondary name whereas for interactive choices, lower/upper differentiates between just-this-time/always.

The primary name is the name as displayed in Windows 95 or Windows NT: i.e. the long name if it exists, and the short name otherwise. The secondary name is the "hidden" name, i.e. the short name if a long name exists.

By default, the user is prompted if the primary name clashes, and the secondary name is autorenamed.

If a name clash occurs in a Unix directory, mtools only asks whether to overwrite the file, or to skip it.

2.7 Case sensitivity of the VFAT file system

The VFAT file system is able to remember the case of the filenames. However, filenames which differ only in case are not allowed to coexist in the same directory. For example if you store a file called LongFileName on a VFAT file system, mdir shows this file as LongFileName, and not as Longfilename. However, if you then try to add LongFilename to the same directory, it is refused, because case is ignored for clash checks.

The VFAT file system allows you to store the case of a filename in the attribute byte, if all letters of the filename are the same case, and if all letters of the extension are the same case too. Mtools uses this information when displaying the files, and also to generate the Unix filename when mcopying to a Unix directory. This may have unexpected results when applied to files written using an pre-7.0 version of DOS: Indeed, the old style filenames map to all upper case. This is different from the behavior of the old version of mtools which used to generate lower case Unix filenames.

2.8 high capacity formats

Mtools supports a number of formats which allow storage of more data on disk than usual. Due to different operating system abilities, these formats are not supported on all operating systems. Mtools recognizes these formats transparently where supported.

In order to format these disks, you need to use an operating system specific tool. For Linux, suitable floppy tools can be found in the `fdutils` package at the following locations~:

`http://www.fdutils.linux.lu/.`

See the manual pages included in that package for further detail: Use `superformat` to format all formats except XDF, and use `xdfcopy` to format XDF.

2.8.1 More sectors

The oldest method of fitting more data on a disk is to use more sectors and more cylinders. Although the standard format uses 80 cylinders and 18 sectors (on a 3 1/2 high density disk), it is possible to use up to 83 cylinders (on most drives) and up to 21 sectors. This method allows to store up to 1743K on a 3 1/2 HD disk. However, 21 sector disks are twice as slow as the standard 18 sector disks because the sectors are packed so close together that we need to interleave them. This problem doesn't exist for 20 sector formats.

These formats are supported by numerous DOS shareware utilities such as `fdformat` and `vgacopy`. In his infinite hubris, Bill Gate$ believed that he invented this, and called it '`DMF disks`', or '`Windows formatted disks`'. But in reality, it has already existed years before! Mtools supports these formats on Linux, on SunOS and on the DELL Unix PC.

2.8.2 Bigger sectors

By using bigger sectors it is possible to go beyond the capacity which can be obtained by the standard 512-byte sectors. This is because of the sector header. The sector header has the same size, regardless of how many data bytes are in the sector. Thus, we save some space by using *fewer*, but bigger sectors. For example, 1 sector of 4K only takes up header space once, whereas 8 sectors of 512 bytes have also 8 headers, for the same amount of useful data.

This method permits storage of up to 1992K on a 3 1/2 HD disk.

Mtools supports these formats only on Linux.

2.8.3 2m

The 2m format was originally invented by Ciriaco Garcia de Celis. It also uses bigger sectors than usual in order to fit more data on the disk. However, it uses the standard format (18 sectors of 512 bytes each) on the first cylinder, in order to make these disks easier to handle by DOS. Indeed this method allows you to have a standard sized boot sector, which contains a description of how the rest of the disk should be read.

However, the drawback of this is that the first cylinder can hold less data than the others. Unfortunately, DOS can only handle disks where each track contains the same amount of data. Thus 2m hides the fact that the first track contains less data by using a *shadow FAT*. (Usually, DOS stores the FAT in two identical copies, for additional safety. XDF stores only one copy, but tells DOS that it stores two. Thus the space that would be taken up by the second FAT copy is saved.) This also means that you should **never use a 2m disk to store anything else than a DOS file system**.

Mtools supports these formats only on Linux.

2.8.4 XDF

XDF is a high capacity format used by OS/2. It can hold 1840 K per disk. That's lower than the best 2m formats, but its main advantage is that it is fast: 600 milliseconds per track. That's faster than the 21 sector format, and almost as fast as the standard 18 sector format. In order to access these disks, make sure mtools has been compiled with XDF support, and set the `use_xdf` variable for the drive in the configuration file. See Chapter 5 [Compiling mtools], page 34, and Section 3.5.5 [miscellaneous variables], page 13, for details

on how to do this. Fast XDF access is only available for Linux kernels which are more recent than 1.1.34.

Mtools supports this format only on Linux.

Caution / Attention distributors: If mtools is compiled on a Linux kernel more recent than 1.3.34, it won't run on an older kernel. However, if it has been compiled on an older kernel, it still runs on a newer kernel, except that XDF access is slower. It is recommended that distribution authors only include mtools binaries compiled on kernels older than 1.3.34 until 2.0 comes out. When 2.0 will be out, mtools binaries compiled on newer kernels may (and should) be distributed. Mtools binaries compiled on kernels older than 1.3.34 won't run on any 2.1 kernel or later.

2.9 Exit codes

All the Mtools commands return 0 on success, 1 on utter failure, or 2 on partial failure. All the Mtools commands perform a few sanity checks before going ahead, to make sure that the disk is indeed an MS-DOS disk (as opposed to, say an ext2 or MINIX disk). These checks may reject partially corrupted disks, which might otherwise still be readable. To avoid these checks, set the MTOOLS_SKIP_CHECK environmental variable or the corresponding configuration file variable (see Section 3.4 [global variables], page 9)

2.10 Bugs

An unfortunate side effect of not guessing the proper device (when multiple disk capacities are supported) is an occasional error message from the device driver. These can be safely ignored.

The fat checking code chokes on 1.72 Mb disks mformatted with pre-2.0.7 mtools. Set the environmental variable MTOOLS_FAT_COMPATIBILITY (or the corresponding configuration file variable, Section 3.4 [global variables], page 9) to bypass the fat checking.

2.11 Credits

Original code (versions through 2.0.7?) of mtools was developped by Emmet P. Gray (Texas, USA).

David C. Niemi contributed VFAT support and helped maintain mtools.

The packaging process of mtools uses a modified version of Beverly A. Erlebacher's texi2roff tool to convert the texi documentation into man pages.

3 How to configure mtools for your environment

3.1 Description

This sections explains the syntax of the configurations files for mtools. The configuration files 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

3.2 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.

3.2.1 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).

3.3 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.

3.4 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.

