NAME

mount_unionfs - mount union file systems

SYNOPSIS

mount_unionfs [**-b**] [**-o** *options*] *directory uniondir*

DESCRIPTION

The **mount_unionfs** utility attaches *directory* above *uniondir* in such a way that the contents of both directory trees remain visible. By default, *directory* becomes the *upper* layer and *uniondir* becomes the *lower* layer.

The options are as follows:

-b Deprecated. Use **-o below** instead.

-o Options are specified with the **-o** flag followed by an option. The following options are available:

below Inverts the default position, so that *directory* becomes the lower layer and *uniondir* becomes the upper layer. However, *uniondir* remains the mount point.

copymode = **traditional** | **transparent** | **masquerade**
Specifies the way to create a file or a directory in the upper layer automatically when needed. The **traditional** mode uses the same way as the old unionfs for backward compatibility, and **transparent** duplicates the file and directory mode bits and the ownership in the lower layer to the created file in the upper layer. For behavior of the **masquerade** mode, see *MASQUERADE MODE* below.

whiteout = **always** | **whenneeded**
Specifies whether whiteouts should always be made in the upper layer when removing a file or directory or only when it already exists in the lower layer.

udir=*mode*
Specifies directory mode bits in octal for **masquerade** mode.

ufile=*mode*
Specifies file mode bits in octal for **masquerade** mode.

gid=*gid*
Specifies group for **masquerade** mode.

uid=*uid*
Specifies user for **masquerade** mode.

To enforce file system security, the user mounting a file system must be superuser or else have write permission on the mounted-on directory. In addition, the *vfs.usermount* sysctl(8) variable must be set to 1 to permit file system mounting by ordinary users. However, note that **transparent** and **masquerade** modes require *vfs.usermount* to be set to 0 because this functionality can only be used by superusers.

Filenames are looked up in the upper layer and then in the lower layer. If a directory is found in the lower layer, and there is no entry in the upper layer, then a *shadow* directory will be created in the upper layer. The ownership and the mode bits are set depending on the **copymode** option. In **traditional** mode, it will be owned by the user who originally did the union mount, with mode 0777 ("rwxrwxrwx") modified by the umask in effect at that time.

If a file exists in the upper layer then there is no way to access a file with the same name in the lower layer. If necessary, a combination of loopback and union mounts can be made which will still allow the lower files to be accessed by a different pathname.

Except in the case of a directory, access to an object is granted via the normal file system access checks. For directories, the current user must have access to both the upper and lower directories (should they both exist).

Requests to create or modify objects in *uniondir* are passed to the upper layer with the exception of a few special cases. An attempt to open for writing a file which exists in the lower layer causes a copy of the *entire* file to be made to the upper layer, and then for the upper layer copy to be opened. Similarly, an attempt to truncate a lower layer file to zero length causes an empty file to be created in the upper layer. Any other operation which would ultimately require modification to the lower layer fails with EROFS.

The union file system manipulates the namespace, rather than individual file systems. The union operation applies recursively down the directory tree now rooted at *uniondir*. Thus any file systems which are mounted under *uniondir* will take part in the union operation. This differs from the **union** option to mount(8) which only applies the union operation to the mount point itself, and then only for lookups.

MASQUERADE MODE

When a file (or a directory) is created in the upper layer, the **masquerade** mode sets it the fixed access mode bits given in **ufile** (for files) or **udir** (for directories) option and the owner given in **udir** and **gid** options, instead of ones in the lower layer. Note that in the **masquerade** mode and when owner of the file or directory matches one specified in **uid** option, only mode bits for the owner will be modified.

More specifically, the file mode bits in the upper layer will be (mode in the lower layer) OR (mode given in **ufile** AND 0700), and the ownership will be the same as one in the lower layer.

The default values for **ufile**, **udir**, **uid**, and **gid** are as follow:

- If none of **ufile** and **udir** were specified, access mode bits in the mount point will be used.
- If none of **uid** and **gid** were specified, ownership in the mount point will be used.
- If one of **udir** or **ufile** is not specified, the value of the other option will be used.
- If one of **uid** or **gid** is not specified, the value of the other option will be used.

EXAMPLES

The commands

```
mount -t cd9660 -o ro /dev/cd0 /usr/src
mount -t unionfs -o noatime /var/obj /usr/src
```

mount the CD-ROM drive */dev/cd0* on */usr/src* and then attaches */var/obj* on top. For most purposes the effect of this is to make the source tree appear writable even though it is stored on a CD-ROM. The **-o noatime** option is useful to avoid unnecessary copying from the lower to the upper layer.

The commands

```
mount -t cd9660 -o ro /dev/cd0 /usr/src
chown 2020 /usr/src
mount -t unionfs -o noatime -o copymode=masquerade -o uid=builder \
   -o udir=755 -o ufile=644 /var/obj /usr/src
```

also mount the CD-ROM drive */dev/cd0* on */usr/src* and then attaches */var/obj* on top. Furthermore, the owner of all files and directories in */usr/src* is a regular user with UID 2020 when seen from the upper layer. Note that for the access mode bits, ones in the lower layer (on the CD-ROM, in this example) are still used without change. Thus, write privilege to the upper layer can be controlled independently from access mode bits and ownership in the lower layer. If a user does not have read privilege from the lower layer, one cannot still read even when the upper layer is mounted by using **masquerade** mode.

The command

```
mount -t unionfs -o noatime -o below /sys $HOME/sys
```

attaches the system source tree below the *sys* directory in the user's home directory. This allows individual users to make private changes to the source, and build new kernels, without those changes

becoming visible to other users. Note that the files in the lower layer remain accessible via */sys*.

SEE ALSO

intro(2), mount(2), unmount(2), fstab(5), mount(8), mount_nullfs(8)

HISTORY

The **mount_null** utility first appeared in 4.4BSD. It was renamed to **mount_unionfs** in FreeBSD 5.0.

The **-r** option for hiding the lower layer completely was removed in FreeBSD 7.0 because this is identical to using mount_nullfs(8).

AUTHORS

In FreeBSD 7.0, Masanori OZAWA <*ozawa@ongs.co.jp*> reimplemented handling of locking, whiteout, and file mode bits, and Hiroki Sato <*hrs@FreeBSD.org*> wrote about the changes in this manual page.

BUGS

THIS FILE SYSTEM TYPE IS NOT YET FULLY SUPPORTED (READ: IT DOESN'T WORK) AND USING IT MAY, IN FACT, DESTROY DATA ON YOUR SYSTEM. USE AT YOUR OWN RISK.

This code also needs an owner in order to be less dangerous - serious hackers can apply by sending mail to <*freebsd-fs@FreeBSD.org*> and announcing their intent to take it over.

Without whiteout support from the file system backing the upper layer, there is no way that delete and rename operations on lower layer objects can be done. EOPNOTSUPP is returned for this kind of operations as generated by VOP_WHITEOUT() along with any others which would make modifications to the lower layer, such as chmod(1).

Running find(1) over a union tree has the side-effect of creating a tree of shadow directories in the upper layer.

The current implementation does not support copying extended attributes for acl(9), mac(9), or so on to the upper layer. Note that this may be a security issue.

A shadow directory, which is one automatically created in the upper layer when it exists in the lower layer and does not exist in the upper layer, is always created with the superuser privilege. However, a file copied from the lower layer in the same way is created by the user who accessed it. Because of this, if the user is not the superuser, even in **transparent** mode the access mode bits in the copied file in the upper layer will not always be the same as ones in the lower layer. This behavior should be fixed.