

VFAT

USING VFAT

To use the vfat filesystem, use the filesystem type 'vfat'. i.e.:

```
mount -t vfat /dev/fd0 /mnt
```

No special partition formatter is required, 'mkdosfs' will work fine if you want to format from within Linux.

VFAT MOUNT OPTIONS

uid=###

Set the owner of all files on this filesystem. The default is the uid of current process.

gid=###

Set the group of all files on this filesystem. The default is the gid of current process.

umask=###

The permission mask (for files and directories, see *umask(1)*). The default is the umask of current process.

dmask=###

The permission mask for the directory. The default is the umask of current process.

fmask=###

The permission mask for files. The default is the umask of current process.

allow_utime=###

This option controls the permission check of mtime/atime.

-20: If current process is in group of file's group ID, you can change timestamp.

-2: Other users can change timestamp.

The default is set from dmask option. If the directory is writable, utime(2) is also allowed. i.e. ~dmask & 022.

Normally utime(2) checks current process is owner of the file, or it has CAP_FOWNER capability. But FAT filesystem doesn't have uid/gid on disk, so normal check is too unflexible. With this option you can relax it.

codepage=###

Sets the codepage number for converting to shortname characters on FAT filesystem. By default, FAT_DEFAULT_CODEPAGE setting is used.

iocharset=<name>

Character set to use for converting between the encoding is used for user visible filename and 16 bit Unicode characters. Long filenames are stored on disk in Unicode format, but Unix for the most part doesn't know how to deal with Unicode. By default, FAT_DEFAULT_IOCHARSET setting is used.

There is also an option of doing UTF-8 translations with the utf8 option.

Note

iocharset=utf8 is not recommended. If unsure, you should consider the utf8 option instead.

utf8=<bool>

UTF-8 is the filesystem safe version of Unicode that is used by the console. It can be enabled or disabled for the filesystem with this option. If 'uni_xlate' gets set, UTF-8 gets disabled. By default, FAT_DEFAULT_UTF8 setting is used.

uni_xlate=<bool>

Translate unhandled Unicode characters to special escaped sequences. This would let you backup and restore filenames that are created with any Unicode characters. Until Linux supports Unicode for real, this gives you an alternative. Without this option, a '?' is used when no translation is possible. The escape character is ':' because it is otherwise illegal on the vfat filesystem. The escape sequence that gets used is ':' and the four digits of hexadecimal unicode.

nonumtail=<bool>

When creating 8.3 aliases, normally the alias will end in '~1' or tilde followed by some number. If this option is set, then if the filename is "longfilename.txt" and "longfile.txt" does not currently exist in the directory, longfile.txt will be the short alias

instead of longfi~1.txt.

usefree

Use the "free clusters" value stored on FSINFO. It will be used to determine number of free clusters without scanning disk. But it's not used by default, because recent Windows don't update it correctly in some case. If you are sure the "free clusters" on FSINFO is correct, by this option you can avoid scanning disk.

quiet

Stops printing certain warning messages.

check=s|r|n

Case sensitivity checking setting.

s: strict, case sensitive

r: relaxed, case insensitive

n: normal, default setting, currently case insensitive

nocase

This was deprecated for vfat. Use `shortname=win95` instead.

shortname=lower|win95|winnt|mixed

Shortname display/create setting.

lower: convert to lowercase for display, emulate the Windows 95 rule for create.

win95: emulate the Windows 95 rule for display/create.

winnt: emulate the Windows NT rule for display/create.

mixed: emulate the Windows NT rule for display, emulate the Windows 95 rule for create.