3.5 Per drive flags and variables

3.5.1 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
```

3.5.2 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.

3.5.3 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 4.13 [mformat], page 23, 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 3.5.7 [multiple descriptions], page 15, 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`

3.5.4 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.

3.5.5 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, postcmd) 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` Executes the given command before opening the device. 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.

`postcmd` Executes the given command after closing the device. May be useful if mtools shares the image file with some other application, in order to release the image file to that application.

`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.

3.5.6 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 2.8.4 [XDF], page 7, 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 4.1 [floppyd], page 17).

3.5.7 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.

3.6 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.

3.7 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.

4 Command list

This section describes the available mtools commands, and the command line parameters that each of them accepts. Options which are common to all mtools commands are not described here, Section 2.1 [arguments], page 3, for a description of those.

4.1 Floppyd

`Floppyd` is used as a server to grant access to the floppy drive to clients running on a remote machine, just as an X server grants access to the display to remote clients. It has the following syntax:

`floppyd` [`-d`] [`-D`] [`-l`] [`-s` *port*] [`-r` *user*] [`-b` *ipaddr*] [`-x` *display*] *devicenames*

`floppyd` is always associated with an X server. It runs on the same machine as its X server, and listens on port 5703 and above.

4.1.1 Authentication

`floppyd` authenticates remote clients using the `Xauthority` protocol. Xhost authentication is not supported. Each floppyd is associated with an X server. When a remote client attempts to connect to floppyd, it sends floppyd the X authority record corresponding to floppyd's X server. Floppyd in turn then tries to open up a connection to the X server in order to verify the authenticity of the xauth record. If the connection to the X server succeeds, the client is granted access. `DISPLAY`.

Caution: In order to make authentication work correctly, the local host should **not** be listed in the `xhost` list of allowed hosts. Indeed, hosts listed in `xhost` do not need a correct `Xauthority` cookie to connect to the X server. As `floppyd` runs on the same host as the X server, all its probe connection would succeed even for clients who supplied a bad cookie. This means that your floppy drive would be open to the world, i.e. a huge security hole. If your X server does not allow you to remove `localhost:0` and `:0` from the `xhost` list, you can prevent floppyd from probing those display names with the `-l` option.

4.1.2 Command line options

`d` Server mode. Floppyd runs its own server loop. Do not supply this if you start floppyd from `inetd.conf`

`D` Do not daemonize. Floppyd will not put itself into the background even if started in server mode.

`s` *port* Port number for daemon mode. Default is 5703 + *displaynumber*. This flag implies daemon mode. For example, for display `hitchhiker:5`, the port would be 5708.

`b` *ipaddr* Bind address (for multi homed hosts). This flag implies daemon mode

`r` *user* Run the server under as the given user

`x` *display* X display to use for authentication. By default, this is taken from the `DISPLAY` variable. If neither the `x` attribute is present nor `DISPLAY` is set, floppyd uses `:0.0`.

devicenames is a list of device nodes to be opened. Default is `/dev/fd0`. Multiple devices are only supported on mtools versions newer than 3.9.11.

4.1.3 Connecting to floppyd

In order to use floppyd, add the flag `remote` to the device description in your `~/.mtoolsrc` file. If the flag `remote` is given, the `file` parameter of the device description is taken to be a remote address. It's format is the following: *hostname*`:`*displaynumber*[`/`[*baseport*][`/`*drive*]]. When using this entry, mtools connects to port *baseport*+*displaynumber* at *hostname*. By default *baseport* is 5703. The drive parameter is to distinguish among multiple drives associated with a single display (only mtools versions more recent than 3.9.11)

4.1.4 Examples:

The following starts a floppy daemon giving access to `/dev/fd0`, listening on the default port 5703, tied to the default X servers:

```
floppyd -d /dev/fd0
```

Each of the following starts a floppy daemon giving access to `/dev/fd1`, tied to the :1 local X servers, and listening on port 5704. We assume that the local host is named `hitchhiker`.

```
floppyd -d /dev/fd0
floppyd -d -x :1 -p 5704 /dev/fd0
```

If you want to start floppyd by `inetd` instead of running it as a daemon, insert the following lines into `/etc/services`:

```
# floppy daemon
floppyd-0    5703/tcp    # floppy daemon for X server :0
floppyd-1    5704/tcp    # floppy daemon for X server :1
```

And insert the following into `/etc/inetd.conf` (assuming that you have defined a user named floppy in your `/etc/passwd`):

```
# floppy daemon
floppyd-0 stream  tcp  wait  floppy  /usr/sbin/floppyd floppyd /dev/fd0
floppyd-1 stream  tcp  wait  floppy  /usr/sbin/floppyd floppyd -x :1 /dev/fd0
```

Note that you need to supply the X display names for the second floppyd. This is because the port is opened by inetd.conf, and hence floppyd cannot know its number to interfere the display number.

On the client side, insert the following into your `~/.mtoolsrc` to define a drive letter accessing floppy drive in your X terminal:

```
drive x: file="$DISPLAY" remote
```

If your X terminal has more than one drive, you may access the additional drives as follows:

