Mtools

Accessing MS-DOS disks

Edition 3.0, for Mtools version 3.0

May 1996

by Alain Knaff

Introduction

Mtools is a collection of tools for manipulating MS-DOS files. Mtools is a public domain collection of programs to allow Unix systems to read, write, and manipulate files on an MS-DOS filesystem (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.

1 Common features of all mtools commands

1.1 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 1.4 [name clashes], page 2, for more details on these. All commands accept the `-V` flags 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 3 [Commands], page 12, for a description of the options which are specific to each command.

1.2 Current working directory

The `mcd` command (Section 3.3 [mcd], page 13) is used to establish the device and the current working directory (relative to the MS-DOS filesystem), 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.

1.3 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 replaces 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,

1.4 Name clashes

When writing a file to disk, its long name (primary 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`, `mwrite` and `mread`. 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:

`-o`	Overwrites primary names by default.
`-O`	Overwrites secondary names by default.
`-r`	Renames primary name by default.
`-R`	Renames secondary name by default.
`-a`	Autorenames primary name by default.
`-A`	Autorenames secondary name by default.
`-s`	Skip primary name by default.
`-S`	Skip secondary name by default.
`-m`	Ask user what to do with primary name.
`-M`	Ask user what to do with secondary name.

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.

1.5 Case sensitivity of the VFAT filesystem

The VFAT filesystem 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 filesystem, 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 filesystem allows 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.

1.6 XDF Disks (Only available on Linux)

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 4 [Compiling mtools], page 18, and Section 2.6 [misc variables], page 8, for details on how to do this. Fast XDF access is only available for Linux kernels which are more recent than 1.1.34.

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.

1.7 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 2.4 [global variables], page 5)

1.8 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 2.4 [global variables], page 5) to bypass the fat checking.

The support for non-Linux OS variants has not been tested for a long time. It may contain bugs, or even not work at all.

2 How to configure mtools for your environment

2.1 Description

This sections explains the syntax of the configurations files for mtools. The configuration files are called `/etc/mtools.conf` and `~/.mtoolsrc`. These configuration files describe the following items:

- Global configuration flags and variables
- Per drive flags and variables
- Character translation tables

2.2 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/defaults/mtools.conf` instead.

2.2.1 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).

2.3 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.

2.4 Global variables

Global variables 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.

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
```

2.5 Per drive flags and variables

2.5.1 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
```

2.5.2 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 3.8 [mformat], page 14, for details.

filtering
: On some Unices 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 2.6.1 [multiple descriptions], page 8, 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).

initial geometry
: The geometry information (if available) is also used to set the initial geometry on configurable device nodes. This initial geometry is used to read the boot sector, which contains the real geometry. If no geometry information is supplied in the configuration file, no initial configuration is done. On Linux, this is not really needed either, 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 don't use geometry configuration unless you actually need it.

The following geometry related variables are available:

`cylinders`
`cylinders`
: 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
```

The following shorthand geometry descriptions are available:

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

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

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

`360k`
: double density 5 1/4 disk. Equivalent to: `fat_bits=12 tracks=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 tracks=40 heads=2 sectors=8`

2.5.3 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.

2.6 General Purpose Drive Variables

The following general purpose drive variables are available:

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

`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 1.6 [XDF disks], page 4, for more details.

`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 Syquests, ZIP drives, and magneto-optical disks. Although traditional DOS sees Syquests and magneto-optical disks as '`giant floppy disks`' which are unpartitioned, OS/2 and Windows NT treat them like hard disks, i.e. partioned 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 filesystem starts. This is useful for logical partitions in DOSEMU hdimages, and for ATARI ram disks. By default, this is zero, meaning that the filesystem start right at the beginning of the device or file.

`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 autodetected number of fat bits wrong, or if you want to mformat a disk with a weird number of fat bits.

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

2.6.1 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.

2.7 Character set translation tables

If you live in the USA, in Western Europe or in Australia, you may skip this section.

2.7.1 Why character set translation tables are needed

DOS uses a different character code mapping than Unix. 7-bit characters still have the same meaning, only characters with the eight bit set are affected. To make matters worse, there are several translation tables available depending on the country where you are. The appearance of the characters is defined using code pages. These code pages aren't the same for all countries. For instance, some code pages don't contain upper case accented characters. On the other hand, some code pages contain characters which don't exist in Unix, such as certain line-drawing characters or accented consonants used by some Eastern European countries. This affects two things, relating to filenames:

upper case characters
: In short names, only upper case characters are allowed. This also holds for accented characters. For instance, in a code page which doesn't contain accented uppercase characters, the accented lowercase characters get transformed into their unaccented counterparts.

long file names
: Micro$oft has finally come to their senses and uses a more standard mapping for the long file names. They use Unicode, which is basically a 32 bit version of ASCII. Its first 256 characters are identical to Unix ASCII. Thus, the code page also affects the correspondence between the codes used in long names and those used in short names

Mtools considers the filenames entered on the command line as having the Unix mapping, and translates the characters to get short names. By default, code page 850 is used with the Swiss uppercase/lowercase mapping. I chose this code page, because its set of existing characters most closely matches Unix's. Moreover, this code page covers most characters in use in the USA, Australia and Western Europe. However, it is still possible to chose a different mapping. There are two methods: the `country` variable and explicit tables.

2.7.2 Configuration using Country

The `COUNTRY` variable is recommended for people which also have access to MS-DOS system files and documentation. If you don't have access to these, I'd suggest you'd rather use explicit tables instead.

Syntax:

`COUNTRY="`*country*`[,[`*codepage*`],` *country-file*`]"`

This tells mtools to use a Unix-to-DOS translation table which matches *codepage* and an lowercase-to-uppercase table for *country* and to use the *country-file* file to get the lowercase-to-uppercase table. The country code is most often the telephone prefix of the country. Refer to the DOS help page on "country" for more details. The *codepage* and the *country-file* parameters are optional. Please don't type in the square brackets, they are only there to say which parameters are optional. The *country-file* file is supplied with MS-DOS, and is usually called `COUNTRY.SYS`, and stored in the `C:\DOS` directory. In most cases you don't need it, as the most common translation tables are compiled into mtools. So, don't worry if you run a Unix-only box which lacks this file.

If *codepage* is not given, a per country default code page is used. If the *country-file* parameter isn't given, compiled-in defaults are used for the lowercase-to-uppercase table. This is useful for other Unices than Linux, which may have no `COUNTRY.SYS` file available online.

The Unix-to-DOS are not contained in the `COUNTRY.SYS` file, and thus mtools always uses compiled-in defaults for those. Thus, only a limited amount of code pages are supported. If your preferred code page is missing, or if you know the name of the Windows 95 file which contains this mapping, could you please drop me a line at `Alain.Knaff@inrialpes.fr`.

The `COUNTRY` variable can also be set using the environment.

2.7.3 Configuration using explicit translation tables

Translation tables may be described in line in the configuration file. Two tables are needed: first the DOS-to-Unix table, and then the Lowercase-to-Uppercase table. A DOS-to-Unix table starts with the `tounix` keyword, followed by a colon, and 128 hexadecimal numbers. A lower-to-upper table starts with the `fucase` keyword, followed by a colon, and 128 hexadecimal numbers.

The tables only show the translations for characters whose codes is greater than 128, because translation for lower codes is trivial.

Example:

```
tounix:
  0xc7 0xfc 0xe9 0xe2 0xe4 0xe0 0xe5 0xe7
  0xea 0xeb 0xe8 0xef 0xee 0xec 0xc4 0xc5
  0xc9 0xe6 0xc6 0xf4 0xf6 0xf2 0xfb 0xf9
  0xff 0xd6 0xdc 0xf8 0xa3 0xd8 0xd7 0x5f
  0xe1 0xed 0xf3 0xfa 0xf1 0xd1 0xaa 0xba
  0xbf 0xae 0xac 0xbd 0xbc 0xa1 0xab 0xbb
  0x5f 0x5f 0x5f 0x5f 0x5f 0xc1 0xc2 0xc0
  0xa9 0x5f 0x5f 0x5f 0x5f 0xa2 0xa5 0xac
  0x5f 0x5f 0x5f 0x5f 0x5f 0x5f 0xe3 0xc3
  0x5f 0x5f 0x5f 0x5f 0x5f 0x5f 0x5f 0xa4
  0xf0 0xd0 0xc9 0xcb 0xc8 0x69 0xcd 0xce
  0xcf 0x5f 0x5f 0x5f 0x5f 0x7c 0x49 0x5f
  0xd3 0xdf 0xd4 0xd2 0xf5 0xd5 0xb5 0xfe
  0xde 0xda 0xd9 0xfd 0xdd 0xde 0xaf 0xb4
```

```
    0xad 0xb1 0x5f 0xbe 0xb6 0xa7 0xf7 0xb8
    0xb0 0xa8 0xb7 0xb9 0xb3 0xb2 0x5f 0x5f

  fucase:
    0x80 0x9a 0x90 0xb6 0x8e 0xb7 0x8f 0x80
    0xd2 0xd3 0xd4 0xd8 0xd7 0xde 0x8e 0x8f
    0x90 0x92 0x92 0xe2 0x99 0xe3 0xea 0xeb
    0x59 0x99 0x9a 0x9d 0x9c 0x9d 0x9e 0x9f
    0xb5 0xd6 0xe0 0xe9 0xa5 0xa5 0xa6 0xa7
    0xa8 0xa9 0xaa 0xab 0xac 0xad 0xae 0xaf
    0xb0 0xb1 0xb2 0xb3 0xb4 0xb5 0xb6 0xb7
    0xb8 0xb9 0xba 0xbb 0xbc 0xbd 0xbe 0xbf
    0xc0 0xc1 0xc2 0xc3 0xc4 0xc5 0xc7 0xc7
    0xc8 0xc9 0xca 0xcb 0xcc 0xcd 0xce 0xcf
    0xd1 0xd1 0xd2 0xd3 0xd4 0x49 0xd6 0xd7
    0xd8 0xd9 0xda 0xdb 0xdc 0xdd 0xde 0xdf
    0xe0 0xe1 0xe2 0xe3 0xe5 0xe5 0xe6 0xe8
    0xe8 0xe9 0xea 0xeb 0xed 0xed 0xee 0xef
    0xf0 0xf1 0xf2 0xf3 0xf4 0xf5 0xf6 0xf7
    0xf8 0xf9 0xfa 0xfb 0xfc 0xfd 0xfe 0xff
```

The first table maps DOS character codes to Unix character codes. For example, the DOS character number 129. This is a u with to dots on top of it. To translate it into Unix, we look at the character number 1 in the first table (1 = 129 - 128). This is 0xfc. (Beware, numbering starts at 0). The second table maps lower case DOS characters to upper case DOS characters. The same lower case u with dots maps to character 0x9a, which is an uppercase U with dots in DOS.

2.7.4 Unicode characters greater than 256

If an existing MS-DOS name contains Unicode character greater than 256, these are translated to underscores or to characters which are close in visual appearance. For example, accented consonants are translated into their unaccented counterparts. This translation is used for mdir and for the Unix filenames generated by mcopy. Linux does support Unicode too, but unfortunately too few applications support it yet to bother with it in mtools. Most importantly, xterm can't display Unicode yet. If there is sufficient demand, I might include support for Unicode in the Unix filenames as well.

Caution: When deleting files with mtools, the underscore matches all characters which can't be represented in Unix. Be careful with mdel!

2.8 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. `/etc/mtools` This is for backwards compatibility only, and is only parsed if `mtools.conf` doesn't exist.
4. `~/.mtoolsrc`.

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.

2.9 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.

3 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 1.1 [arguments], page 1, for a description of those.

3.1 Mattrib

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

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

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

3.2 Mbadblocks

The `mbadblocks` command is used to scan an MS-DOS floppy and mark its unused bad blocks as bad. It uses the following syntax:

`mbadblocks` *drive*`:`

`Mbadblocks` scans an MS-DOS floppy for bad blocks. All unused bad blocks are marked as such in the FAT. This is intended to be used right after `mformat`. It is not intended to salvage bad disks.

3.2.1 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.

3.3 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 filesystem.

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.

3.4 Mcopy

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