Default setting is *mixed*.

tz=UTC

Interpret timestamps as UTC rather than local time. This option disables the conversion of timestamps between local time (as used by Windows on FAT) and UTC (which Linux uses internally). This is particularly useful when mounting devices (like digital cameras) that are set to UTC in order to avoid the pitfalls of local time.

time_offset=minutes

Set offset for conversion of timestamps from local time used by FAT to UTC. I.e. <minutes> minutes will be subtracted from each timestamp to convert it to UTC used internally by Linux. This is useful when time zone set in `sys_tz` is not the time zone used by the filesystem. Note that this option still does not provide correct time stamps in all cases in presence of DST - time stamps in a different DST setting will be off by one hour.

showexec

If set, the execute permission bits of the file will be allowed only if the extension part of the name is .EXE, .COM, or .BAT. Not set by default.

debug

Can be set, but unused by the current implementation.

sys_immutable

If set, ATTR_SYS attribute on FAT is handled as IMMUTABLE flag on Linux. Not set by default.

flush

If set, the filesystem will try to flush to disk more early than normal. Not set by default.

rodir

FAT has the ATTR_RO (read-only) attribute. On Windows, the ATTR_RO of the directory will just be ignored, and is used only by applications as a flag (e.g. it's set for the customized folder).

If you want to use ATTR_RO as read-only flag even for the directory, set this option.

errors=panic|continue|remount-ro

specify FAT behavior on critical errors: panic, continue without doing anything or remount the partition in read-only mode (default behavior).

discard

If set, issues discard/TRIM commands to the block device when blocks are freed. This is useful for SSD devices and sparse/thinly-provisioned LUNs.

nfs=stale_rw|nostale_ro

Enable this only if you want to export the FAT filesystem over NFS.

stale_rw: This option maintains an index (cache) of directory *inodes* by *i_logstart* which is used by the nfs-related code to improve look-ups. Full file operations (read/write) over NFS is supported but with cache eviction at NFS server, this could result in ESTALE issues.

nostale_ro: This option bases the *inode* number and filehandle on the on-disk location of a file in the MS-DOS directory entry. This ensures that ESTALE will not be returned after a file is evicted from the inode cache. However, it means that operations such as rename, create and unlink could cause filehandles that previously pointed at one file to point at a different file, potentially causing data corruption. For this reason, this option also mounts the filesystem readonly.

To maintain backward compatibility, '-o nfs' is also accepted, defaulting to "stale_rw".

dos1xfloppy <bool>: 0,1,yes,no,true,false

If set, use a fallback default BIOS Parameter Block configuration, determined by backing device size. These static parameters match defaults assumed by DOS 1.x for 160 kiB, 180 kiB, 320 kiB, and 360 kiB floppies and floppy images.

LIMITATION

The fallocated region of file is discarded at umount/evict time when using fallocate with FALLOC_FL_KEEP_SIZE. So, User should assume that fallocated region can be discarded at last close if there is memory pressure resulting in eviction of the inode from the memory. As a result, for any dependency on the fallocated region, user should make sure to recheck fallocate after reopening the file.

TODO

Need to get rid of the raw scanning stuff. Instead, always use a get next directory entry approach. The only thing left that uses raw scanning is the directory renaming code.

POSSIBLE PROBLEMS

- vfat_valid_longname does not properly checked reserved names.
- When a volume name is the same as a directory name in the root directory of the filesystem, the directory name sometimes shows up as an empty file.
- autoconv option does not work correctly.

TEST SUITE

If you plan to make any modifications to the vfat filesystem, please get the test suite that comes with the vfat distribution at

http://web.archive.org/web/*/http://bmrc.berkeley.edu/people/chaffee/vfat.html

This tests quite a few parts of the vfat filesystem and additional tests for new features or untested features would be appreciated.

NOTES ON THE STRUCTURE OF THE VFAT FILESYSTEM

This documentation was provided by Galen C. Hunt gchunt@cs.rochester.edu and lightly annotated by Gordon Chaffee.

This document presents a very rough, technical overview of my knowledge of the extended FAT file system used in Windows NT 3.5 and Windows 95. I don't guarantee that any of the following is correct, but it appears to be so.

The extended FAT file system is almost identical to the FAT file system used in DOS versions up to and including 6.223410239847 :-). The significant change has been the addition of long file names. These names support up to 255 characters including spaces and lower case characters as opposed to the traditional 8.3 short names.