```
drive y: file="$DISPLAY//1" remote
drive z: file="$DISPLAY//2" remote
```

4.2 Floppyd_installtest

`Floppyd_installtest` is used to check for the presence of a running floppyd daemon. This is useful, if you have a small front-end script to mtools, which decides whether to use floppyd or not.

`floppyd_installtest` [`-f`] Connect-String

If the `-f` option is specified, `floppyd_installtest` does a full X-Cookie authentication and complains if this does not work.

The connect-String has the format described in the floppyd-section: *hostname*`:`*displaynumber*[`/`*baseport*]

4.3 Mattrib

`Mattrib` is used to change MS-DOS file attribute bits. It has the following syntax:

`mattrib` [`-a|+a`] [`-h|+h`] [`-r|+r`] [`-s|+s`] [`-/`] [`-p`] [`-X`] *msdosfile* [*msdosfiles* ...]

`Mattrib` adds attribute bits to an MS-DOS file (with the '`+`' operator) or remove attribute bits (with the '`-`' operator).

`Mattrib` supports the following attribute bits:

`a` Archive bit. Used by some backup programs to indicate a new file.

`r` Read-only bit. Used to indicate a read-only file. Files with this bit set cannot be erased by `DEL` nor modified.

`s` System bit. Used by MS-DOS to indicate a operating system file.

`h` Hidden bit. Used to make files hidden from `DIR`.

`Mattrib` supports the following command line flags:

`/` Recursive. Recursively list the attributes of the files in the subdirectories.

`X` Concise. Prints the attributes without any whitespace padding. If neither the `"/"` option is given, nor the *msdosfile* contains a wildcard, and there is only one MS-DOS file parameter on the command line, only the attribute is printed, and not the filename. This option is convenient for scripts

`p` Replay mode. Outputs a series of `mformat` commands that will reproduce the current situation, starting from a situation as left by untarring the MS-DOS file system. Commands are only output for attribute settings that differ from the default (archive bit set for files, unset for directories). This option is intended to be used in addition to tar. The `readonly` attribute is not taken into account, as tar can set that one itself.

4.4 Mbadblocks

The `mbadblocks` command is used to mark some clusters on an MS-DOS filesystem bad. It has the following syntax:

`mbadblocks` [`-s` *sectorlist*`|-c` *clusterlist*`|-w`] *drive*`:`

If no command line flags are supplied, `Mbadblocks` scans an MS-DOS filesystem for bad blocks by simply trying to read them and flag them if read fails. All blocks that are unused are scanned, and if detected bad are marked as such in the FAT.

This command is intended to be used right after `mformat`. It is not intended to salvage data from bad disks.

4.4.1 Command line options

`c` *file* Use a list of bad clusters, rather than scanning for bad clusters itself.

`s` *file* Use a list of bad sectors (counted from beginning of filesystem), rather than trying for bad clusters itself.

`w` Write a random pattern to each cluster, then read it back and flag cluster as bad if mismatch. Only free clusters are tested in such a way, so any file data is preserved.

4.4.2 Bugs

`Mbadblocks` should (but doesn't yet :-( ) also try to salvage bad blocks which are in use by reading them repeatedly, and then mark them bad.

4.5 Mcat

The `mcat` command is used to copy an entire disk image from or to the floppy device. It uses the following syntax:

`mcat` [`-w`] *drive*`:`

`Mcat` performs the same task as the Unix `cat` command. It is included into the mtools package, since `cat` cannot access remote floppy devices offered by the mtools floppy daemon. Now it is possible to create boot floppies remotely.

The default operation is reading. The output is written to stdout.

If the `-w` option is specified, mcat reads a disk-image from stdin and writes it to the given device. **Use this carefully!** Because of the low-level nature of this command, it will happily destroy any data written before on the disk without warning!

4.6 Mcd

The `mcd` command is used to change the mtools working directory on the MS-DOS disk. It uses the following syntax:

`mcd [`*msdosdirectory*`]`

Without arguments, `mcd` reports the current device and working directory. Otherwise, `mcd` changes the current device and current working directory relative to an MS-DOS file system.

The environmental variable `MCWD` may be used to locate the file where the device and current working directory information is stored. The default is `$HOME/.mcwd`. Information in this file is ignored if the file is more than 6 hours old.

`Mcd` returns 0 on success or 1 on failure.

Unlike MS-DOS versions of `CD`, `mcd` can be used to change to another device. It may be wise to remove old `.mcwd` files at logout.

4.7 Mcopy

The `mcopy` command is used to copy MS-DOS files to and from Unix. It uses the following syntax:

```
mcopy [-bspanvmQT] [-D clash_option] sourcefile targetfile
mcopy [-bspanvmQT] [-D clash_option] sourcefile [ sourcefiles... ] targetdirectory
mcopy [-tnvm] MSDOSsourcefile
```

`Mcopy` copies the specified file to the named file, or copies multiple files to the named directory. The source and target can be either MS-DOS or Unix files.

The use of a drive letter designation on the MS-DOS files, 'a:' for example, determines the direction of the transfer. A missing drive designation implies a Unix file whose path starts in the current directory. If a source drive letter is specified with no attached file name (e.g. `mcopy a: .`), all files are copied from that drive.

If only a single, MS-DOS source parameter is provided (e.g. "mcopy a:foo.exe"), an implied destination of the current directory ('`.`') is assumed.

A filename of '`-`' means standard input or standard output, depending on its position on the command line.

`Mcopy` accepts the following command line options:

`t` Text file transfer. Mcopy translates incoming carriage return/line feeds to line feeds when copying from MS-DOS to Unix, and vice-versa when copying from Unix to MS-DOS. Moreover copying stops at Control-Z characters when copying from MS-DOS to Unix, and Control-Z is added at end when copying from Unix to MS-DOS.

`b` Batch mode. Optimized for huge recursive copies, but less secure if a crash happens during the copy.

`s` Recursive copy. Also copies directories and their contents

`p` Preserves the attributes of the copied files

`Q` When mcopying multiple files, quits as soon as one copy fails (for example due to lacking storage space on the target disk)

`a` Text (ASCII) file transfer. Same as t above.

`T` Text (ASCII) file transfer with character set conversion. Differs from `-a` in the `ASCII` also translates incoming PC-8 characters to ISO-8859-1 equivalents as far as possible. When reading DOS files, untranslatable characters are replaced by '`#`'; when writing DOS files, untranslatable characters are replaced by '`.`'.

`n` No confirmation when overwriting Unix files. `ASCII` doesn't warn the user when overwriting an existing Unix file. If the target file already exists, and the `-n` option is not in effect, `mcopy` asks whether to overwrite the file or to rename the new file (see Section 2.6 [name clashes], page 5, for details). In order to switch off confirmation for DOS files, use `-o`.

`m` Preserve the file modification time.

`v` Verbose. Displays the name of each file as it is copied.

4.7.1 Bugs

Unlike MS-DOS, the '+' operator (append) from MS-DOS is not supported. However, you may use `mtype` to produce the same effect:

```
mtype a:file1 a:file2 a:file3 >unixfile
mtype a:file1 a:file2 a:file3 | mcopy - a:msdosfile
```

4.8 Mdel

The `mdel` command is used to delete an MS-DOS file. Its syntax is:

`mdel` [`-v`] *msdosfile* [*msdosfiles* . . .]

`Mdel` deletes files on an MS-DOS file system.

`Mdel` asks for verification prior to removing a read-only file.

4.9 Mdeltree

The `mdeltree` command is used to delete an MS-DOS file. Its syntax is:

`mdeltree` [`-v`] *msdosdirectory* [*msdosdirectories*. . .]

`Mdeltree` removes a directory and all the files and subdirectories it contains from an MS-DOS file system. An error occurs if the directory to be removed does not exist.

4.10 Mdir

The `mdir` command is used to display an MS-DOS directory. Its syntax is:

`mdir` [`-/`] [`-f`] [`-w`] [`-a`] [`-b`] *msdosfile* [*msdosfiles*. . .]

`Mdir` displays the contents of MS-DOS directories, or the entries for some MS-DOS files.

`Mdir` supports the following command line options:

`/` Recursive output, just like MS-DOS' `-s` option

`w` Wide output. With this option, `mdir` prints the filenames across the page without displaying the file size or creation date.

`a` Also list hidden files.

`f` Fast. Do not try to find out free space. On larger disks, finding out the amount of free space takes up some non trivial amount of time, as the whole FAT must be read in and scanned. The `-f` flag bypasses this step. This flag is not needed on FAT32 file systems, which store the size explicitly.

`b` Concise listing. Lists each directory name or filename, one per line (including the filename extension). This switch displays no heading information and no summary. Only a newline separated list of pathnames is displayed.

An error occurs if a component of the path is not a directory.

4.11 Mdoctorfat

The `mdoctorfat` command was initially intended as a test/debugging tool. It allows to flag FAT clusters as bad blocks, and also to change the clusters allocated to a file. No consistency check is done, clusters previously allocated to the file are not freed. Use at your own risk.

Syntax:

`mdoctorfat` [`-o` *offset*] `-b` *drive clusters*
`mdoctorfat` [`-o` *offset*] [`-s` *size*] *file clusters*

If flag `-b` is present, the command marks the clusters as bad blocks rather than change the allocation of a file.

If flag `-b` is not present, the commands sets the clusters to allocated to the file.

If option `-s` is given in addition to the `-b`, this also changes the size in bytes of the file.

The cluster list is represented as a sequence of clusters or cluster ranges. Example: `7-10 20 21-24`. Each range needs to be a different parameter.

If `-o` is given, this specifies an offset to be added to each cluster (offset is expressed in number of clusters).

4.12 Mdu

`Mdu` is used to list the space occupied by a directory, its subdirectories and its files. It is similar to the `du` command on Unix. The unit used are clusters. Use the minfo command to find out the cluster size.

`mdu` [`-a`] [*msdosfiles* . . .]

`a` All files. List also the space occupied for individual files.

`s` Only list the total space, don't give details for each subdirectory.

4.13 Mformat

The `mformat` command is used to add an MS-DOS file system to a low-level formatted diskette. Its syntax is:

`mformat` [`-t` *cylinders*|`-T` *tot_sectors*] [`-h` *heads*] [`-s` *sectors*]
[`-f` *size*] [`-1`] [`-4`] [`-8`]
[`-v` *volume_label*]
[`-F`] [`-S` *sizecode*]
[`-M` *software_sector_size*]
[`-N` *serial_number*] [`-a`]
[`-C`] [`-H` *hidden_sectors*] [`-I` *fsVersion*]
[`-r` *root_sectors*] [`-L` *fat_len*]
[`-B` *boot_sector*] [`-k`]
[`-m` *media_descriptor*]
[`-K` *backup_boot*]
[`-R` *nb_reserved_sectors*]
[`-c` *clusters_per_sector*]
[`-d` *fat_copies*]

[-X] [-2 *sectors_on_track_0*] [-3]
[-0 *rate_on_track_0*] [-A *rate_on_other_tracks*]
drive:

`Mformat` adds a minimal MS-DOS file system (boot sector, FAT, and root directory) to a diskette that has already been formatted by a Unix low-level format.

The following options are supported: (The S, 2, 1 and M options may not exist if this copy of mtools has been compiled without the USE_2M option)

The following options are the same as for MS-DOS's format command:

`v` Specifies the volume label. A volume label identifies the disk and can be a maximum of 11 characters. If you omit the -v switch, mformat will assign no label to the disk.

`f` Specifies the size of the DOS file system to format. Only a certain number of predefined sizes are supported by this flag; for others use the -h/-t/-s flags. The following sizes are supported:

- 160 — 160K, single-sided, 8 sectors per track, 40 cylinders (for 5 1/4 DD)
- 180 — 160K, single-sided, 9 sectors per track, 40 cylinders (for 5 1/4 DD)
- 320 — 320K, double-sided, 8 sectors per track, 40 cylinders (for 5 1/4 DD)
- 360 — 360K, double-sided, 9 sectors per track, 40 cylinders (for 5 1/4 DD)
- 720 — 720K, double-sided, 9 sectors per track, 80 cylinders (for 3 1/2 DD)
- 1200 — 1200K, double-sided, 15 sectors per track, 80 cylinders (for 5 1/4 HD)
- 1440 — 1440K, double-sided, 18 sectors per track, 80 cylinders (for 3 1/2 HD)
- 2880 — 2880K, double-sided, 36 sectors per track, 80 cylinders (for 3 1/2 ED)

`t` Specifies the number of tracks on the disk.

`T` Specifies the number of total sectors on the disk. Only one of these 2 options may be specified (tracks or total sectors)

`h` The number of heads (sides).

`s` Specifies the number of sectors per track. If the 2m option is given, number of 512-byte sector equivalents on generic tracks (i.e. not head 0 track 0). If the 2m option is not given, number of physical sectors per track (which may be bigger than 512 bytes).

`1` Formats a single side (equivalent to -h 1)

`4` Formats a 360K double-sided disk (equivalent to -f 360). When used together with -the 1 switch, this switch formats a 180K disk

`8` Formats a disk with 8 sectors per track.

MS-DOS format's `q`, `u` and `b` options are not supported, and `s` has a different meaning. The following options are specific to mtools:

`F` Format the partition as FAT32.

`S` The size code. The size of the sector is 2 ^ (sizecode + 7).

`X` formats the disk as an XDF disk. See Section 2.8.4 [XDF], page 7, for more details. The disk has first to be low-level formatted using the xdfcopy utility included in the fdutils package. XDF disks are used for instance for OS/2 install disks.

`2` 2m format. The parameter to this option describes the number of sectors on track 0, head 0. This option is recommended for sectors bigger than normal.

`3` don't use a 2m format, even if the current geometry of the disk is a 2m geometry.

`0` Data transfer rate on track 0

`A` Data transfer rate on tracks other than 0

`M` software sector size. This parameter describes the sector size in bytes used by the MS-DOS file system. By default it is the physical sector size.

`N` Uses the requested serial number, instead of generating one automatically

`a` If this option is given, an Atari style serial number is generated. Ataris store their serial number in the OEM label.

`C` creates the disk image file to install the MS-DOS file system on it. Obviously, this is useless on physical devices such as floppies and hard disk partitions, but is interesting for image files.

`H` number of hidden sectors. This parameter is useful for formatting hard disk partition, which are not aligned on track boundaries (i.e. first head of first track doesn't belong to the partition, but contains a partition table). In that case the number of hidden sectors is in general the number of sectors per cylinder. This is untested.

`I` Sets the fsVersion id when formatting a FAT32 drive. In order to find this out, run minfo on an existing FAT32 drive, and mail me about it, so I can include the correct value in future versions of mtools.

`c` Sets the size of a cluster (in sectors). If this cluster size would generate a FAT that too big for its number of bits, mtools automatically increases the cluster size, until the FAT is small enough. If no cluster size is specified explicitly, mtools uses a default value as described in section "Number of sectors per cluster" below.

`d` Sets the number of FAT copies. Default is 2. This setting can also be specified using the `MTOOLS_NFATS` environment variable.

`r` Sets the size of the root directory (in sectors). Only applicable to 12 and 16 bit FATs. This setting can also be specified using the `MTOOLS_DIR_LEN` environment variable.

`L` Sets the length of the FAT.

`B` Use the boot sector stored in the given file or device, instead of using its own. Only the geometry fields are updated to match the target disks parameters.

`k` Keep the existing boot sector as much as possible. Only the geometry fields and other similar file system data are updated to match the target disks parameters.

`K` Sets the sector number where the backup of the boot sector should be stored (only relevant on FAT32).

`R` Sets the number of reserved sectors for this filesystem. This must be at least 1 for non-FAT32 disks, and at least 3 for FAT disks (in order to accommodate the boot sector, the info sector and the backup boot sector).

`m` Use a non-standard media descriptor byte for this disk. The media descriptor is stored at position 21 of the boot sector, and as first byte in each FAT copy. Using this option may confuse DOS or older mtools version, and may make the disk unreadable. Only use if you know what you are doing.

`b` Use a non-standard bios disk number for this disk. By default, bios disk number is inferred from media descriptor: 0x80 for media descriptor 0xf8, or 0x00 otherwise.

To format a diskette at a density other than the default, you must supply (at least) those command line parameters that are different from the default.

`Mformat` returns 0 on success or 1 on failure.

It doesn't record bad block information to the Fat, use `mbadblocks` for that.

4.13.1 Number of sectors per cluster

If the user indicates no cluster size, mformat figures out a default value for it.

For FAT32 it uses the following table to determine the number of sectors per cluster, depending on the total number of sectors on the filesystem.

more than 32*1024*1024*2: 64 sectors
between 16*1024*1024*2 and 32*1024*1024*2: 32 sectors
between 8*1024*1024*2 and 16*1024*1024*2: 16 sectors
between 260*1024*2 and 81024*1024*2: 1 sectors

This is derived from information on page 20 of Microsoft's `fatgen103` document, which currently can be found at the following address:

`https://staff.washington.edu/dittrich/misc/fatgen103.pdf`

For FAT12 and FAT16, mformat uses an iterative approach, where it starts with a set value, which it doubles until it is able to fill up the disk using that cluster size and a number of cluster less than the maximum allowed.

The starting value is 1 for disks with one head or less than 2000 sectors, and 2 for disks with more than one head, and more than 2000 sectors.

The number of sectors per cluster cannot go beyond 128.

4.14 Mkmanifest

The `mkmanifest` command is used to create a shell script (packing list) to restore Unix filenames. Its syntax is:

`mkmanifest` [*files*]

`Mkmanifest` creates a shell script that aids in the restoration of Unix filenames that got clobbered by the MS-DOS filename restrictions. MS-DOS filenames are restricted to 8 character names, 3 character extensions, upper case only, no device names, and no illegal characters.

The mkmanifest program is compatible with the methods used in `pcomm, arc,` and `mtools` to change perfectly good Unix filenames to fit the MS-DOS restrictions. This command is only useful if the target system which will read the diskette cannot handle VFAT long names.

4.14.1 Example

You want to copy the following Unix files to a MS-DOS diskette (using the `mcopy` command).