```
    mcopy [-tnvmoOsSrRA] sourcefile targetfile
    mcopy [-tnvmoOsSrRA] 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.

`n` No warning. `Mcopy` doesn't warn the user when overwriting an existing file.

`m` Preserve the file modification time. 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 (Section 1.4 [name clashes], page 2) for details).

3.4.1 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
```

3.5 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 filesystem.

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

3.6 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 filesystem. An error occurs if the directory to be removed does not exist.

3.7 Mdir

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

`mdir` [`-w`] *msdosdirectory*

`mdir` [`-w`] [`-a`] *msdosfile* [*msdosfiles*. . .]

`Mdir` displays the contents of an MS-DOS directory.

`Mdir` supports the following command line options:

`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.

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

3.8 Mformat

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

```
mformat [-t tracks] [-h heads] [-s sectors] [-l volume_label] [-S sizecode]
[-2 sectors_on_track_0] [-M software_sector_size] [-a] [-X] [-C] [-H hidden_sectors]
drive:
```

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

The follow 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)

`t` The number of tracks (not cylinders).

`h`	The number of heads (sides).
`s`	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). If the 2m option is not given, number of physical sectors per track (which may be bigger than 512 bytes).
`l`	An optional volume label.
`S`	The sizecode. The size of the sector is 2 ^ (sizecode + 7).
`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.
`1`	don't use a 2m format, even if the current geometry of the disk is a 2m geometry.
`M`	software sector size. This parameter describes the sector size in bytes used by the MS-DOS filesystem. By default it is the physical sector size.
`a`	If this option is given, an Atari style serial number is generated. Ataris store their serial number in the OEM label.
`X`	formats the disk as an XDF disk. See Section 1.6 [XDF disks], page 4, for more details. The disk has first to be low-level formatted using the xdfcopy utility included in the fdutils package.
`C`	creates the disk image file to install the MS-DOS filesystem on it. Obviously, this is useless on physical devices such as floppies and hard disk partitions.
`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.
`n`	serial number.

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 `mkmanifest` for that.

3.9 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.

3.9.1 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
```

`Mcopy` 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.

3.9.2 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.

3.10 Mlabel

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

```
mlabel [-vcs] 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.

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.

3.11 Mmd

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

`mmd` [`-voOsSrRA`] *msdosdirectory* [*msdosdirectories*...]

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

3.12 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 to configure 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.

3.13 Mmove

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

```
mmove [-voOsSrRA] sourcefile targetfile
mmove [-voOsSrRA] 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.

3.14 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 filesystem. An error occurs if the directory does not exist or is not empty.

3.15 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 filesystem.

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

If the first syntax is used (only one sourcefile), 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.

3.16 Mtest

The `mtest` command is used to tests the mtools configuration files. To invoke it, just type `mtest` without any arguments. `Mtest` 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.

3.17 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.

`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 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 1.6 [XDF disks], page 4, for details.

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

5 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 config.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)

Entries may be described in more detail:

```
fat_bits,open_flags,tracks,heads,sectors,offset,DEF_ARG
```

`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.

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

`offset` is used if the DOS filesystem 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. For purely local file, I recommend that you use the /etc/mtools.conf and ~/.mtoolsrc configuration files.

However, the devices files also allows 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

Variable index

Concept index