Here is the description of the traditional FAT entry in the current Windows 95 filesystem:

```
struct directory { // Short 8.3 names
    unsigned char name[8];           // file name
    unsigned char ext[3];            // file extension
    unsigned char attr;              // attribute byte
    unsigned char lcase;             // Case for base and extension
    unsigned char ctime_ms;          // Creation time, milliseconds
    unsigned char ctime[2];          // Creation time
    unsigned char cdate[2];          // Creation date
    unsigned char adate[2];          // Last access date
    unsigned char reserved[2];       // reserved values (ignored)
    unsigned char time[2];           // time stamp
    unsigned char date[2];           // date stamp
    unsigned char start[2];          // starting cluster number
    unsigned char size[4];           // size of the file
};
```

The lcase field specifies if the base and/or the extension of an 8.3 name should be capitalized. This field does not seem to be used by Windows 95 but it is used by Windows NT. The case of filenames is not completely compatible from Windows NT to Windows 95. It is not completely compatible in the reverse direction, however. Filenames that fit in the 8.3 namespace and are written on Windows NT to be lowercase will show up as uppercase on Windows 95.

Note

Note that the `start` and `size` values are actually little endian integer values. The descriptions of the fields in this structure are public knowledge and can be found elsewhere.

With the extended FAT system, Microsoft has inserted extra directory entries for any files with extended names. (Any name which legally fits within the old 8.3 encoding scheme does not have extra entries.) I call these extra entries slots. Basically, a slot is a specially formatted directory entry which holds up to 13 characters of a file's extended name. Think of slots as additional labeling for the directory entry of the file to which they correspond. Microsoft prefers to refer to the 8.3 entry for a file as its alias and the extended slot directory entries as the file name.

The C structure for a slot directory entry follows:

```
struct slot { // Up to 13 characters of a long name
    unsigned char id;           // sequence number for slot
    unsigned char name0_4[10];  // first 5 characters in name
    unsigned char attr;         // attribute byte
    unsigned char reserved;     // always 0
    unsigned char alias_checksum; // checksum for 8.3 alias
    unsigned char name5_10[12]; // 6 more characters in name
    unsigned char start[2];     // starting cluster number
    unsigned char name11_12[4]; // last 2 characters in name
};
```

If the layout of the slots looks a little odd, it's only because of Microsoft's efforts to maintain compatibility with old software. The slots must be disguised to prevent old software from panicking. To this end, a number of measures are taken:

1. The attribute byte for a slot directory entry is always set to 0x0f. This corresponds to an old directory entry with attributes of "hidden", "system", "read-only", and "volume label". Most old software will ignore any directory entries with the "volume label" bit set. Real volume label entries don't have the other three bits set.
2. The starting cluster is always set to 0, an impossible value for a DOS file.

Because the extended FAT system is backward compatible, it is possible for old software to modify directory entries. Measures must be taken to ensure the validity of slots. An extended FAT system can verify that a slot does in fact belong to an 8.3 directory entry by the following:

1. Positioning. Slots for a file always immediately proceed their corresponding 8.3 directory entry. In addition, each slot has an id which marks its order in the extended file name. Here is a very abbreviated view of an 8.3 directory entry and its corresponding long name slots for the file "My Big File.Extension which is long".

```
<proceeding files...>
<slot #3, id = 0x43, characters = "h is long">
<slot #2, id = 0x02, characters = "xtension whic">
<slot #1, id = 0x01, characters = "My Big File.E">
<directory entry, name = "MYBIGFIL.EXT">
```

Note

Note that the slots are stored from last to first. Slots are numbered from 1 to N. The Nth slot is marked with 0x40 to mark it as the last one.

2. Checksum. Each slot has an `alias_checksum` value. The checksum is calculated from the 8.3 name using the following algorithm:

```
for (sum = i = 0; i < 11; i++) {
    sum = (((sum&1)<<7)|((sum&0xfe)>>1)) + name[i]
}
```

3. If there is free space in the final slot, a Unicode `NULL` (0x0000) is stored after the final character. After that, all unused characters in the final slot are set to Unicode 0xFFFF.

Finally, note that the extended name is stored in Unicode. Each Unicode character takes either two or four bytes, UTF-16LE encoded.