```
very_long_name
2.many.dots
illegal:
good.c
prn.dev
Capital
```

`ASCII` converts the names to:

```
very_lon
2xmany.dot
illegalx
good.c
xprn.dev
capital
```

The command:

```
mkmanifest very_long_name 2.many.dots illegal: good.c prn.dev Capital >manifest
```

would produce the following:

```
mv very_lon very_long_name
mv 2xmany.dot 2.many.dots
mv illegalx illegal:
mv xprn.dev prn.dev
mv capital Capital
```

Notice that "good.c" did not require any conversion, so it did not appear in the output.

Suppose I've copied these files from the diskette to another Unix system, and I now want the files back to their original names. If the file "manifest" (the output captured above) was sent along with those files, it could be used to convert the filenames.

4.14.2 Bugs

The short names generated by `mkmanifest` follow the old convention (from mtools-2.0.7) and not the one from Windows 95 and mtools-3.0.

4.15 Minfo

The `minfo` command prints the parameters of a MS-DOS file system, such as number of sectors, heads and cylinders. It also prints an mformat command line which can be used to create a similar MS-DOS file system on another media. However, this doesn't work with 2m or XDF media, and with MS-DOS 1.0 file systems

`minfo` *drive*:

Minfo supports the following option:

`v` Prints a hexdump of the boot sector, in addition to the other information

4.16 Mlabel

The `mlabel` command adds a volume label to a disk. Its syntax is:

`mlabel` [`-vcsn`] [`-N` *serial*] *drive*:[*new_label*]

`Mlabel` displays the current volume label, if present. If *new_label* is not given, and if neither the `c` nor the `s` options are set, it prompts the user for a new volume label. To delete an existing volume label, press return at the prompt.

The label is limited to 11 single-byte characters, e.g. `Name1234567`.

Reasonable care is taken to create a valid MS-DOS volume label. If an invalid label is specified, `mlabel` changes the label (and displays the new label if the verbose mode is set). `Mlabel` returns 0 on success or 1 on failure.

Mlabel supports the following options:

`c` Clears an existing label, without prompting the user

`s` Shows the existing label, without prompting the user.

`n` Assigns a new (random) serial number to the disk

`N` *serial* Sets the supplied serial number. The serial number should be supplied as an 8 digit hexadecimal number, without spaces

4.17 Mmd

The `mmd` command is used to make an MS-DOS subdirectory. Its syntax is:

`mmd` [`-D` *clash_option*] *msdosdirectory* [*msdosdirectories*...]

`Mmd` makes a new directory on an MS-DOS file system. An error occurs if the directory already exists.

4.18 Mmount

The `mmount` command is used to mount an MS-DOS disk. It is only available on Linux, as it is only useful if the OS kernel allows configuration of the disk geometry. Its syntax is:

`mmount` *msdosdrive* [*mountargs*]

`Mmount` reads the boot sector of an MS-DOS disk, configures the drive geometry, and finally mounts it passing `mountargs` to `mount`. If no mount arguments are specified, the name of the device is used. If the disk is write protected, it is automatically mounted read only.

4.19 Mmove

The `mmove` command is used to move or rename an existing MS-DOS file or subdirectory.

`mmove` [`-v`] [`-D` *clash_option*] *sourcefile targetfile*
`mmove` [`-v`] [`-D` *clash_option*] *sourcefile* [*sourcefiles*. . .] *targetdirectory*

`Mmove` moves or renames an existing MS-DOS file or subdirectory. Unlike the MS-DOS version of `MOVE`, `mmove` is able to move subdirectories. Files or directories can only be moved within one file system. Data cannot be moved from MS-DOS to Unix or vice-versa. If you omit the drive letter from the target file or directory, the same letter as for the source is assumed. If you omit the drive letter from all parameters, drive a: is assumed by default.

4.20 Mpartition

The `mpartition` command is used to create MS-DOS file systems as partitions. This is intended to be used on non-Linux systems, i.e. systems where fdisk and easy access to SCSI devices are not available. This command only works on drives whose partition variable is set.

`mpartition -p` *drive*
`mpartition -r` *drive*
`mpartition -I` [`-B` *bootSector*] *drive*
`mpartition -a` *drive*
`mpartition -d` *drive*
`mpartition -c` [`-s` *sectors*] [`-h` *heads*]
[`-t` *cylinders*] [`-v` [`-T` *type*] [`-b`
begin] [`-l` length] [`-f`]

Mpartition supports the following operations:

`p` Prints a command line to recreate the partition for the drive. Nothing is printed if the partition for the drive is not defined, or an inconsistency has been detected. If verbose (`-v`) is also set, prints the current partition table.

`r` Removes the partition described by *drive*.

`I` Initializes the partition table, and removes all partitions.

`c` Creates the partition described by *drive*.

`a` "Activates" the partition, i.e. makes it bootable. Only one partition can be bootable at a time.

`d` "Deactivates" the partition, i.e. makes it unbootable.

If no operation is given, the current settings are printed.

For partition creations, the following options are available:

`s` *sectors* The number of sectors per track of the partition (which is also the number of sectors per track for the whole drive).

`h` *heads* The number of heads of the partition (which is also the number of heads for the whole drive). By default, the geometry information (number of sectors and heads) is figured out from neighboring partition table entries, or guessed from the size.

`t` *cylinders*
: The number of cylinders of the partition (not the number of cylinders of the whole drive.

`b` *begin*
: The starting offset of the partition, expressed in sectors. If begin is not given, `mpartition` lets the partition begin at the start of the disk (partition number 1), or immediately after the end of the previous partition.

`l` *length*
: The size (length) of the partition, expressed in sectors. If end is not given, `mpartition` figures out the size from the number of sectors, heads and cylinders. If these are not given either, it gives the partition the biggest possible size, considering disk size and start of the next partition.

The following option is available for all operation which modify the partition table:

`f`
: Usually, before writing back any changes to the partition, mpartition performs certain consistency checks, such as checking for overlaps and proper alignment of the partitions. If any of these checks fails, the partition table is not changed. The `-f` allows you to override these safeguards.

The following options are available for all operations:

`v`
: Together with `-p` prints the partition table as it is now (no change operation), or as it is after it is modified.

`vv`
: If the verbosity flag is given twice, `mpartition` will print out a hexdump of the partition table when reading it from and writing it to the device.

The following option is available for partition table initialization:

`B` *bootSector*
: Reads the template master boot record from file *bootSector*.

4.20.1 Choice of partition type

Mpartition proceeds as follows to pick a type for the partition:

- FAT32 partitions are assigned type 0x0C ("`Win95 FAT32, LBA`")
- For all others, if the partition fits entirely within the first 65536 sectors of the disk, assign 0x01 ("`DOS FAT12, CHS`") for FAT12 partition and 0x04 ("`DOS FAT16, CHS`") for FAT16 partitions
- If not covered by the above, assign 0x06 ("`DOS BIG FAT16 CHS`") if partition fits entirely within the first 1024 cylinders (CHS mode)
- All remaining cases get 0x0E ("`Win95 BIG FAT16, LBA`")

If number of fat bits is not known (not specified in drive's definition), then FAT12 is assumed for all drives with less than 4096 sectors, and FAT16 for those with more than 4096 sectors.

This corresponds more or less to the definitions outlined at `https://en.wikipedia.org/wiki/Partition_type#List_of_partition_IDs` and `https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-2000-server/cc977219(v=technet.10)`, with two notable differences:

- If fat bits are unknown, the reference documents consider drives with less than 32680 sectors to be FAT12. Mtools uses 4096 sectors as the cutoff point, as older versions of

DOS only support FAT12 on disks with less than 4096 sectors (and these older versions are the ones which would be most likely to use FAT12 in the first place).

- The reference documents use a 8GB (wikipedia) or a 4GB (Microsoft) cutoff between 0x06 (`DOS BIG FAT16 CHS`) and 0x0E. Mtools uses 1024 cylinders. This is because any partition beyond 1024 cylinders must be LBA and cannot be CHS. 8GB works out to be the biggest capacity which can be represented as CHS (63 sectors, 255 heads and 1024 cylinders). 4GB is the capacity limit for windows 2000, so it makes sense that a documentation for windows 2000 would specify this as the upper limit for any partition type.

4.21 Mrd

The `mrd` command is used to remove an MS-DOS subdirectory. Its syntax is:

`mrd` [`-v`] *msdosdirectory* [*msdosdirectories*. . .]

`Mrd` removes a directory from an MS-DOS file system. An error occurs if the directory does not exist or is not empty.

4.22 Mren

The `mren` command is used to rename or move an existing MS-DOS file or subdirectory. Its syntax is:

`mren` [`-voOsSrRA`] *sourcefile targetfile*

`Mren` renames an existing file on an MS-DOS file system.

In verbose mode, `Mren` displays the new filename if the name supplied is invalid.

If the first syntax is used (only one source file), and if the target name doesn't contain any slashes or colons, the file (or subdirectory) is renamed in the same directory, instead of being moved to the current `mcd` directory as would be the case with `mmove`. Unlike the MS-DOS version of `REN`, `mren` can be used to rename directories.

4.23 Mshortname

The `mshortname` command is used to display the short name of a file. Syntax:

`mshortname` *files*

The shortname is displayed as it is stored in raw format on disk, without any character set conversion.

4.24 Mshowfat

The `mshowfat` command is used to display the FAT entries for a file. Syntax:

`mshowfat` [`-o` *offset*] *files*

If no offset is given, a list of all clusters occupied by the file is printed. If an offset is given, only the number of the cluster containing that offset (in bytes) is printed.

4.25 Mtoolstest

The `mtoolstest` command is used to tests the mtools configuration files. To invoke it, just type `mtoolstest` without any arguments. `Mtoolstest` reads the mtools configuration files, and prints the cumulative configuration to `stdout`. The output can be used as a configuration file itself (although you might want to remove redundant clauses). You may use this program to convert old-style configuration files into new style configuration files.

4.26 Mtype

The `mtype` command is used to display contents of an MS-DOS file. Its syntax is:

`mtype` [`-ts`] *msdosfile* [*msdosfiles*. . .]

`Mtype` displays the specified MS-DOS file on the screen.

In addition to the standard options, `Mtype` allows the following command line options:

`t` Text file viewing. `Mtype` translates incoming carriage return/line feeds to line feeds. Typing stops at Control-Z character.

`s` `Mtype` strips the high bit from the data.

The `mcd` command may be used to establish the device and the current working directory (relative to MS-DOS), otherwise the default is `A:/`.

`Mtype` returns 0 on success, 1 on utter failure, or 2 on partial failure.

Unlike the MS-DOS version of `TYPE`, `mtype` allows multiple arguments.

4.27 Mzip

The `mzip` command is used to issue ZIP disk specific commands on Linux, Solaris or HP-UX. Its syntax is:

`mzip` [`-epqrwx`]

`Mzip` allows the following command line options:

`e` Ejects the disk.

`f` Force eject even if the disk is mounted (must be given in addition to `-e`).

`r` Write protect the disk.

`w` Remove write protection.

`p` Password write protect.

`x` Password protect

`u` Temporarily unprotect the disk until it is ejected. The disk becomes writable, and reverts back to its old state when ejected.

`q` Queries the status

To remove the password, set it to one of the password-less modes `-r` or `-w`: mzip will then ask you for the password, and unlock the disk. If you have forgotten the password, you can get rid of it by low-level formatting the disk (using your SCSI adapter's BIOS setup).

The ZipTools disk shipped with the drive is also password protected. On MS-DOS or on a Mac, this password is automatically removed once the ZipTools have been installed. From various articles posted to Usenet, I learned that the password for the tools disk is `APlaceForYourStuff`[1]. Mzip knows about this password, and tries it first, before prompting you for a password. Thus `mzip -w z:` unlocks the tools disk[2]. The tools disk is formatted in a special way so as to be usable both in a PC and in a Mac. On a PC, the Mac file system appears as a hidden file named `partishn.mac`. You may erase it to reclaim the 50 Megs of space taken up by the Mac file system.

4.27.1 Bugs

This command is a big kludge. A proper implementation would take a rework of significant parts of mtools, but unfortunately I don't have the time for this right now. The main downside of this implementation is that it is inefficient on some architectures (several successive calls to mtools, which defeats mtools' caching).

[1] To see the articles, search for `APlaceForYourStuff` using Google Groups

[2] I didn't know about this yet when I bought my own Zip drive. Thus I ended up reformatting my tools disk, and hence I haven't had the opportunity to test the password yet. If anybody still has their tools disk with the original password, could you try it out? Thanks in advance

5 Architecture specific compilation flags

To compile mtools, first invoke `./configure` before `make`. In addition to the standard `autoconfigure` flags, there are two architecture specific flags available.

`./configure --enable-xdf`
`./configure --disable-xdf`
: Enables support for XDF disks. This is on by default. See Section 2.8.4 [XDF], page 7, for details.

`./configure --enable-vold`
`./configure --disable-vold`
: Enables support for vold on Solaris. When used in conjunction with vold, mtools should use different device nodes than for direct access.

`./configure --enable-new-vold`
`./configure --disable-new-vold`
: Enables new support for vold on Solaris. This is supposed to work more smoothly than the old support.

`./configure --enable-floppyd`
`./configure --disable-floppyd`
: Enables support for floppyd. By default, floppyd support is enabled as long as the necessary X includes and libraries are available.

6 Porting mtools to architectures which are not supported yet

This chapter is only interesting for those who want to port mtools to an architecture which is not yet supported. For most common systems, default drives are already defined. If you want to add default drives for a still unsupported system, run configuration.guess, to see which identification autoconf uses for that system. This identification is of the form cpu-vendor-os (for example sparc-sun-sunos). The cpu and the OS parts are passed to the compiler as preprocessor flags. The OS part is passed to the compiler in three forms.

1. The complete OS name, with dots replaced by underscores. SCO3.2v2 would yield sco3_2v2
2. The base OS name. SCO3.2v2 would yield Sco
3. The base OS name plus its major version. SCO3.2v2 would yield Sco3

All three versions are passed, if they are different.

To define the devices, use the entries for the systems that are already present as templates. In general, they have the following form:

```
#if (defined (my_cpu) && defined(my_os))
#define predefined_devices
struct device devices[] = {
        { "/dev/first_drive", 'drive_letter', drive_description},
        ...
        { "/dev/last_drive", 'drive_letter', drive_description}
}
#define INIT_NOOP
#endif
```

"/dev/first_drive" is the name of the device or image file representing the drive. Drive_letter is a letter ranging from a to z giving access to the drive. Drive_description describes the type of the drive:

`ED312` extra density (2.88M) 3 1/2 disk

`HD312` high density 3 1/2 disk

`DD312` double density 3 1/2 disk

`HD514` high density 5 1/4 disk

`DD514` double density 5 1/4 disk

`DDsmall` 8 sector double density 5 1/4 disk

`SS514` single sided double density 5 1/4 disk

`SSsmall` single sided 8 sector double density 5 1/4 disk

`GENFD` generic floppy drive (12 bit FAT)

`GENHD` generic hard disk (16 bit FAT)

`GEN` generic device (all parameters match)

`ZIPJAZ(flags)`
generic ZIP drive using normal access. This uses partition 4. `Flags` are any special flags to be passed to open.

`RZIPJAZ(flags)`
: generic ZIP drive using raw SCSI access. This uses partition 4. `Flags` are any special flags to be passed to open.

`REMOTE`
: the remote drive used for floppyd. Unlike the other items, this macro also includes the file name ($DISPLAY) and the drive letter (X)

Entries may be described in more detail:

```
fat_bits,open_flags,cylinders,heads,sectors,DEF_ARG
```

or, if you need to describe an offset (file system doesn't start at beginning of file system)

```
fat_bits, open_flags, cylinders, heads, sectors, offset, DEF_ARG0
```

`fat_bits`
: is either 12, 16 or 0. 0 means that the device accepts both types of FAT.

`open_flags`
: may include flags such as O_NDELAY, or O_RDONLY, which might be necessary to open the device. 0 means no special flags are needed.

`cylinders,heads,sectors`
: describe the geometry of the disk. If cylinders is 0, the heads and sectors parameters are ignored, and the drive accepts any geometry.

`offset`
: is used if the DOS file system doesn't begin at the start of the device or image file. This is mostly useful for Atari Ram disks (which contain their device driver at the beginning of the file) or for DOS emulator images (which may represent a partitioned device.

Definition of defaults in the devices file should only be done if these same devices are found on a large number of hosts of this type. In that case, could you also let me know about your new definitions, so that I can include them into the next release. For purely local file, I recommend that you use the `/etc/mtools.conf` and `~/.mtoolsrc` configuration files.

However, the devices files also allows you to supply geometry setting routines. These are necessary if you want to access high capacity disks.

Two routines should be supplied:

1. Reading the current parameters

   ```
   static inline int get_parameters(int fd, struct generic_floppy_struct *floppy)
   ```

   This probes the current configured geometry, and return it in the structure generic_floppy_struct (which must also be declared). Fd is an open file descriptor for the device, and buf is an already filled in stat structure, which may be useful. This routine should return 1 if the probing fails, and 0 otherwise.

2. Setting new parameters

   ```
   static inline int set_parameters(int fd, struct generic_floppy_struct *floppy)
                                            struct stat *buf)
   ```

   This configures the geometry contained in floppy on the file descriptor fd. Buf is the result of a stat call (already filled in). This should return 1 if the new geometry cannot be configured, and 0 otherwise.

A certain number of preprocessor macros should also be supplied:

`TRACKS(floppy)`
: refers to the track field in the floppy structure

`HEADS(floppy)`
: refers to the heads field in the floppy structure

`SECTORS(floppy)`
: refers to the sectors per track field in the floppy structure

`SECTORS_PER_DISK(floppy)`
: refers to the sectors per disk field in the floppy structure (if applicable, otherwise leave undefined)

`BLOCK_MAJOR`
: major number of the floppy device, when viewed as a block device

`CHAR_MAJOR`
: major number of the floppy device, when viewed as a character device (a.k.a. "raw" device, used for fsck) (leave this undefined, if your OS doesn't have raw devices)

For the truly high capacity formats (XDF, 2m, etc), there is no clean and documented interface yet.

Command Index

F

floppyd 17
floppyd_installtest 18

M

mattrib 19
mbadblocks 19
mcat 20
mcd 20
mcd (introduction) 4
mcopy 21
mdel 22
mdeltree 22
mdir 22
mdoctorfat 23
mdu 23
mformat 23
mformat (geometry used for) 11
minfo 28
mkmanifest 27
mlabel 28
mmd 28
mmount 28
mmove 29
mpartition 29
mrd 31
mren 31
mshortname 31
mshowfat 31
mtoolstest 32
mzip 32

Variable index

C

cylinders 12

D

drive 10

E

exclusive 13

F

fat_bits 13
file 11
filter 15

H

heads 12

M

mformat_only 15
MTOOLS_DOTTED_DIR 9
MTOOLS_FAT_COMPATIBILITY 9
MTOOLS_LOCK_TIMEOUT 9
MTOOLS_LOWER_CASE 9
MTOOLS_NAME_NUMERIC_TAIL 9
MTOOLS_NO_VFAT 9
MTOOLS_SKIP_CHECK 9
MTOOLS_TWENTY_FOUR_HOUR_CLOCK 9
MTOOLSRC 9

N

nodelay 13

S

sectors 12
sync 13

T

tracks 12

U

use_xdf 15

Concept index

2

2m 7

A

ALPHA patches 2
APlaceForYourStuff 32
Archive bit 19
Atari 15
Atari Ram disk 11

B

Backwards compatibility 16
Bad blocks 19
bigger sectors 7
blocksize 14
bugs 2

C

Case sensitivity 6
Changing file attributes 19
character devices 14
Checking configuration file 32
Clusters of a file 23, 31
Command list 17
Compile time configuration 34
Compiled-in defaults 35
Concatenating MS-DOS files 21
Configuration file 9
Configuration file name 9
Configuration file name (parsing order) 16
Configuration file parsing order 16
Configuration file syntax 9
Configuration file, old syntax 16
Configuration files 9
Configuration of disk geometry 11
Copying an entire disk image 20
Copying MS-DOS files 21
CR/LF conversions 21
Creating a directory 28
Current working directory 4
Current working directory (changing the) 20

D

Default configuration 9
Default directory 4
Default directory (changing the) 20
Default values 9
Deleting a directory 31
deleting an MS-DOS directory recursively 22
deleting MS-DOS files 22
Description of disk geometry 11
diffs 2
Directory 4
Directory (changing) 20
Directory creation 28
Directory listing 22
Directory removing 31
disable locking 14
Disk Geometry 11
Disk image 20
Disk label 28
DMF disks 7
DOSEMU hard disk image 11
Drive configuration 10
Drive configuration, example 10
Drive description 10
Drive description, example 10
Drive independent configuration variables 9
du 23
Duplicate file names 5

E

Ejecting a Zip/Jaz disk 32
Environmental variables 9
Erasing a directory 31
erasing an MS-DOS directory recursively 22
erasing MS-DOS files 22
exclusive access to a drive 13
Executing commands after closing the device ... 14
Executing commands before opening the device. 14

F

Fat 23, 31
fdformat 7
File name of device node 11
File system creation 23
Filenames 3
Floppyd cat 20
Format of disk 11
Formats, high capacity 6
Formatting disks 23
FreeDOS 9

G

getting parameters of a MS-DOS file system 28
Global configuration variables 9

H

Hdimage 11
Hidden files 19
High capacity formats 6
High capacity formats, mounting 28
High density disk 11

I

Image file 11
Initializing disks 23

J

Jaz disk (utilities) 32
Jaz disks (partitioning them) 29
Jaz disks (partitions) 11
Jaz disks (raw SCSI access) 14

L

Labeling a disk 28
Linux enhancements (High Capacity Formats) ... 6
Linux enhancements (mmount) 28
List of available commands 17
Listing a directory 22
Listing space occupied by directories and files .. 23
Location of configuration files 9
Location of configuration files (parsing order) ... 16
locking (disabling it) 14
Long file name 4
Low density disk 11

M

Magneto-optical disks 11
mailing list 2
Making a directory 28
Marking blocks as bad 19
Mcwd file 20
mformat parameters 28
Mounting a disk 28
Moving files (mmove) 29
Moving files (mren) 31

N

Name clashes 5
Name of configuration files 9
Name of configuration files (parsing order) 16
Name of device node 11

O

Occupation of space by directories and files 23
Odd formats 6
Old configuration file syntax 16
open flags 13
Options 3
OS/2 (layout of removable media) 11
OS/2 (XDF disks) 7
Overwriting files 5

P

packing list 27
Parsing order 16
Partitioned image file 11
partitions (creating) 29
password protected Zip disks 32
patches 2
plain floppy: device xxx busy 14
Porting 35
Primary file name (long names) 4
Primary file name (name clashes) 5

R

Ram disk 11
raw device 14
Read errors 19
Read-only files (changing the attribute) 19
Read-only files (listing them) 22
Reading MS-DOS files 21
recursively removing an MS-DOS directory 22
remote floppy access 17, 18
Removable media 11
Removing a directory 31
removing an MS-DOS directory recursively 22
removing MS-DOS files 22
Renaming files (mmove) 29
Renaming files (mren) 31

S

SCSI devices ... 14
Secondary file name (long names) ... 4
Secondary file name (name clashes) ... 5
setgid installation ... 14
setuid installation ... 14
setuid installation (needed for raw SCSI I/O) ... 14
Solaris (compile time configuration of vold) ... 34
Solaris (Raw access to SCSI devices such as Zip & Jaz) ... 14
Solaris (volcheck) ... 14
Solaris (vold) ... 15
Space occupied by directories and files ... 23
Special formats ... 6
Subdirectory creation ... 28
Subdirectory removing ... 31
SunOS (Raw access to SCSI devices such as Zip & Jaz) ... 14
synchronous writing ... 13
Syntax of the configuration file ... 9
Syquest disks ... 11
Syquest disks (raw SCSI access) ... 14
System files ... 19

T

Testing configuration file for correctness ... 32
Text files ... 21
Tools disk (Zip and Jaz drives) ... 32

V

Verifying configuration file ... 32
VFAT-style file names ... 4
vgacopy ... 7
Vold (compile time configuration) ... 34
Vold (mediamgr) ... 15

W

Weird formats ... 6
Windows 95 (DMF disks) ... 7
Windows 95-style file names ... 4
Windows NT (layout of removable media) ... 11
Wordswapped ... 15
Working directory ... 4, 20
Write protecting a Zip/Jaz disk ... 32
Writing MS-DOS files ... 21

X

X terminal ... 17, 18
XDF disks ... 7
XDF disks (compile time configuration) ... 34
XDF disks (how to configure) ... 15

Z

Zip disk (utilities) ... 32
Zip disks (partitioning them) ... 29
Zip disks (partitions) ... 11
Zip disks (raw SCSI access) ... 14
ZipTools disk ... 32